Henrik Blunck

Modelle und Algorithmen für mobile Datenobjekte und Umgebungen

Münster
- 2006 -

Fach Informatik

Modelle und Algorithmen für mobile Datenobjekte und Umgebungen

Inaugural-Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften im Fachbereich
Mathematik und Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von

Henrik Blunck
aus Lübeck

- 2006 -

Dekan:	Prof. Dr. Klaus H. Hinrichs
Erster Gutachter:	Prof. Dr. Klaus H. Hinrichs
Zweiter Gutachter:	Prof. Dr. Jan Vahrenhold
Tag der mündlichen Prüfung:	27.10.2006
Tag der Promotion:	

Inhaltsverzeichnis

Kurzfassung

In den letzten Jahren hat das Management mobiler Objekte und ihrer Bewegungsdaten sowie das Datenmanagement in mobilen Umgebungen an Bedeutung gewonnen, bedingt durch wachsende Anwendungsfelder wie geographische Informationssysteme, Flottenmanagement, Verkehrsüberwachung und -führung und lokationsbasierte Dienste in mobilen Kommunikationssystemen.

Im ersten Teil dieser Arbeit werden zunächst Anforderungen an die Repräsentation der Bewegungen mobiler Objekte zum Zwecke der Datenhaltung und -verarbeitung beschrieben und anschließend eine Modellierung eines Rahmenwerks vorgestellt, das diverse Repräsentationskonzepte für Bewegungen integriert, um auch verschieden charakterisierte Bewegungen jeweils realistisch darstellen zu können. Es wird zudem das Ziel der Abgeschlossenheit eines Typsystems für zeitvariante räumliche Daten gegen zeitvariante Verarbeitungen von Bewegungen verfolgt und ein Lösungsvorschlag hierzu vorgestellt. Des Weiteren wird die Realisierbarkeit einer umfassenden, effizienten und verlässlichen Anfragebearbeitung, die auf verschiedensten Repräsentationen von Bewegungen operieren kann, untersucht und eine entsprechende Realisierung in das erwähnte Rahmenwerk integriert.

Im zweiten Teil dieser Arbeit wird die speichereffiziente Verarbeitung von geometrischen Daten betrachtet, wie sie in mobilen Umgebungen wie Sensornetzwerken oder mobilen Kommunikationssystemen von Nutzen sein kann, da in solchen Umgebungen die Datenverarbeitung teilweise von ressourcenbeschränkten Kleinstrechnern durchzuführen ist. Konkret werden optimal oder nahezu optimal speicher- und im erwarteten Fall laufzeiteffiziente Algorithmen für Probleme des Geraden-Schätzens vorgestellt, die zur Aggregation von geometrisch interpretierbaren und insbesondere von zeitvarianten Daten verwendet werden können. Des Weiteren werden Algorithmen für die Berechnung der Maximamengen und -schichten von Punktmengen präsentiert, die eine optimale asymptotische Laufzeit- und Speicherbedarfskomplexität aufweisen und die für prioritätsgesteuerte Datenselektion und -gruppierung insbesondere auf mobilen Endgeräten eingesetzt werden können.

Danksagung

Ich danke Prof. Dr. Klaus H. Hinrichs für die Möglichkeit zur Mitarbeit in seiner Arbeitsgruppe sowie für sein Vertrauen und seine stete Unterstützung und Hilfestellung bei meinem Dissertationsvorhaben.

Ich danke Prof. Dr. Jan Vahrenhold und Dr. Ludger Becker für zahlreiche Anregungen und die sehr gute und bereichernde Zusammenarbeit sowie für viel Lernenswertes und nach wie vor Richtungsweisendes, das ich aus dieser mitnehmen durfte. Prof. Dr. Jan Vahrenhold danke ich außerdem für die Bereitschaft, sich meiner Dissertation auch als zweiter Gutachter anzunehmen.

Den weiteren Mitgliedern unserer Arbeitsgruppe Evelyn Egelkamp, Michael Jacob, Dr. Frank Steinicke, Dr. Timo Ropinski und Jennis Meyer-Spradow danke ich für das stets erfreuliche Mitaneinander am Arbeitsplatz und insbesondere für ihre Unterstützung auch über den beruflichen Auftrag hinaus.

Auch danke ich den Studenten, mit denen zusammenzuarbeiten ich die Gelegenheit und das Vergnügen hatte und die auch die Forschung unserer Arbeitsgruppe insbesondere im Bereich der Modellierung mobiler Objekte bereichert haben. Namentlich herausheben möchte ich hierbei Katrin Boege, Anja Dunkelmann, Stephanie Jabs, Philipp Kegel, Björn Kopiske, Dennis Kötterheinrich, Martin Lensing, Iris Puke, Thorsten Schmolke sowie Joëlle Sondern.

An dieser Stelle möchte ich mich auch bei meinen ehemaligen Komilitonen und Kollegen im Geiste, insbesondere bei Wolfram Urich, Christof Henseler und Jens Reinel, für Beistand und Anregungen danken. Mein Dank gilt auch Familie Bolsmann, die mich während meiner Promotionszeit oft aufopferungsvoll unterstützt hat, und im Besonderen Ronja, die mir in dieser akademischen Lehrzeit noch viel Wesentliches beibringen konnte.

Vor allem gilt mein persönlicher Dank meinen Eltern Beate und Hans-Joachim, die einen nicht wegzudenkenden Anteil am erfolgreichen Abschluss meiner Ausbildung und meiner Dissertation haben, der mir ohne ihren Rückhalt und ihre unbedingte Unterstützung so nicht möglich gewesen wäre.

Kapitel 1

Einleitung

In den letzten Jahren hat das Management mobiler Objekte und ihrer Bewegungsdaten sowie das Datenmanagement in mobilen Umgebungen an Bedeutung gewonnen, bedingt durch wachsende Anwendungsfelder wie geographische Informationssysteme, Flottenmanagement, Verkehrsüberwachung und -führung oder lokationsbasierte Dienste in mobilen Kommunikationssystemen. Die Entwicklung im Mobilitätsmanagement profitiert von Ergebnissen aus etablierten Forschungsbereichen wie denen der räumlichen, der temporalen sowie der mobilen Datenbanken sowie der algorithmischen Geometrie. Dennoch existieren nach wie vor offene Fragestellungen, betreffend unter anderem die angemessene Repräsentation von Bewegung in Anwendungsszenarien wie den oben genannten; zudem bietet das Datenmanagement in mobilen Umgebungen noch algorithmische Herausforderungen — insbesondere da dieses im Allgemeinen durch ressourcenbeschränkte Computer zu erfolgen hat. Kürzlich wurden von prominenten Autoren aus den Forschungsbereichen der räumlichen-temporalen Datenbanken [RHE+04] sowie der Algorithmik [AGE+02] zwei Übersichten publiziert, die offene Fragestellungen bzw. relevante Gegenstände weiterer Forschung aufführen, zu denen auch die beiden Schwerpunkte dieser Arbeit zählen. Diese Arbeit folgt zudem der Anregung von Roddick *et al.*, sich in Dissertationsvorhaben auf einige dieser offenen Fragestellungen zu fokussieren und diese für sich genommen zu betrachten, anstatt eine komplette monolithische Modellierung oder Implementierung für das Management räumlich-temporaler bzw. mobiler Daten in den Vordergrund einer Dissertation zu stellen.

Repräsentation und Verarbeitung von Bewegungsdaten Der erste Teil dieser Arbeit, bestehend aus den Kapiteln 2 und 3, behandelt die adäquate Repräsentation der Bewegungsverläufe mobiler Objekte. Die zu betrachtenden sich bewegenden Objekte entsprechen je nach Anwendungskontext Fahr- oder Flugzeugen, Tieren oder Personen oder auch Kleinstpartikeln wie Molekülen in physikalischen Simulationen; zu repräsentieren sind — wiederum abhängig vom Anwendungskontext — aufgezeichnete, gerade im Geschehen begriffene oder geplante Bewegungen. Zwar können Objekte, deren Position im Raum sich nur in diskreten Schritten ändert, durch bekannte räumlich-temporale Erweiterungen von Datenbanksystemen repräsentiert werden; zur Repräsentation von sich kontinuierlich bewegenden Objekten jedoch sind traditionelle Datenbankmodelle wenig geeignet, da sie von der

Annahme ausgehen, dass ein Datum — wie die Position eines mobilen Objektes — konstant bleibt, wenn keine explizite Änderung vorgenommen wird. Neuere Anwendungen wie die oben genannten erfordern jedoch eine (implizite) Beschreibung der stetigen Positionsänderungen eines sich bewegenden Objektes durch Trajektorien, d. h. durch stetige Funktionen. Obwohl bereits zahlreiche entsprechende Datenmodelle vorgestellt wurden, unterstützen diese zumeist nur Trajektorien in Form von stückweise linearen Funktionen. Eine solche Darstellung der Bewegungen mobiler Objekte ist für viele Anwendungsszenarien nicht hinreichend realistisch, unter anderem da Geschwindigkeit und Richtung als nur in diskreten Schritten sprunghaft veränderlich (und ansonsten konstant) abgebildet werden.

In beiden der genannten Übersichten von Agarwal *et al.* [AGE+02] und Roddick *et al.* [RHE+04] zu relevanten Forschungsgegenständen im Bereich der Bewegungsverarbeitung wird die Entwicklung realistischer Bewegungsrepräsentationen für notwendig erachtet und auch in in weiteren Arbeiten wird dies angeregt [CR99a, CR99b, PJ99, YdC95]. Die Verwendung von realistischeren statt nur stückweise linearen Trajektorien impliziert jedoch einen höheren Berechnungsaufwand für Konstruktion und Verarbeitung von solchen Bewegungsrepräsentationen. Daher wird in den genannten Übersichten zusätzlich eine diesbezügliche Evaluation neuer Repräsentationskonzepte eingefordert, auf deren Grundlage für eine jeweilige Anwendung ein geeignetes Konzept zur Bewegungsrepräsentation gewählt werden kann, wobei Realismus und Ressourcenaufwand gegeneinander abgewägt werden können.

In dieser Arbeit werden zunächst Anforderungen an die Repräsentation von Bewegungen zusammengefasst; in realen Anwendungsszenarien sind diese vielfältig und differieren je nach Art der darzustellenden Bewegung. Beispielsweise wird in der Flugverkehrssicherung von offizieller Stelle eine Datenbank gepflegt, in der die Bewegungscharakteristika von über 250 Flugzeugtypen vermerkt sind [Eur99, NPI+05]; für das Verkehrsmanagement speziell an Flughäfen wären zusätzlich zu diesen sich unterschiedlich bewegenden Objekten die Bewegungen auch von Bodenpersonal und Fahrzeugen zu modellieren. In dieser Arbeit werden in Kapitel 2 verschiedene Arten von Trajektorien vorgestellt und deren Eignung zur Repräsentation von Bewegungen mit verschiedenen, spezifischen Charakteristiken und in verschiedenen Anwendungsszenarien diskutiert. Es wird zudem ein erweiterbares Rahmenwerk vorgestellt, das die Darstellung von Bewegung durch diverse stetige Trajektorienarten und die Repräsentation beliebiger, aus Bewegungen abgeleiteter Informationen ermöglicht. Dieses Rahmenwerk ist als Erweiterung eines objektorientierten Geo-Datenbankkerns implementiert und modelliert für Trajektorien beliebigen Typs eine einheitliche minimale Schnittstelle: Trajektorien werden als stetige Funktionen $f(t)$ realisiert, die als Funktionalität nur die Auswertung zu beliebigen Zeitpunkten bereitzustellen haben. Beispielhaft wird für solche Auswertungen der Mehraufwand durch die Verwendung kubischer Splines im Vergleich zur Verwendung klassischer stückweise linearer Funktionen zur Repräsentation von Bewegungen sowohl theoretisch als auch an Hand praktischer Evaluationen diskutiert. Zudem wird das ebenfalls offene Problem behandelt, geeignete Repräsentationen auch für Informationen, die von Bewegungen abgeleitet werden, z.B. für die (zeitvariante) Distanz zweier sich bewegender Objekte, zu finden und in ein Modell für Bewegungsdaten (bzw. allgemeiner: für mobile Objekte) zu integrieren.

Die minimalistische Schnittstelle für die Modellierung von Trajektorien erlaubt zwar vielfältige adäquate Repräsentationen der realen Bewegungen verschiedenster mobiler Objekte, wirft jedoch die Frage auf, wie komplexere Operationen und Anfragetypen, die die Verarbeitung und Analyse von Bewegungen betreffen, effizient realisiert werden können. In Kapitel 3 dieser Arbeit wird deshalb beschrieben, wie ein erweiterbares Anfragesystem und unterschiedliche Operationen zur Bewegungsverarbeitung implementiert werden können, ohne dass hierfür die für Bewegungsrepräsentationen verwendete Trajektorienart einheitlich für alle zu betrachtenden mobilen Objekte festgelegt werden muss. In diesem System wird unter anderem von Bewegungsrestriktionen, wie Beschränkungen für Geschwindigkeit und Beschleunigung, die zu den modellierten Bewegungen bekannt sind, Gebrauch gemacht, um trotz der Universalität des Systems die Korrektheit und Effizienz der Anfragebearbeitung zu garantieren. An Hand von Beispielanfragen zur Erkennung von Kollisionen und kritischen Nähebeziehungen zwischen mobilen Objekten wird dieser Ansatz konkretisiert beschrieben und evaluiert. Abschließend wird seine Integration in das in Kapitel 2 vorgestellte Rahmenwerk vorgestellt.

Speichereffiziente Algorithmen für mobile Umgebungen Im zweiten Teil dieser Arbeit in den Kapiteln 4 bis 6 wird die ressourcenschonende Verarbeitung von Daten behandelt — ein Forschungsfeld, das in den genannten Übersichten von Agarwal *et al.* [AGE+02] und Roddick *et al.* [RHE+04] als relevant für die Weiterentwicklung des mobilen Datenmanagements herausgestellt wird. In Anwendungsszenarien wie lokationsbasierten Diensten für mobile Kommunikationssysteme oder der Datengewinnung und -verbreitung durch Sensornetzwerke ist auch die Verarbeitung von Daten dezentral auf mobilen, ressourcenbeschränkten Computern durchzuführen. Während Mobiltelefone und ähnliche mobile Geräte zwar inzwischen über wachsende Ressourcen verfügen, sind für Geräteklassen wie Sensor-Computer oder Kleinst-Nutzergeräte (*engl.: wearables*) die Miniaturisierung und ein geringer Energieverbrauch zentrale Zielsetzungen, so dass für diese generell eine optimale Ausnutzung auch geringer Ressourcen vorteilhaft ist.

Die Ressourcenbeschränkungen, denen solche Geräte unterliegen, betreffen den Energievorrat, die Bandbreite für die mobile Kommunikation, die Rechenleistung sowie nicht zuletzt den für Datenhaltung und -verarbeitung zur Verfügung stehenden Speicherplatz [VCdSdM03, KDM05, LM03, LHY+04]. In der Übersicht von Agarwal *et al.* [AGE+02] wird herausgestellt, dass es für den Einsatz in solchen Umgebungen speziell angepasste, d. h. ressourcenschonende, Algorithmen zu entwickeln gilt. In Kapitel 4 dieser Arbeit wird die Technik und Organisation von mobilen Kommunikationssystemen und insbesondere von Sensornetzwerken skizziert und ein Überblick über speichereffiziente Algorithmen gegeben, d. h. über Algorithmen, die eine optimale oder nahezu optimale Laufzeitkomplexität besitzen und zudem für Berechnungen möglichst wenig Speicher neben demjenigen, der die zu verarbeitenden Daten enthält, verwenden.

In den Kapiteln 5 und 6 werden algorithmische Aufgabenstellungen der Aggregation und der Gruppierung von räumlichen oder räumlich-temporalen Daten betrachtet, deren Bedeutung für die Datenverbreitung in Sensornetzwerken sowie für lokationsbasierte Dienste aufgezeigt wird. In Kapitel 5 wird die Aufgabenstellung

der Bestimmung einer Ausgleichsgeraden zu einer Menge von Punkten — und damit einem robusten Schätzer für diese Punktdaten — näher betrachtet. In Szenarien der mobilen Datenverarbeitung dient diese Aggregationsoperation insbesondere dem verlässlichen Rekonstruieren bzw. Extrapolieren von gradlinigen räumlichen Berandungen sowie auch von gradlinigen Verläufen von Bewegungen. In Kapitel 6 wird zunächst die Aufgabe behandelt, die Maxima (oder Minima) einer Punktmenge zu bestimmen, d. h. diejenigen Punkte der Menge, zu denen jeweils kein anderer Punkt der Menge existiert, der in allen Koordinaten größer ist. Für lokationsbasierte Dienste kann die Minimabestimmung beispielsweise eingesetzt werden, um für einen mobilen Nutzer genau die Hotels zu identifizieren, die keinem anderen Hotel hinsichtlich sowohl der Nähe zum Nutzer als auch hinsichtlich des Preises „unterboten“ werden. Anschließend wird die Gruppierung einer Punktmenge in Schichten von Maxima (oder Minima) betrachtet; hierbei bildet die Maximamenge der Punkte die oberste Schicht und alle weiteren Schichten bestimmen sich jeweils als die Maximamengen der noch nicht einer Schicht zugeordneten Punkte.

Alle genannten Aufgabenstellungen werden jeweils mit der Zielsetzung untersucht, laufzeiteffiziente Algorithmen bereitzustellen, die einen möglichst geringen Bedarf an Speicher neben demjenigen, der die zu verarbeitenden Daten enthält, aufweisen.

Kapitel 2

Modellierung mobiler Objekte und Repräsentation ihrer Bewegungen

In diesem Kapitel wird die Modellierung mobiler Objekte und insbesondere ihrer Bewegungen behandelt. Der Schwerpunkt liegt hierbei auf der in der Einleitung skizzierten Zielsetzung der realistischen Repräsentation von Bewegungen sowie von zeitvarianten Größen wie Distanzen, die von den Bewegungsdaten mobiler Objekte abzuleiten sind. Als Resultat dieser Betrachtungen wird ein Rahmenwerk vorgestellt, welches zur Erfüllung — auch in verschiedensten Anwendungsszenarien — dieser Zielsetzung bzw. dieser Anforderung an die Darstellung von Bewegungsdaten die Integration verschiedenster Trajektorienarten leistet [BBHV04].

2.1 Repräsentation von Bewegungen durch Trajektorien

Nach einem kurzen historischen Überblick über die Modellierung von Bewegungen in Datenbanken und anderen Systemen wird ausführlicher die Modellierung von Güting *et al.* [GBE+00] vorgestellt, welche für das in diesem Kapitel vorgestellte Rahmenwerk adaptiert wird. Des Weiteren werden Grundlagen für die Repräsentation von Bewegungen durch Trajektorien beschrieben und nachfolgend generelle sowie anwendungsspezifische Anforderungen an die Repräsentation von Bewegungen zusammengetragen.

2.1.1 Ein Überblick über Modellierungen von Bewegungsdaten

Räumliche und temporale Datenmodelle Es wurde bereits vielfach angemerkt [SWCD97, EGSV99, GBE+00, Mer05], dass für die Modellierung mobiler Objekte die Anwendbarkeit der Ergebnisse aus den bereits etablierten Forschungsbereichen der räumlichen und temporalen Datenbanken prüfenswert ist. Einen Überblick über frühe Arbeiten zu temporalen Datenbanken liefern Salzberg und Tsotras [ST99]; die hierin erwähnten Systeme bilden allerdings nur sich in diskreten Schritten vollziehende, zeitliche Entwicklungen ab. Diese geschieht durch Darstellung ei-

nes jeden Datenobjektes durch eine endliche Menge von Versionen bzw. Tupeln. Auch in den Bereichen räumlicher Datenbanken und Geo-Informationssysteme wurde versucht, zeitliche Entwicklungen der zu betrachtenden Daten zu reflektieren, was unter anderem zu Modellen wie dem von Armstrong vorgestellten *snapshot*-Modell [Arm88] führte. Worboys [Wor94] präsentierte ein Modell, in dem räumliche Datentypen mit Zeitstempel-Attributen versehen sind. Während diese Systeme eine Fusion der Forschungsergebnisse aus den Bereichen der räumlichen und der temporalen Datenbanken darstellen, wurde unter anderem von Erwig *et al.* [EGSV99] sowie von Hornsby und Egenhofer [HE00] ausgeführt, dass sich Bewegungen — als sich kontinuierlich ändernde Daten — nicht durch Zeitstempel- und Versions-basierte Datenhaltung, wie in den genannten, erweiterten räumlichen sowie den meisten temporalen Datenbanksystemen üblich, geeignet abbilden lassen.

Funktionsbasierte Datenmodelle In neueren Ansätzen wie dem von Yeh und de Cambray [YdC95] werden Bewegungen kontinuierlich, d. h. durch Funktionen von der Zeit in den Raum der Bewegung, repräsentiert. Die unendlich vielen Positionsänderungen eines mobilen Objektes lassen sich so implizit durch eine endlich darstellbare Funktionsbeschreibung speichern. Auch im Bereich der spezialisierten Datenbanken wurden räumlich-temporale Modelle entwickelt, die Bewegungsverläufe mobiler Objekte durch stetige zeitvariante Funktionen abbilden. Hier sind insbesondere die Arbeiten von Güting *et al.* [GBE+00] bzw. Erwig *et al.* [EGSV99] und Lema *et al.* [LFG+03] zu nennen, die im folgenden Abschnitt 2.1.2 detaillierter vorgestellt werden. Sistla und Wolfson [SW95] entwickelten die temporale Abfragesprache *Past Temporal Logic* (PTL), die Anfragen über Nähebeziehungen auf räumlichen, zeitvarianten Daten unterstützt. Sistla *et al.* [SWCD97] stellten später ein Datenmodell für sich bewegende Objekte vor und entwickelten, basierend auf PTL, die Sprache *Future Temporal Logic* (FTL). Diese Ansätze wurden später zu einem Datenmodell namens *Moving Objects Spatio-Temporal* (MOST) erweitert, welches die Basis des räumlich-temporalen Datenbanksystems DOMINO (*Databases fOr MovINg Objects tracking*) bildet und welches auch die Modellierung von Unsicherheiten bezüglich der Aktualität der Bewegungsdaten sowie von Kommunikationskosten zur Aktualisierung solcher beinhaltet [WCD+98, WSX+99, WXCJ98, WJS+99]. Trajcevski *et al.* [TWHC04] stellten eine hierzu nochmals erweiterte Modellierung vor, in der Bewegungen durch Trajektorien modelliert sind, die mit einem Unsicherheitsbereich umgeben werden können.

Während die genannten Datenmodelle ausschließlich stückweise lineare Funktionen für die Bewegungsbeschreibung verwenden, unterscheiden Erwig *et al.* [EGSV99] zumindest zwischen einem abstrakten und einem diskreten Modell: Während das abstrakte Modell die vom Anwender gewünschte Funktionalität bietet, wird vom diskreten Modell insbesondere seine Implementierbarkeit verlangt. Konkret lässt das abstrakte Modell von Erwig *et al.* beliebige Funktionen zur Bewegungsrepräsentation zu, das diskrete Modell erlaubt jedoch nur stückweise lineare Trajektorien. Für eine solche Einschränkung spricht, dass die Speicherung und Auswertung einer Trajektorie geringen Aufwand erfordert. Außerdem operieren die meisten Algorithmen und Indexstrukturen für räumlich-temporale Datenbanksysteme nur auf linearen Bewegungsbeschreibungen, siehe z.B. [AAV01, PJ01]; neuere algorithmi-

sche Entwicklungen wie kinetische Datenstrukturen [BG99] sind hingegen auch für nicht-lineare Trajektorien verwendbar, vergleiche auch die Ausführungen in Kapitel 3. Die Diskrepanz zwischen realer Bewegung und der Darstellung durch Polylinien wird in vielen Publikationen als Problem angesehen und eine Modellierung, die die Darstellung auch durch nicht-lineare Trajektorien erlaubt, unter anderem von Chomicki und Revesz [CR99b], Erwig *et al.* [EGSV99], Pfoser und Jensen [PJ01], Agarwal *et al.* [AGE+02] sowie von Roddick *et al.* [RHE+04] angeregt.

Constraint-basierte Datenmodelle Das Forschungsgebiet der Constraint-Datenbanken wurde mit dem Artikel von Kannellakis *et al.* [KKR90] begründet und entwickelte sich aus der Forschung im Bereich des logischen Programmierens mit *Constraints*, siehe Frühwirth und Abdennadher [FA97] für einen Überblick. Als Constraints werden hierbei logische Aussagen in Form von Gleichungen und Ungleichungen bezeichnet. Die Notation eines (punktförmigen) Tupels im relationalen Modell kann in Form einer Verknüpfung von Constraints, d. h. von einschränkenden Bedingungen zur Beschreibung einer (potentiell unendlichen) Menge von Datenbankeinträgen, verallgemeinert werden. Ein wesentlicher Vorteil dieser Art der Modellierung liegt in der kompakten Beschreibung von großen und eventuell unendlichen Datensätzen wie etwa für ausgedehnte räumliche Objekte und insbesondere auch für kontinuierliche Bewegungsbeschreibungen. Erste Arbeiten, die eine auf Constraints basierende Modellierung speziell für Bewegungsdaten vorstellen, stammen von Chomicki und Revesz [CR97, CR99b].

Des Weiteren bieten Constraint-Datenbanken die Option einer generischen und automatisierbaren Anfragebearbeitung, die auf dem Lösen von Ungleichungssystemen fußt und deren Korrektheit zudem streng mathematisch nachgewiesen werden kann. Diese Form der Anfragebearbeitung ist insbesondere unabhängig von der Semantik der durch Constraints dargestellten Daten anwendbar.[1] Allerdings resultiert aus dieser semantikfreien Modellierung eine fehlende Transparenz und intuitive Bedienbarkeit durch den Nutzer sowie die fehlende Typsicherheit einer semantisch typisierten Modellierung, wie sie mit dem im folgenden Abschnitt beschriebenen und auf der Notation abstrakter Datentypen basierenden Ansatz möglich ist. Aus diesen Gründen wurde versucht, das Konzept der Constraint-Datenbanksysteme mit semantischer Datenhaltung und -verarbeitung zu integrieren und insbesondere eine SQL-Erweiterung bereit zu stellen, die zur Abfrage der Inhalte von Constraint-Datenbanken genutzt werden kann. Eine weiterreichende Integration von Constraint- und ADT-Ansatz ist im prototypischen Constraint-Datenbanksystem DEDALE realisiert, das als Erweiterung des objektorientierten Datenbanksystems O2 implementiert und speziell für die Verarbeitung räumlicher und räumlich-temporaler Daten konzipiert ist [RSSG03, GLRS00, GRS98]. DEDALE ist allerdings nach unserem Wissenstand das einzige bereits in praktischen Anwendungen eingesetzte Constraint-Datenbanksystem und unterstützt als solches nur lineare Constraints und damit auch nur stückweise lineare Bewegungsrepräsentationen, vergleiche Rigaux *et al.* [RSSG03].

Das von Kuper *et al.* [KPL99] editierte Handbuch bietet einen Überblick über Aufbau, Möglichkeiten und Grenzen der auf dem Constraint-Ansatz beruhenden

[1] Auf die Anfragebearbeitung für durch Constraints dargestellte Daten wird in Abschnitt 3.6.3 genauer eingegangen.

Datenhaltung und -verarbeitung sowie über verschiedene Anfragesprachen für Constraint-Datenbanken. Eine Diskussion über die Auswirkungen der Verwendung nichtlinearer statt nur linearer Gleichungen auf die Behandlung von Ungleichungssystemen bieten Afrati *et al.* [ACGK94]. Kuijpers *et al.* [KKP00] stellen eine alternative Erweiterung der linearen Constraints um die arithmetische Operation der euklidischen Distanz vor und diskutieren ihre Auswirkung auf die in räumlichen Constraint-Datenbanken zu lösenden Ungleichungssysteme. Geerts [Gee04] präsentierte zudem einen ersten Vorschlag, wie sich Differentialgleichungen zur Beschreibung von Bewegungen in einem Constraint-Datenbanksystem einsetzen lassen und konzipierte eine entsprechende Anfragesprache. Su *et al.* [SXI01] präsentierten ein Constraint-Datenmodell speziell für mobile Objekte und eine hierzu passende und ebenfalls auf der Prädikatenlogik basierende Anfragesprache. Später wurde dieses Datenmodell von Mokhtar *et al.* [MSI02] adaptiert, um auch zeitvariante Ergebnisse spezieller Operationen wie der Distanz zweier mobiler Objekte zu modellieren. Chomicki und Revesz [CR99b] verwendeten zur Beschreibung von Bewegungen Matrizen und untersuchten die Abgeschlossenheit ihres Modells hinsichtlich verschiedener affin-linearer Transformationen.

Auch wenn die Einbindung von Operationsergebnissen in einigen der oben erwähnten Modelle für Bewegungsdaten unterstützt wird, so ist in keinem von diesen die Repräsentation der Ergebnisse von konkatenierten Operationen wie der iterierten Distanz möglich. Vielmehr wird die Abgeschlossenheit gegen (iterierte) Operationen wie der Distanz als ein offenes und relevantes Problem der Modellierung von Bewegungsdaten angesehen [CR99b, EGSV99, SXI01], welches in dieser Arbeit in Abschnitt 2.3 näher behandelt wird.

Die Modellierung von Bewegung und entsprechende Literatur wird speziell unter algorithmischen Aspekten auch in Abschnitt 3 betrachtet. Einen weiteren ausführlichen Überblick über Datenmodelle für Bewegungen sowie eine Kategorisierung dieser bietet auch Meratnia [Mer05, Kapitel 2].

2.1.2 Ein ADT-basiertes Datenmodell für sich bewegende Objekte

Die Modellierung von Erwig *et al.* [EGSV99] bzw. von Güting *et al.* [GBE+00], die später noch erweitert wurde [FGNS00, LFG+03], verwendet den ADT-Ansatz, in dem Daten und Funktionalitäten durch Definition abstrakter Datentypen beschrieben werden, vergleiche etwa Nievergelt und Hinrichs [NH93]. Güting *et al.* propagieren, wie erwähnt, eine separate Modellierung eines abstrakten, d. h. idealen, sowie eines diskreten, d. h. implementierbaren, Modells. Das abstrakte Modell von Güting *et al.* bildet den Ausgangspunkt für die Modellierung des hier vorgestellten Rahmenwerks. Letzteres kann als ein diskretes Modell aufgefasst werden, das anders als das diskrete Modell von Güting *et al.* [GBE+00] auch komplexere als nur stückweise lineare Funktionen für die Beschreibung von Bewegungen zulässt. Das von Güting *et al.* vorgestellte abstrakte Modell besitzt konstante Basistypen wie `int`, `real`, `string` und `bool`. Zusätzlich besitzt es räumliche Typen `point`$\langle d \rangle$,

`line`$\langle d \rangle$ und `region`$\langle d \rangle$, wobei d die Dimensionalität des Raumes angibt.[2] Generell existiert zu jedem Basis- und jedem räumlichen Datentyp `type` ein *geliftetes*, d. h. zeitvariantes, Pendant `mtype`. Eine Instanz von `mtype` stellt die zeitliche Entwicklung eines Objektes dar, welches durch eine Instanz von `type` modelliert ist. Ferner wird verlangt, dass jede für Instanzen zeitinvarianter Datentypen definierte Operation auch für Instanzen der korrespondierenden gelifteten Versionen dieser Datentypen anwendbar ist. Die Operation `distance` liefert für punktförmige Objekte beispielsweise nur dann einen konstanten numerischen Wert (vom Typ `real`), wenn beide Argumente Instanzen von `point`$\langle d \rangle$ sind; ist zumindest eines der Argumente Instanz des zeitvarianten Typs `mpoint`$\langle d \rangle$, so wird ein zeitvariantes Resultat (vom Typ `mreal`) zurückgegeben. Zusätzliche Operationen, die ausschließlich für zeitvariante Argumente anwendbar sind, sind beispielsweise `trajectory`, `derivative`, `speed`, `velocity` und `acceleration` [LFG+03]. Das Modell wird durch temporale Datentypen zur Darstellung von Zeitpunkten und Intervallen komplettiert.

Diese ADT-Modellierung des abstrakten sowie des diskreten Modells folgt — insbesondere mit der Gruppierung von Datentypen in Domänen (hier: Basis-, Raum- und temporale Domäne) sowie mit der Konstruktion gelifteter Versionen von Typen und Operationen — dem Ansatz der *second order signature*, siehe [Güt93]. Mit diesem Ansatz wurde bereits eine Algebra räumlicher Datentypen konzipiert, vergleiche Güting und Schneider [GS95], als deren Ergänzung die soeben beschriebene Modellierung anzusehen ist [ES99].

Wir erweitern das abstrakte Modell von Güting *et al.* um einen Konstruktor `mreal` $\longrightarrow$ `point`$\langle 1 \rangle$ und `point`$\langle a \rangle \times$ `point`$\langle 1 \rangle$ $\longrightarrow$ `point`$\langle a+1 \rangle$ bzw. in der iterativen Anwendung `point`$\langle a \rangle \times$`point`$\langle b \rangle$ $\longrightarrow$ `point`$\langle a+b \rangle$. Diese Konstruktoren erlauben in ihrer gelifteten Version insbesondere, Bewegungsrepräsentationen aus eindimensionalen, funktionalen Repräsentationen (des Typs `mreal`) zu erstellen. Die Motivation für diese Ergänzung des Modells wird in Abschnitt 2.3.1 ausgeführt.

Im diskreten Modell von Güting *et al.* [GBE+00] bzw. Lema *et al.* [LFG+03] werden Objekte vom Typ `line`$\langle d \rangle$ durch Polylinien, ausgedehnte Objekte vom Typ `region`$\langle d \rangle$ durch polygonale Regionen repräsentiert. In der Tat werden auch in den meisten praktischen Anwendungen solche Objekte durch eine endliche Menge von Raumpunkten beschrieben; eine weitere gebräuchliche Darstellung der Ausdehnung von Objekten ist die Kreis- bzw. Kugelform. Der Zustand eines solchen Objektes ist wiederum durch einen Raumpunkt sowie optional durch einen zeitvarianten Wert für den Radius zu modellieren. Auf Grund dieser Rückführbarkeit auf das Beschreiben von Raumpunkten liegt der Fokus in den folgenden Ausführungen auf der diskreten Realisierung des Datentyps `point`$\langle d \rangle$ und seiner gelifteten Version `mpoint`$\langle d \rangle$, d. h. auf der Repräsentation d-dimensionaler, sich bewegender Punkte bzw. ihrer Bewegungen, sowie auf dem zeitvarianten numerischen Datentyp `mreal`.

Die Konzeption eines abstrakten Datenmodells für sich bewegende Objekte motivieren Güting *et al.* mit einer Fokussierung auf die idealerweise im Datenmodell abzubildenden Eigenschaften realer Bewegungen, ohne sich von der Fragestellung der Realisierbarkeit eines solchen Modells einschränken zu lassen [EGSV99, GBE+00].

[2] Das abstrakte Modell, wie es bei Güting *et al.* [GBE+00] beschrieben wird, beschränkt sich auf zweidimensionale Räume und verzichtet daher auf eine Parametrisierung der Datentypen durch ihre Dimensionalität.

Im folgenden Abschnitt wird diskutiert, ob die von Güting *et al.* präsentierte Diskretisierung, die nur stückweise lineare Bewegungsrepräsentationen als Instanzen des Datentyps `mpoint`$\langle d \rangle$ zulässt, dem idealisierten Modell nahe genug kommt. In Abschnitt 2.2 werden dann alternative bzw. erweiterte Diskretisierungen des abstrakten Modells vorgestellt.

2.1.3 Anforderungen an die Repräsentation durch Trajektorien

Die Bewegungsbeschreibung eines Objektes wird in den meisten Anwendungsszenarien aus zu diskreten Zeitpunkten gültigen Positionsinformationen rekonstruiert, die etwa durch Radar-, Radio- oder GPS-basierte Trackingsysteme erfasst werden.[3] Ist statt einer beobachteten eine geplante Bewegung, d. h. ein „Flugplan“, zu repräsentieren, so ist auch dieser zumeist durch Angabe diskreter Informationen beschrieben.

Ein Konzept zur Repräsentation sich bewegender Objekte hat insbesondere die (Re-)Konstruktion einer Bewegung aus solchen diskreten Daten zur Aufgabe, wobei gewöhnlich deren Interpolation gefordert ist. Im Folgenden sei mit einem *definierenden Faktum* ein Paar $(t_i,\ f_i)$ bezeichnet, welches ausdrückt, dass eine Information (z.B. eine Positionsangabe) f_i für ein zu repräsentierendes Objekt zu einem Zeitpunkt t_i gültig ist. Die Repräsentation einer beobachteten oder geplanten Bewegung kann somit durch eine Menge von definierenden Fakten sowie durch eine geeignete Interpolationsfunktion beschrieben werden. Die *Trajektorie* eines sich bewegenden Objektes ist der Graph dieser Interpolationsfunktion in einem $(d+1)$-dimensionalen Raum, der aus einer zeitlichen und d räumlichen Dimensionen besteht.[4]

Die bei der Repräsentation zu berücksichtigen räumlichen Informationen f_i sind in den meisten Anwendungen Positionsangaben des Objektes, sie können jedoch auch (zusätzlich oder alternativ) Angaben zu Geschwindigkeit, Richtung und/oder Beschleunigung enthalten. Ein *Konzept zur Bewegungsrepräsentation* besteht aus einer oder mehreren Arten von Trajektorien, z.B. stückweise linearen Kurven oder kubischen Splines, einem Speicherformat für diese, einem Konstruktionsalgorithmus zur Erstellung einer Interpolationsfunktion sowie einem Algorithmus zu deren Auswertung. Ein solches Konzept hat die folgenden technischen Minimalanforderungen zu erfüllen:

Vollständigkeit des Konstruktionsalgorithmus: Für jede vorgegebene Menge von Fakten kann eine eindeutige Interpolationsfunktion generiert werden.

Vollständige Auswertbarkeit der Trajektorie: Eine konstruierte Interpolationsfunktion kann zu jedem Zeitpunkt innerhalb ihres zeitlichen Definitionsbereiches ausgewertet werden.

[3] Einen Überblick über Techniken zur Erfassung von Bewegungsinformationen bietet die Diplomarbeit von Schmolke [Sch04].

[4] Eine Modellierung der inhärenten Ungenauigkeit, die bei der Messung von Fakten sowie bei Rekonstruktion einer Bewegung aus diesen berücksichtigt werden kann, wird in dieser Arbeit nicht explizit beschrieben, siehe hierfür unter anderem [TWHC04, WJS+99, WCD+98, PT01, PJ99, DG04b, Bli00, BHKR05].

Abrufbarkeit der definierenden Fakten: Alle die Interpolationsfunktion definierenden Fakten sind abrufbar.

Als *Repräsentationstyp* wird fortan eine Art von Trajektorien zusammen mit einem Speicherformat, das von (einem oder mehreren) Repräsentationskonzepten generiert bzw. verwendet wird, bezeichnet. Im Folgenden werden weitere Anforderungen an die Repräsentation von Bewegung genauer erläutert.

Auswertbarkeit

Die obige Forderung nach Auswertbarkeit ist je nach Einsatzbereich des Systems zur Bewegungsrepräsentation zu differenzieren. In Echtzeitanwendungen sind Bewegungen nicht zwingend über einen zeitlichen Bereich fixiert repräsentiert, sondern werden nur zum Zeitpunkt ihres Geschehens, d. h. *online*, beobachtet und gleichzeitig analysiert und verarbeitet. Im Folgenden unterscheiden wir dementsprechend Anwendungen danach, ob Bewegungsrepräsentationen nur für spezielle Argument-Zeitpunkte auswertbar sind.

Echtzeitanwendungen: Diskrete Daten über den Status eines sich bewegenden Objektes können nur für den aktuellen Zeitpunkt t_{act}, an dem die Auswertung angestoßen wird, — und folglich auch nur in chronologischer Reihenfolge — abgerufen werden.

Retrospektive Anwendungen: Diskrete Daten zum Status eines sich bewegenden Objektes können für beliebige Zeitpunkte abgerufen werden, zu denen die Repräsentation der Bewegung definiert ist.

Zur Behandlung von sich bewegenden Objekten in Echtzeitanwendungen ist zahlreiche Literatur erschienen, vergleiche etwa [WCD+98, PXK+02, HAFG95]. Rein retrospektive Anwendungen umfassen die rückblickende Analyse von aufgezeichneten Bewegungen sowie die Planung zukünftiger Bewegungen.

Größenbeschränktheit

Eine weitere Anforderung betrifft das Speicherformat bzw. den Speicheraufwand für Trajektorien sowie für zeitvariante Verarbeitungen von Trajektorien wie der Distanz zweier sich bewegender Objekte.

Definition 2.1.1 Für ein sich bewegendes Objekt mp bezeichne $|F(mp)|$ die Anzahl der diese Bewegung definierenden Fakten, f_{mp} die Interpolationsfunktion, die mp repräsentiert, und $|f_{mp}|$ ihre *Repräsentationsgröße*, d. h. der maximal zulässige Speicherbedarf für die (interne und system-spezifische) Repräsentation von f_{mp}.

Es kann ratsam sein, bei der Genauigkeit einer Repräsentation Abstriche zu machen, um einen beschränkten Speicherbedarf zu gewährleisten. Eine Repräsentation von beliebiger Genauigkeit würde im Allgemeinen zu einem beliebig hohen Berechnungsaufwand führen, zum anderen würde sich auf Grund des unvorhersehbaren

Speicherbedarfs der Datenobjekte auch die Organisation des Datenlayouts und des Datenzugriffs erschweren.

Durch Ergebnisse der *Fourier-Theorie* ist belegt, dass jede stetige Kurve sich beliebig genau durch ihre *Fourier-Reihe* approximieren lässt, die sich wie folgt ergibt [Chu92]:

$$f(t) = \frac{a}{2} + \sum_{j=1}^{m} \left(a_j \cdot \cos(j \cdot t) + b_j \cdot \sin(j \cdot t)\right), \text{ für } a, a_j, b_j \in \mathbb{R},\ 1 \leq j \leq m.$$

Der Speicheraufwand für die Repräsentation der Kurve f kann durch die Anzahl der zu speichernden Terme ihrer Fourier-Reihe angepasst werden. Analoges gilt für andere Approximationskonzepte wie *Taylor-Entwicklungen* [For04] bzw. polynomiale Interpolationen, die als Interpolationsfunktionen Linearkombination von Monomen, d. h. von Potenzen von t, liefern (anstatt von parametrisierten Sinus- und Kosinusfunktionen wie bei der Fourier-Interpolation).

Cao *et al.* [CWT03] betrachten Algorithmen zur Kompression von stückweise linearen Trajektorien. Diese Algorithmen wählen aus den zu einer Bewegung bekannten Fakten nur die relevantesten zur Interpolation durch eine (somit komprimierte) Trajektorie aus. Cao *et al.* evaluieren ferner die Auswirkungen der Kompression auf Anfrageergebnisse an bzw. Verarbeitungen von Bewegungen. Eine Verallgemeinerung der Resultate von Cao *et al.* auf nicht stückweise lineare Trajektorien sowie auf eine erweiterte Menge an Anfragetypen und Verarbeitungen von Bewegungen ist in der Diplomarbeit von Dunkelmann [Dun05] untersucht worden. Der Fokus in diesem Kapitel liegt nicht auf der Kompression von Bewegungsrepräsentationen: Wir abstrahieren hiervon, indem wir annehmen, dass alle bei einer Konstruktion übergebenen definierenden Fakten zu interpolieren sind; somit ergibt sich die Größenordnung des Speicherbedarfs einer Trajektorie als linear in der Anzahl der übergebenen Fakten.[5] Um das in der Einleitung und in Abschnitt 2.1.1 motivierte Ziel der Abgeschlossenheit des Datenmodells unter Verarbeitungen von Bewegungen zu erfüllen, sei gefordert, dass der Speicherbedarf für Ergebnisse von Verarbeitungen zeitvarianter Daten durch den Speicherbedarf der zu verarbeitenden zeitvarianten Daten selbst beschränkt werden kann.

Beschränkung des Speicherbedarfs: Die (system-spezifische) Repräsentationsgröße $|f_{mp}|$ einer Trajektorie (bzw. ihrer Interpolationsfunktion f_{mp}) eines sich bewegenden Objektes mp sei durch die Anzahl $|F(mp)|$ der sie definierenden Fakten linear beschränkt, d. h. es wird gefordert: $|f_{mp}| \leq c \cdot |F(mp)|$, wobei $c > 1$ eine globale Konstante sei.
Des Weiteren wird für die Anwendung einer Operation op gefordert, dass die Repräsentationsgröße ihres Ergebnisses durch die unter allen Operanden maximale Repräsentationsgröße beschränkt ist, d. h. für eine auf Operanden $mp_1, \ldots, mp_n$

[5] Um dies nachzuvollziehen, ist nur zu beachten, dass der Konstruktionsalgorithmus eines Repräsentationskonzeptes als eindeutig vorausgesetzt wird. Somit kann eine jede Trajektorie implizit durch die Menge der sie definierenden Fakten sowie eine Referenz auf den Konstruktionsalgorithmus vorgehalten werden. Zudem existiert für jedes folgend vorgestellte Repräsentationskonzept ein Format für ein alternative, *explizite* Speicherung von Interpolationsfunktionen, das ebenfalls obiger Aussage zur Beschränktheit der Repräsentationsgröße gerecht wird.

angewendete Operation *op* gilt dann: $|f_{\mathtt{op}(mp_1,\ldots,mp_n))}| \leq \max\{|f_{mp_1}|,\ldots,|f_{mp_n}|\}$

Lokalität und Aktualisierung

Eine weitere, optionale Anforderung an Konzepte zur Bewegungsrepräsentation ist die begrenzte Auswirkung einzelner definierender Fakten.

Lokalität der Repräsentation: Ein Repräsentationskonzept wird als *lokal* bezeichnet, wenn ein einzelnes definierendes Faktum sowie die Modifikation eines solchen sich auf eine Interpolationsfunktion nur in einem begrenzten Bereich ihres zeitlichen Definitionsbereiches auswirkt.

Die Eigenschaft der Lokalität ist in vielen Anwendungen wünschenswert: Zum einen kann gesichert werden, dass sich einzelne Messfehler nur auf einen begrenzten Zeitraum der Bewegungsrepräsentation auswirken; zusätzlich entspricht die Lokalität eines Repräsentationskonzeptes der Anschauung, dass die Repräsentation einer Bewegung in einem konkreten Zeitpunkt nicht durch Messungen zu wesentlich früheren oder späteren Zeitpunkten beeinflusst werden sollte. Zum anderen ist die Eigenschaft der Lokalität nützlich, wenn das Aktualisieren oder Modifizieren einer Bewegungsrepräsentation notwendig wird. In retrospektiven Anwendungen und insbesondere in der Bewegungsplanung sind eventuell einzelne definierende Fakten zu modifizieren (bzw. zu korrigieren), zu ergänzen oder zu entfernen, wobei eine Auswirkung auf die gesamte Trajektorie vermieden werden sollte. In online-Anwendungen, in denen auch bereits vollzogene (Teile von) Bewegungen repräsentiert werden, ist eine speziellere Form der Aktualisierung wesentlich: Eine Bewegungsrepräsentation sollte sich über ihren derzeitigen zeitlichen Definitionsbereich hinaus fortsetzen lassen; dabei ist wünschenswert, dass sich diese Fortsetzung in möglichst geringem Maße auf den bereits erstellten Teil der Repräsentation auswirkt.
Die oben erwähnte Fourier-Interpolation etwa weist keine Lokalität auf, da sich ein einzelner Fakt auf alle Koeffizienten der Fourier-Reihe auswirken kann und jeder Koeffizient sich wiederum auf die Interpolationsfunktion auf ihrem gesamten zeitlichen Definitionsbereich auswirkt. Allgemein gilt, dass die Forderung nach Lokalität für Interpolationsfunktionen in einem gewissen Widerspruch zur Forderung nach ihrer mehrfachen stetigen Ableitbarkeit bzw. — allgemeiner gesprochen — nach ihrer „Sanftheit“ steht. Dieser Widerspruch wird in Kapitel 2.2 für geeignetere Repräsentationskonzepte präzisiert und teilweise aufgelöst.

Realitätsnähe von Repräsentationen

Ein zentrales Ziel für bzw. eine zentrale Anforderung an Repräsentation und Verarbeitung von Bewegungen ist die Bereitstellung von Repräsentationen, die Bewegungen realistischer als stückweise lineare Kurven darstellen können, wie auch in den Übersichten von Roddick *et al.* [RHE+04] und Agarwal *et al.* [AGE+02] erläutert wird. Ein nahezu universell gültiges Kriterium für die Realitätsnähe einer Bewegungsrepräsentation ist die korrekte Abbildung der physikalischen Eigenschaften von Bewegungen. So sollten zumindest die ersten beiden Ableitungen einer Repräsentation stetig sein, da ein reales, sich bewegendes Objekt (als Folgerung des

Trägheitsprinzips, vergleiche Daniel [Dan97]) weder seine Position noch seine Richtung oder Geschwindigkeit sprunghaft verändern kann.[6] Ein weiteres, insbesondere für Interpolationen von Bewegungen gültiges Kriterium ergibt sich aus probabilistischen Betrachtungen zu möglichen Aufenthaltsorten zu Zeitpunkten zwischen den definierenden Fakten. Eine Interpolation sollte sich insbesondere nicht zu weit und zu willkürlich von der linearen Verbindung der definierenden Fakten „entfernen".[7]
Neben den genannten Kriterien für den Realismus einer Bewegungsrepräsentation, die als weitestgehend allgemeingültig angenommen werden können, existieren in spezifischen Szenarien und Anwendungen weitere Kriterien. Ein Beispiel hierfür ergibt sich bei der Modellierung von Objekten, die sich ausschließlich auf einem geometrischen Netzwerk, wie einem Straßennetz, fortbewegen, siehe beispielsweise [VW01, DG04a, DG04b, Md03] für Vorschläge zur Modellierung solcher Bewegungen. In diesem und weiteren Anwendungsfällen lassen sich spezifische Anforderungen an Bewegungsrepäsentationen durch geometrische Eigenschaften, die letztere einzuhalten haben, modellieren.[8]

2.2 Repräsentationskonzepte und ihre Implementierung

Im Folgenden werden einige Repräsentationskonzepte vorgestellt, die sich für verschiedene (Gewichtungen von) Anforderungen an die Repräsentation von Bewegung eignen und somit für die Integration in das hier vorgestellte Rahmenwerk anbieten. Es wird zudem beschrieben, in welcher Weise sich diese geeignet implementieren lassen, so dass die resultierenden Interpolationsfunktionen realistische und effiziente Repräsentation für sich bewegende Objekte des Typs `mpoint`$\langle d \rangle$ liefern. Dabei wird insbesondere auf Konstruktion, Speicherung und Auswertung solcher Repräsentationen eingegangen.[9] Als einführendes Beispiel eines Repräsentationskonzeptes wird zunächst die stückweise lineare Interpolation betrachtet. Anschließend werden als komplexere Beispiele Repräsentationskonzepte vorgestellt, die auf der Interpolation durch Splines basieren. Abschließend werden weitere Repräsentationskonzepte skizziert und ihre Vor- und Nachteile für die Bewegungsdarstellung erörtert.

[6] Von dieser Betrachtung sei die Repräsentation von Bewegungen auf Quantenebene ausgeschlossen, siehe etwa Heisenberg [Hei30] für eine Erläuterung physikalischer Aspekte der Quantentheorie.

[7] Pfoser und Jensen [PJ99] bieten eine ausführliche probabilistische Analyse zu solchen Aufenthaltswahrscheinlichkeiten. Insbesondere leiten sie für geschwindigkeitsbeschränkte Bewegungen ein Datenmodell ab, das die möglichen Aufenthaltsorte der sich bewegenden Objekte (nach Wahrscheinlichkeit gewichtet) modelliert.

[8] Die Einhaltung solcher geometrischer Eigenschaften (auch Gestalt-Erhaltung genannt) wird häufig in CAD-Systemen für die Repräsentation geometrischer Gebilde gefordert und von einigen im Folgenden vorgestellten Interpolationstechniken auch unterstützt, siehe Abschnitt 2.2.3 und vergleiche Peña [Peñ99].

[9] Zur Vereinfachung der Notation wird im Folgenden implizit von eindimensionalen Repräsentationen ausgegangen. Die geschilderten Sachverhalte lassen sich auf d-dimensionale Trajektorien insofern übertragen, als dass diese sich dimensionsweise aus d eindimensionalen Interpolationsfunktionen zusammensetzen lassen, vergleiche die Beschreibung des entsprechenden Konstruktors in Abschnitt 2.1.2.

2.2.1 Stückweise lineare Interpolation

Sind nur diskrete Positionen eines mobilen Objektes bekannt, nicht jedoch Geschwindigkeiten, Richtungen oder Beschleunigungen, so kann seine Bewegung durch stückweise lineare Interpolation, d. h. durch eine Polylinie im $(d+1)$-dimensionalen Raum, repräsentiert werden. Die Menge der zeitlichen Stützstellen einer solchen Interpolation entspricht einer *Zerlegung* $t_0 < \ldots < t_n$ des Definitionsbereiches durch die definierenden Fakten.

Die stückweise lineare Interpolation ist, wie auch in der Einleitung ausgeführt, das meist verwendete Konzept zur Interpolation diskreter Punktdaten. Grund hierfür ist vor allem ihre einfache Handhabung bezüglich Konstruktion und Auswertung. Um eine stückweise lineare Trajektorie f an einem Punkt t auszuwerten, sind nur die beiden t zeitlich benachbarten, f definierenden Fakten (f_i, t_i) und (f_{i+1}, t_{i+1}) zu identifizieren. Die Position $f(t)$ des durch f repräsentierten Objektes zum Zeitpunkt t ergibt sich dann wie folgt:

$$f(t) = f_i \; + \; (f_{i+1} - f_i) \cdot \frac{t - t_i}{t_{i+1} - t_i} \tag{2.1}$$

Eine explizite Speicherung der Interpolationsfunktion f ist daher nicht notwendig, sofern die sie definierenden Fakten abrufbar sind. Neben der Auswertung sind auch weitere Analyseaufgaben für die stückweise lineare Interpolation einfach algorithmisch zu realisieren. So ergibt sich beispielsweise die konvexe Hülle der besuchten Raumpunkte einer Trajektorie durch die konvexe Hülle allein der Raumpositionen der definierenden Fakten. Dies impliziert auch, dass sich Eigenschaften einer Trajektorie wie Monotonie oder Positivität (hinsichtlich einer gewählten Richtung) durch Ansicht der Menge der definierenden Fakten erschließen.

Nachteilig an stückweise linearen Trajektorie f ist hingegen, dass die Ableitungen von f wenig aussagekräftig sind bzw. Bewegungen physikalisch inkorrekt abbilden: Die durch eine stückweise lineare Trajektorie repräsentierte Geschwindigkeit und Richtung ist offensichtlich konstant zwischen den Zeitpunkten von definierenden Fakten und an diesen Zeitpunkten selbst ändert sie sich sprunghaft. Die zweite Ableitung von f besitzt zu den Zeitpunkten definierender Fakten im Allgemeinen sogar Unendlichkeitsstellen und ist ansonsnten verschwindend und liefert hinsichtlich der Veränderung von Geschwindigkeit und Richtung keinerlei Information.

Interne Darstellung der Repräsentation

Wie erwähnt besteht das geeignetste Speicherformat für eine stückweise lineare Trajektorie f allein aus den f bedingenden Fakten (t_i, f_i); weitere Informationen müssen nicht vorgehalten werden. Diese implizite Darstellung der Interpolationsfunktion verursacht keine zusätzlichen Speicherkosten: Der Speicherbedarf für die bedingenden Fakten ist entsprechend der Forderung in Abschnitt 2.1.3 nach deren Abrufbarkeit nicht zu umgehen — unabhängig von der kontinuierlichen Repräsentation der Bewegung.

Aktualisieren der Repräsentation

Die stückweise lineare Interpolation weist eine für eine stetige Interpolation größtmögliche Lokalität (vergleiche Kapitel 2.1.3) auf, wie an der Gleichung 2.1 zu ersehen ist: In einem konkreten, aber beliebigen Zeitpunkt wird eine stückweise lineare Trajektorie nur durch die beiden zeitlich benachbarten Fakten beeinflusst; entsprechend ist der Einflussbereich eines einzelnen Faktums auf die beiden angrenzenden Intervalle der Zerlegung beschränkt.
Eine Modifikation einer stückweise linearen Bewegungsrepräsentation durch Ergänzen, Löschen oder Abändern eines definierenden Faktums wirkt sich ausschließlich auf die unmittelbare (zeitliche) Nachbarschaft des Faktums aus: Soll ein Faktum (t, f), das zeitlich zwischen zwei Fakten (t_i, f_i) und (t_{i+1}, f_{i+1}) liegt, der Repräsentation durch f hinzugefügt werden, so ist nur das Segment $[(t_i, f_i), (t_{i+1}, f_{i+1})]$ (im $(d+1)$-dimensionalen Raum) durch zwei Segmente $[(t_i, f_i), (t, f)]$ und $[(t, f), (t_{i+1}, f_{i+1})]$ zu ersetzen. Das Löschen eines Faktums (t_i, f_i) entspricht dem inversen Vorgang. Das Abändern eines Faktums (t_i, f_i) wirkt sich nur auf die benachbarten Segmente $[(t_{i-1}, f_{i-1}), (t_i, f_i)]$ und $[(t_i, f_i), (t_{i+1}, f_{i+1})]$ aus, sofern der modifizierte Wert von t_i nach wie vor zwischen t_{i-1} und t_{i+1} liegt. Andernfalls kann das Modifizieren von (t_i, f_i) wie ein Entfernen des alten und ein anschließendes Einfügen des modifizierten Faktums behandelt werden.
Insbesondere bleibt das Fortführen einer stückweise linearen Bewegungsrepräsentation — über den bisherigen zeitlichen Definitionsbereich $[t_0, t_n]$ hinaus — ohne Auswirkung auf die Trajektorie in ihrem bisherigen Definitionsbereich.

2.2.2 Splines

Um realistische und stetige Änderungen von Richtung und Geschwindigkeit darzustellen, können zur Bewegungsrepräsentation Splines höheren Grades verwendet werden. Splines werden in der geometrischen Modellierung, z.B. in CAD-Systemen, und in der Computergrafik häufig eingesetzt, um „sanfte“ Kurven darzustellen, vergleiche beispielsweise Bartels *et al.* [BBB87]. Neben stetigen ersten und zweiten Ableitungen besitzen interpolierende kubische Spline-Kurven eine nützliche Balance-Eigenschaft: Sie minimieren annähernd (unter allen interpolierenden Kurven) die durchschnittliche quadratische Krümmung [Wer92]. Im Kontext der Bewegungsrepräsentation bedeutet dies, dass Änderungen von Geschwindigkeit und Richtung minimiert und über den zeitlichen Verlauf der Bewegung „gleichmäßig“ verteilt werden. Spline-Kurven oszillieren insbesondere weniger als andere sanfte Interpolationskurven, ändern aber gleichzeitig ihre Krümmung auch nicht so abrupt wie Polylinien.

Hintergrund: B-Splines und Spline-Interpolation

Folgend sind einige mathematische Grundlagen der Spline-Interpolation und der Darstellung durch B-Splines zusammengefasst (vergleiche auch de Boor [dB78] oder Nürnberger [Nür78]), um anschließend die Implementierung von Konstruktions- und Auswertungsalgorithmen erläutern sowie resultierende Speicher- und Laufzeitkosten herleiten und einordnen zu können.

Definition 2.2.1 Eine *Spline-Funktion* $\mathcal{S}_\delta$ vom Grad δ zu einer zeitlichen Zerlegung $t_0 < \ldots < t_n$ erfüllt zwei Bedingungen: Erstens ist die Einschränkung $\mathcal{S}_\delta$ für jedes Intervall $[t_i, t_{i+1}]$, $0 \leq i < n$, ein Polynom vom Grad höchstens δ und zweitens ist $\mathcal{S}_\delta$ mindestens $(\delta - 1)$-mal stetig ableitbar in $[t_0, t_n]$.

Definition 2.2.2 *B-Splines* (oder auch *Basis-Splines*) $\mathcal{B}_{i,\delta+1}$ vom Grad δ zu einer Zerlegung $t_0 < \ldots < t_n$ sind Spline-Funktionen vom Grade δ, die sich durch den nachfolgend beschriebenen rekursiven Prozess (der Tiefe δ) aus charakteristischen Funktionen $(\mathcal{B}_{i,1})_{i=1,\ldots,n+\delta}$ ergeben, die auf den Intervallen $[t_i, t_{i+1}[$ definiert sind. Um Informationen über Ableitungswerte zu den Zeitpunkten t_0 und t_n darzustellen, werden δ Zeitintervalle der Länge 0 an beiden Endpunkten der Zerlegung ergänzt; hierfür sei $t_{-\delta} := \ldots := t_{-1} := t_0$ und $t_{n+\delta} := \ldots := t_{n+1} := t_n$ definiert.

$$
\begin{aligned}
\mathcal{B}_{i,1}(t) &:= \begin{cases} 1, & t \in [t_i, t_{i+1}[\\ 0, & \text{sonst} \end{cases} \\
\mathcal{B}_{i,j}(t) &:= \omega_{i,j} \cdot \mathcal{B}_{i,j-1}(t) \; + \; (1 - \omega_{i+1,j}) \cdot \mathcal{B}_{i+1,j-1}(t), \quad \text{mit} \\
\omega_{i,j}(t) &:= \begin{cases} \frac{t - t_i}{t_{i+j-1} - t_i}, & t_i \neq t_{i+j-1} \\ 0, & \text{sonst} \end{cases}
\end{aligned}
$$

Jede Spline-Funktion $\mathcal{S}_\delta$ vom Grad δ (und zu einer Zerlegung $t_0 < \ldots < t_n$) kann durch eine eindeutige Linearkombination von B-Splines $\mathcal{B}_{i,\delta+1}$ (zu eben jener Zerlegung) dargestellt werden [dB78].

Spline-Interpolation Ein auf der Spline-Interpolation beruhendes Repräsentationskonzept konstruiert aus einer Menge von definierenden Fakten $f(t_0), \ldots, f(t_n)$ eine Spline-Interpolationsfunktion $\mathcal{SI}_\delta(f)(t) = \sum_{i=-\delta}^{n-1} a_i \cdot \mathcal{B}_{i,\delta+1}(t)$. Die Koeffizienten a_i berechnen sich durch Lösen eines linearen Gleichungssystems, das sich aus den folgenden Interpolationsbedingungen ergibt:

$$
\sum_{i=-\delta}^{n-1} a_i \cdot \mathcal{B}_{i,\delta+1}(t_j) \;=\; f(t_j), \quad j = 0, \ldots, n \tag{2.2}
$$

Um eine eindeutige Lösung des obigen Systems zu erhalten, sind $\delta - 1$ zusätzliche Interpolationsbedingungen vorzugeben. Diese *Randbedingungen* werden üblicherweise verwendet, um das Verhalten der Splinefunktion an den Randpunkten t_0 und t_n festzulegen.[10] Somit ergibt sich ein Gleichungssystem mit $n + \delta$ linearen Gleichungen in $n + \delta$ Unbekannten. Da die B-Splines $\mathcal{B}_{i,\delta+1}(t)$ außerhalb des jeweiligen Intervalls $[t_i, t_{i+\delta+1}[$ verschwinden, ist die zu einem solchen Gleichungssystem korrespondierende Matrix eine *Bandmatrix* [BBB87] der Breite δ, d. h. die nicht verschwindenden

[10] Die oben erwähnte Eigenschaft kubischer Splines, die durchschnittliche quadratische Krümmung annähernd zu minimieren, bezieht sich auf kubische Spline-Interpolation mit *natürlichen Randbedingungen*. Letztere geben eine verschwindende erste Ableitung in den Randpunkten vor [Wer92].

Matrixelemente befinden sich auf den δ Hautpdiagonalen, und das Gleichungssystem ist somit in asymptotischer Laufzeitkomplexität von $\mathcal{O}(\delta \cdot n)$ lösbar.[11] Dies entspricht einer Komplexität von $\mathcal{O}(n)$, sofern δ als konstant (oder nur als beschränkt) interpretiert wird.[12] Somit können die Koeffizienten a_i, $i = -\delta, \ldots, n-1$ sämtlich in $\mathcal{O}(n)$ Zeit berechnet werden. Für den Spezialfall $\delta = 1$ konstruiert eine Spline-Interpolation stückweise lineare Funktionen, für den Fall $\delta = 3$ die in der Praxis häufig verwendeten kubischen Splines.

Bemerkung 2.2.3 Die vorgestellte Form der Spline-Interpolation wird auch als nicht-parametrische Spline-Interpolation bezeichnet. Durch *parametrische Spline-Interpolation* [BBB87] hingegen ließe sich eine d-dimensionale Bewegung als Bild eines Splines im $(d+1)$-dimensionalen Raum darstellen. Bei dieser Interpolationsform besitzt allerdings die Zerlegung, die wie in Abschnitt 2.2.2 beschrieben die Interpolation gewissermaßen unterteilt, keine Semantik. Diese Form der Spline-Interpolation hat den Vorteil, dass diese Zerlegung frei gewählt werden kann. Dies wird in der geometrischen Modellierung und in der Computergrafik ausgenutzt, indem Zerlegungselemente mit Vielfachheiten verwendet werden. Der Grad des Splines zwischen zwei Zerlegungselementen sinkt mit deren Vielfachheiten; somit lassen sich zum Beispiel in einer kubischen Spline-Interpolation auch lineare Segmente darstellen.[13]
Trotz dieser Vorteile scheint folgender Nachteil der parametrisierten Interpolation im Kontext der Repräsentation und Verarbeitung von Bewegungen schwerer zu wiegen: Die Auswertung einer Trajektorie zu einem Zeitpunk t' wäre nicht mehr durch Auswerten einer Funktion für den Wert t', sondern nur durch die aufwändige Bestimmung des Schnittes des Bildes einer Spline-Funktion mit der Hyperebene $t = t'$ (im $(d+1)$-dimensionalen Raum) zu realisieren.

Interne Darstellung der Repräsentation

Für die interne Darstellung einer Spline-Trajektorie existieren mehrere sinnvolle Optionen. Eine implizite Darstellung erfordert nur die Speicherung der definierenden Fakten (sowie optional $\mathcal{O}(\delta)$ vieler wählbarer Werte für die Randbedingungen). In diesem Fall muss allerdings, um eine Auswertung der Funktion $\mathcal{SI}_\delta(f)$ vorzunehmen, zunächst die explizite Darstellung, d. h. die Koeffizienten a_i, durch Lösen des beschriebenen Gleichungssystems berechnet werden.
Für eine geeignete explizite Darstellung wären die Zerlegung durch die Zeitpunkte t_i der definierenden Fakten sowie die Koeffizienten a_i (für jede Dimension der Bewegung) vorzuhalten. Als effizienteste Methode zur Auswertung einer solcherart dargestellten Spline-Funktion $\mathcal{SI}_\delta(f)$ gilt der *de Boor-Algorithmus* [dB78]. Dieser

[11] Eine Herleitung obiger Laufzeit ergibt sich mit Hilfe der linearen Algebra, siehe beispielsweise Fischer [Fis05], und wird bei Bartels [BBB87] sowie detaillierter bei Blunck [Blu02] ausgeführt. Eine Erläuterung des Begriffes der asymptotischen Laufzeitkomplexität findet sich in Kapitel 4.

[12] Im Datenmodell unseres Rahmenwerks wird eine Obergrenze für den maximalen Splinegrad δ für die Repräsentation von Bewegung vorgegeben, um die Forderung nach Größenbeschränktheit aus Abschnitt 2.1.3 zu erfüllen. Die nähere Begründung ergibt sich aus den Ausführungen in Kapitel 2.3.

[13] Die Vorteile der Bewegungsrepräsentation durch verschieden charakterisierte Typen von Funktionen sowie alternative Realisierungen solcher Mischform-Repräsentationskonzepte werden in Abschnitt 2.2.3 skizziert.

nutzt die rekursive Definition der B-Splines, um $\mathcal{SI}_\delta(f)$ als Linearkombination von B-Splines auszuwerten, ohne allerdings die B-Splines explizit berechnen oder auswerten zu müssen. Die Zeitkomplexität einer einzelnen Auswertung an einem Zeitpunkt t ergibt sich zu $\mathcal{O}(\log_2 n + \delta^2)$, wobei der logarithmische Faktor aus einer binären Suche nach dem Zeitpunkt t in der Zerlegung durch die definierenden Fakten resultiert.
Die die Interpolationsfunktion $\mathcal{SI}_\delta(f)$ definierenden Fakten können aus der expliziten Darstellung von $\mathcal{SI}_\delta(f)$ rekonstruiert werden. Da jedoch numerische Ungenauigkeiten bei der Konstruktion und Auswertung einer Spline-Interpolationsfunktion auftreten können, sind diese Fakten eventuell nur mit Informationsverlust zurückzugewinnen. Daher werden im hier vorgestellten Rahmenwerk für eine durch Splines repräsentierte Bewegung deren explizite Repräsentation vorgehalten sowie — defaultmäßig für alle Repräsentationskonzepte — die exakten Daten der definierenden Fakten. Dadurch ist die Forderung aus Abschnitt 2.1.3 nach der Abrufbarkeit der (exakten) definierenden Fakten gewährleistet — ohne dass eine explizite Rekonstruktion dieser Fakten durch Auswertungen der Interpolationsfunktion nötig würde. Durch die explizite Darstellung der Funktion ist zudem eine schnellere Auswertung einer Trajektorie gewährleistet.

Aktualisieren der Repräsentation

Spline-Interpolationsfunktionen bieten auf Grund ihrer Balance-Eigenschaft den Vorteil, eine „sanfte“ Verbindung zwischen definierenden Fakten herzustellen, d. h. keine unnatürlich abrupten oder willkürlichen Geschwindigkeits- oder Richtungsänderungen darzustellen. Ein hiermit korrespondierender Nachteil ist ihre suboptimale Lokalität: Eine Modifikation (d. h. ein Einfügen, Löschen oder Verändern) eines einzelnen definierenden Faktums wirkt sich auf die Spline-Funktion in ihrem gesamten zeitlichen Definitionsbereich aus; allerdings nimmt diese Auswirkung auf das Verhalten der Funktion mit der (zeitlichen) Entfernung von der Stelle der Modifikation stark ab.[14] Dennoch müsste eine Spline-Interpolationsfunktion, die die Modifikation exakt reflektiert, komplett neu berechnet werden.
Für Szenarien, in denen die Neuberechnung im Falle einer Modifikation inakzeptabel, aber die Balance-Eigenschaft der Interpolation in einer eingeschränkten Form ausreichend ist, kann folgende Unterteilungsstrategie angewendet werden, die sicherstellt, dass eine Modifikation an der Stelle t sich nur auf einen beschränkten zeitlichen Bereich der Interpolationsfunktion auswirkt: Es ist zunächst eine Teilzerlegung $\{t_{i_1}, \ldots, t_{i_k}\}$ der Zerlegung durch die definierenden Fakten zu bestimmen, um den zeitlichen Definitionsbereich der Interpolationsfunktion in Intervalle $I_j = [t_{i_j}, t_{i_{j+1}}]$ zu unterteilen; für jedes dieser Zeitintervalle ist eine im Folgenden als *Sub-Spline-Interpolation* $\mathcal{SI}_{\delta,j}$ bezeichnete Interpolationsfunktion zu berechnen. Die jeweiligen Randbedingungen dieser Teil-Interpolationen können so gewählt werden, dass stetige Ableitungswerte in allen Zeitpunkten t_{i_j} gesichert sind (o.B.d.A. dargestellt für ungerades δ):

[14] Bei Fourier- oder polynomialer Interpolation hingegen kann sich eine Änderung eines definierenden Faktums im gesamten Definitionsbereich der Interpolation gleich stark auswirken.

$$(\mathcal{SI}_{\delta,j}(f))^{(i)}\left(t_{i_j}\right) = (\mathcal{SI}_{\delta,j-1}(f))^{(i)}\left(t_{i_j}\right) \quad \text{und}$$

$$(\mathcal{SI}_{\delta,j}(f))^{(i)}\left(t_{i_{j+1}}\right) = (\mathcal{SI}_{\delta,j+1}(f))^{(i)}\left(t_{i_{j+1}}\right) \quad \text{für } i = 1, \ldots, \frac{\delta-1}{2}$$

Mit diesem Verfahren kann allerdings die Stetigkeit der $((\delta+1)/2)$-ten bis $(\delta-1)$-ten Ableitung in Punkten t_{i_j} der Teilzerlegung nicht mehr (durch Angabe von Interpolationsbedingungen bei der Konstruktion der Sub-Spline-Interpolationen) zugesichert werden; im Falle kubischer Splines geht somit die Stetigkeit der zweiten Ableitung in den Punkten t_{i_j} im Allgemeinen verloren. Um diese Stetigkeit dennoch zu sichern, wäre eine Erhöhung des Grades der verwendeten Sub-Spline-Interpolation von δ auf $2 \cdot \delta - 1$ erforderlich, so dass nun $\delta - 1$ stetige Ableitungen auch in den Intervallendpunkten der Teilzerlegung zugesichert werden können. Die Erhöhung des Grades impliziert für die Laufzeitkosten für Konstruktion und Auswertung, entsprechend den Ausführungen zu den mathematischen Grundlagen von Splines, eine Erhöhung um in etwa einen Faktor 4. Sind die zeitlichen Teilintervalle I_j fixiert, so bleiben die einzelnen Sub-Spline-Interpolationen $\mathcal{SI}_{\delta,j}$ insofern unabhängig voneinander, als dass sich eine Aktualisierung der Interpolationsfunktion durch Modifikation eines definierenden Faktums (t_i, f_i) nur auf die (maximal zwei) Sub-Spline-Interpolationen $\mathcal{SI}_{\delta,j}$ auswirkt, zu deren zeitlichem Definitionsbereich t_i gehört. Entsprechend sind auch nur diese maximal zwei Sub-Spline-Interpolationen neu zu konstruieren, um die Modifikation exakt zu reflektieren. Ein Fortführen einer Trajektorie (über den zeitlichen Definitionsbereich $[t_0, t_n]$ hinaus) wirkt sich sogar nur auf die Sub-Spline-Interpolation mit dem spätesten zeitlichen Definitionsbereich aus.
Die Festlegung (der Granularität) der Teilzerlegung für eine Bewegungsrepräsentation durch Sub-Splines wirkt sich sowohl auf den Speicherbedarf als auch auf die Lokalität der Bewegegungsrepräsentation aus. Bei der Auswahl der Intervallgrenzen t_{i_j} können diese beiden Kriterien anwendungsspezifisch gewichtet werden — etwa durch Schwellwerte μ_{min} und μ_{max} für die minimal bzw. maximal zulässige Anzahl an definierenden Fakten in einem Sub-Spline-Intervall. Für die Verwendung von Splines vom Grad δ und eine Aufteilung in Intervalle mit jeweils $\mu_{min} = \mu_{max}$ definierenden Fakten erhöht sich die Repräsentationsgröße der Interpolation um einen Faktor $(1 + (\delta - 1)/\mu_{min})$ gegenüber der Verwendung einer linearen Interpolation.
Eine laufzeiteffizientere Variante des beschriebenen Verfahrens zur Gewährleistung der Lokalität der Spline-Interpolation ergibt sich durch das jeweilige Verbinden zweier Sub-Spline-Interpolationen (definiert auf Intervallen $[\ldots, t_i]$ und $[t_{i+1}, \ldots]$) durch eine geeignete Funktion, etwa durch ein Polynom vom Grade $2 \cdot \delta - 1$. Somit wäre die $(\delta - 1)$-fache Ableitbarkeit der Interpolationsfunktion in den Intervallendpunkten t_i und t_{i+1} gesichert, ohne dass der Grad der Sub-Spline-Interpolationen selbst hierfür erhöht werden müsste.

2.2.3 Weitere Repräsentationskonzepte

Auch wenn Spline-Interpolationen durch ihre Sanftheit und stetige Ableitbarkeit Bewegungsrepräsentationen liefern, die Bewegungen physikalisch realistischer abbilden

als stückweise lineare Interpolation und gleichzeitig weniger willkürlich von einer solchen abweichen als beispielsweise Fourier- oder polynomiale Interpolation, weisen Splines für die realistische Darstellung von Bewegungen auch Einschränkungen auf. So können neben den zu interpolierenden Fakten keine weiteren Informationen über das generell zu erwartende Bewegungsverhalten des betrachteten Objektes berücksichtigt werden, etwa die Manövrierfähigkeit des Objektes. Des Weiteren impliziert die Sanftheit von Spline-Kurven die Gefahr, dass Bewegungen wie abrupte Bremsmanöver verfälscht abgebildet werden: Eine durch Spline-Interpolation resultierende Bewegungsrepräsentation würde das Abbremsen eines Objektes eventuell so darstellen, dass es sich kurzzeitig sogar rückwärts bewegt. Folgend werden daher einige Alternativen zu Spline-Interpolationen und ihre Vorteile gegenüber diesen vorgestellt.

ν-Splines In einigen Anwendungen sind grundsätzliche Bewegungseigenschaften der zu repräsentierenden Objekte bekannt und in solchen Situationen ist es sinnvoll, diese zusätzliche Information in die Konstruktion der Trajektorien eingehen zu lassen. Ein Flugzeug beispielsweise vollführt Richtungs- und Höhenänderungen typischerweise in der näheren Umgebung festgelegter Punkte und bewegt sich ansonsten fast linear. Solch eine Bewegung kann durch ν-Splines repräsentiert werden, bei denen die Änderung der zweiten Ableitung in Bezugspunkten vorgeschrieben werden kann. Sind für verschiedene Flugzeugtypen jeweils ihre Wendekreise bekannt, so kann der Wendekreis eines Flugzeugtyps als Parameter für die Konstruktion von ν-Spline-Trajektorien genutzt werden. Diese möglichst genaue Repräsentation von Bewegungseigenschaften ist in Flugplanung und –überwachung wesentlich; auch hier sei wiederum die BADA-Datenbank erwähnt, die zur Vorhaltung solcher Bewegungscharakteristika für verschiedene Flugzeugtypen von EUROCONTROL gepflegt wird [Eur99, NPI+05]. Die Speicherung und mathematische Handhabung von interpolierenden ν-Splines entspricht in weiten Teilen den Ausführungen zur Spline-Interpolation aus Abschnitt 2.2.2. Auch ν-Splines können eindeutig als Linearkombinationen von B-Spline-Koeffizienten dargestellt werden. Genannte Parameter wie Wendekreise gehen als Interpolationsbedingungen in ein Gleichungssystem wie das für Spline-Interpolation geschilderte ein und ersetzen dort Bedingungen, die die stetige Ableitbarkeit höherer Ordnung in den Zeitpunkten der Zerlegung zusichern. Um dieselben Stetigkeitsbedingungen wie eine Spline-Interpolationsfunktion zu erfüllen, muss für eine ν-Spline-Funktion entsprechend ein höherer Grad gewählt werden.[15]

Hermite-Interpolationen Sind zu einer darzustellenden Bewegung als diskrete Informationen nicht nur Positionen, sondern auch Richtungen und Geschwindigkeiten bekannt, wie z.B. bei durch GPS erfassten Bewegungen, so können diese In-

[15] Einschränkend ist zur Verwendung von ν-Splines zur Bewegungsrepräsentation zu sagen, dass diese in der geometrischen Modellierung zumeist für parametrische Interpolationen, vergleiche Bemerkung 2.2.3, genutzt werden. Eine naive Verwendung von ν-Splines zur Interpolation zu einer Zerlegung mit zeitlicher Semantik kann zu einer irritierenden Repräsentation der Geschwindigkeit führen: Es muss besondere Aufmerksamkeit darauf gelegt werden, dass für gleichförmige Bewegungen keine Geschwindigkeitsschwankungen in der zeitlichen Nähe der Zerlegungselemente resultieren.

formationen in die Bewegungsrepräsentation eingehen, d. h. interpoliert werden.[16] Techniken zur Interpolation nicht nur von Funktionswerten, sondern auch von Ableitungswerten sind unter dem Begriff *Hermite-Interpolation* zusammengefasst, vergleiche Nürnberger [Nür78] für eine Übersicht. Die stückweise lineare Interpolation ist hingegen nicht in der Lage, mehr als ein Datum zu je einem Zerlegungselement zu interpolieren. Bei Hermite-Interpolationen ist wie für ν-Spline-Interpolationen ein gegenüber Abschnitt 2.2.2 modifiziertes Gleichungssystem aufzustellen und zu lösen. Auch hier lassen sich Bedingungen, die bei der Spline-Interpolation stetige Ableitungen höherer Ordnung in den Zeitpunkten der Zerlegung garantieren, ersetzen — in diesem Fall durch Bedingungen, die die Ableitungswerte in diesen Zeitpunkten fest vorgeben. Dementsprechend gilt wie für ν-Splines, dass die stetige Ableitbarkeit eines nur Positionsangaben interpolierenden Splines für einen Hermite-interpolierenden Spline nur zu erreichen ist, wenn der Spline-Grad entsprechend vergrößert gewählt wird.

Mischform-Repräsentationskonzepte Ein *Mischform*-Repräsentationskonzept, d. h. ein Konzept, das Trajektorien segmentweise durch verschiedene Interpolationsformen, beispielsweise Liniensegmente und eine weitere Trajektorienart wie Kreisbögen oder kubische Splines, darstellt, scheint in vielen Anwendungen vorteilhaft. Dies ist insbesondere dann der Fall, wenn die Objekte, deren Bewegung zu repräsentieren ist, sich je nach Kontext verschieden bewegen, wie beispielsweise im Flugverkehrsmanagement, vergleiche [GM97]. Von Meratnia [Mer05] und Meratnia und de By [Md03] wurden Mischform-Repräsentationskonzepte, die hauptsächlich für sich in geometrischen Netzwerken bewegende Objekte konzipiert wurden, vorgestellt und motiviert. Die Argumentation der Autoren basiert auf der Beobachtung, dass für viele Typen sich bewegender Objekte ihr Verhalten in verschiedenen Phasen ihrer Bewegung unterschiedlich ist; so mögen solche Objekte wie Fahrzeuge in der Lage sein, sehr viel abrupter abzubremsen (also negativ zu beschleunigen) als (positiv) zu beschleunigen, d. h. beim Vorgang des Abbremsens kann der Absolutbetrag der Beschleunigung sehr viel größer sein als beim positiven Beschleunigen. Ein einzelnes Repräsentationskonzept, d. h. eine einzelne Form der Interpolation, unterscheidet im Allgemeinen nicht zwischen diesen Arten der Beschleunigung (bzw. allgemeiner: zwischen verschiedenen charakterisierten Phasen der Bewegung eines mobilen Objektes). Meratnia und de By regen daher an, für Repräsentation einer Bewegung ihre einzelnen Phasen — wie Brems- und Beschleunigungsphasen — zu identifizieren bzw. zu kategorisieren, diese durch jeweils adäquate Interpolationen darzustellen und die Interpolationen dann geeignet und insbesondere stetig zu verbinden.
Die Modellierung des hier vorgestellten Rahmenwerks lässt Mischform-Repräsentationskonzepte zu. Für die interne Darstellung einer entsprechenden Repräsentation ist zu beachten, dass für jedes Segment die Information, wie es zu interpretieren ist, d. h. welches (Teil-)Repräsentationskonzept ihm assoziiert ist, vorgehalten werden muss. Mit dieser Information ist die Auswertbarkeit der Trajektorie gesichert; ein dem Mischform-Repräsentationskonzept zugehöriger Evaluationsalgorithmus hat

[16] Die Erfassung von Geschwindigkeit und Orientierung einer Bewegung durch GPS-Geräte wird bei El-Rabbany [El-06] beschrieben und als eine inzwischen dem GPS-Standard zugehörige Funktionalität dargestellt.

eine Fallunterscheidung hinsichtlich dieser Information zu beinhalten. Von Forlizzi *et al.* [FGNS00] wurde eine Mischform und ihre Handhabung vorgestellt, die aus Liniensegmenten und (radizierten und nicht radizierten) polynomialen Segmenten zusammengesetzte Trajektorien bereit stellt, um stückweise lineare Trajektorien sowie die Distanzen zweier solcher durch einen einzigen Datentyp `mreal` innerhalb des in dieser Arbeit adaptierten ADT-Modells von Güting *et al.* [GBE+00] darzustellen.

Exponentialsplines Exponentialsplines sind im Hinblick auf die Repräsentation sehr stark gekrümmter Kurven entwickelt worden, für die die in Kapitel 2.2.2 beschriebenen polynomialen B-Spline-Linearkombinationen nur bedingt geeignet sind. Im Gegensatz zu letzteren sind die Basis-Funktionen der Exponentialsplines nicht polynomial, sondern eine Kombination von Sinus- und Kosinusfunktionen. Dennoch werden auch für diesen Anwendungsfall häufig speziell adaptierte polynomiale Splinefunktionen statt der Exponentialsplines verwendet, da die Verwendung letzterer einen hohen Rechenaufwand impliziert, vergleiche Hoschek und Lasser [HL89].

Polynominiale und Fourier-Interpolationen Durch klassische polynomiale Interpolation werden n Funktionswerte durch ein eindeutiges Polynom n-ten Grades verbunden. Problematisch an dieser Art der Interpolation ist die in Abhängigkeit der Eingabegröße n unterschiedliche Komplexität und damit auch Charakteristik der resultierenden Interpolationsfunktionen. Für große Eingabegrößen neigen diese zudem zum Oszillieren, vergleiche Hoschek und Lasser [HL89]; des Weiteren erhöht sich der Aufwand für ihre Auswertung linear mit der Anzahl der interpolierten Werte.[17]
Ähnliche Nachteile wie die polynomiale Interpolation besitzt auch die Fourier-Interpolation, die eine Menge von n definierenden Fakten eindeutig durch eine Linearkombination von n periodischen Schwingungskurven mit unbeschränktem Träger interpoliert, vergleiche Abschnitt 2.1.3. Bei wachsender Eingabegröße gehen immer hochfrequentere Schwingungen in die Interpolation mit ein, so dass auch diese für große Eingaben stark oszilliert. Allerdings eignet sich die Fourier-Interpolation durchaus, um periodische Bewegungen, wie etwa Schwingungen, darzustellen. Für die Repräsentation nicht zyklischer und nicht schwingender Bewegungen scheinen andere Interpolationstechniken geeigneter.
Zu polynomialer und zur Fourier-Interpolation sind Varianten konstruierbar, in denen die Basisfunktionen, durch deren Linearkombination sich die Interpolationsfunktionen bestimmen, nicht als die n niedrigsten Monome (bzw. nicht als die n Schwingungsfunktionen mit der niedrigsten Frequenz) gewählt werden. Stattdessen können aus den jeweiligen Basisfunktionen n beliebige zur Linearkombination ausgewählt werden. Es ist für die interne Darstellung einer so zusammengesetzten Interpolationsfunktion zu beachten, dass nun nicht nur die Gewichte, sondern auch Identifikatoren, d. h. die Indizes, der Basisfunktionen, auf die die Gewichte sich beziehen, zu speichern sind. Eine derartige Fourier-Interpolation wird häufig

[17] Die polynomiale Interpolation spielt in Bereichen der Algorithmik zur Analyse sich bewegender Objekte insofern eine wesentliche Rolle, als dass viele Algorithmen in der Notation der kinetischen Datenstrukturen vorgestellt werden, welche in Kapitel 3.2 genauer erläutert wird. Bei der theoretischen Evaluation dieser Algorithmen werden als Bewegungsrepräsentationen zumeist Polynome eines wählbaren, aber festen Grades angenommen.

für kompressionsfähige Datenformate, etwa für das JPEG-Format für Bilddaten, eingesetzt: Die Interpolationen lassen sich einfach und gleichzeitig mit geringem Informationsverlust komprimieren, indem man die Elemente der Linearkombination mit den betragsmäßig kleinsten Gewichten aus der internen Darstellung der Interpolationsfunktion entfernt, vergleiche Meyer [Mey93, Kapitel 3.3]. Vorteile von Bewegungsrepräsentationen, die eine einfache Komprimierbarkeit mit geringem Informationensverlust erlauben, werden in Kapitel 2.3 genauer besprochen und auch unter anderem von Cao *et al.* [CWT03] herausgestellt.

Wavelets Wie bei der Fourier- werden auch bei Wavelet-Interpolationen die Interpolationsfunktionen als Linearkombination von Wellenfunktionen dargestellt — allerdings besitzen diese *Wavelets* im Gegensatz zu den modifizierten Sinus- und Kosinusfunktionen der Fourier-Interpolation jeweils einen beschränkten Träger.
Die resultierende Basis von Schwingungsfunktionen kann von vielerlei Gestalt sein und folgerichtig sind verschiedenste Techniken der Wavelet-Interpolation vorgestellt worden.[18] Ein Wavelet-basiertes Repräsentationskonzept sind beispielsweise Spline-Wavelets (des Grades δ), die als Interpolationsfunktionen Splines (vom Grade δ) generieren. Die resultierenden Interpolationsfunktionen werden allerdings in einem anderen Format vorgehalten als den in Abschnitt 2.2.2 dargestellten. Vielmehr bilden die Basisfunktionen (verschiedenster Wavelet-Techniken) eine Hierarchie von Wellenfunktionen, deren obere Stufen aus jeweils wenigen Funktionen mit einer niederfrequenten Schwingung, aber einem großen Träger bestehen, während die unteren Stufen von vielen hochfrequenten Schwingungsfunktionen mit stark eingeschränktem Träger gebildet werden; klassischerweise verdoppelt sich in jeder Hierarchiestufe die Anzahl der Wellenfunktionen und gleichzeitig halbiert sich die Größe ihrer Träger. Im Kontext der Bewegungsrepräsentation bilden die Gewichte, die zu Funktionen der oberen Hierarchieebenen korrespondieren, die globalen, aber nicht zu heftigen Schwankungen einer Bewegung ab. Durch die Gewichte der Funktionen der unteren Hierarchieebenen werden lokale, aber starke Schwankungen abgebildet.
Vorteil einer solchen Hierarchie von Basisfunktionen sind die kleinen (bzw. die wenigen nicht verschwindenden) Gewichte, die sich bei der Darstellung größtenteils gleichmäßiger Bewegung ergeben. Insbesondere lässt sich deswegen eine Wavelet-Interpolation sehr effizient (durch Entfernen der betragsmäßig geringsten Gewichte) komprimieren, was inzwischen zu ihrem Einsatz für neuere JPEG-Formate geführt hat, vergleiche wiederum Meyer [Mey93]. Für die Bewegungsrepräsentation ist diese Eigenschaft insbesondere nützlich, da für die geeignete Repräsentation einer Bewegung bestimmte zeitliche Bereiche von besonderem Interesse sind — zumeist diejenigen, an denen sich das Bewegungsverhalten ändert. Eine komprimierte Wavelet-Darstellung würde genau diese Bereiche detaillierter repräsentieren.
Es ist allerdings zu beachten, dass bei Kompression einer Wavelet-Darstellung im Allgemeinen die bedingenden Fakten nicht mehr exakt interpoliert werden.[19] Für

[18] Übersichten über Mathematik und Anwendungen von Wavelet-Interpolationen und -Approximationen finden sich beispielsweise bei Chui [Chu92], Louis *et al.* [LMR94], Meyer [Mey93] sowie bei Sweldens und Schröder [SS96].

[19] Präziser gesprochen ist die Interpolation all jener Fakten nicht gesichert, deren Zeitpunkt in einem Träger einer Basisfunktion, deren Gewicht bei der Kompression entfernt wurde, enthalten

das Ziel verlustfreier komprimierter Wavelet-Darstellungen wurde die Verwendung größenvariabler Datentypen vorgeschlagen [Mey93], so dass kleinere Gewichte eine hinsichtlich des Speicherbedarfs billigere Darstellung implizieren, ohne hierfür explizit entfernt werden zu müssen. Da im Kontext der Bewegungsrepräsentation zumeist die exakte Interpolation von Fakten relevant ist, erscheint dieser Ansatz vielversprechend. Generell nachteilig bleibt, dass Wavelet-Repräsentationen wenig Lokalität garantieren können und diese auch durch Modifikationen — wie in Abschnitt 2.2.2 für Splines beschrieben — nicht erreichbar scheint. Problematisch für die Bewegungs-Repräsentation ist der Einsatz klassischer Wavelet-Interpolationen zudem, da diese von einer äquidistanten zeitlichen Zerlegung (und einer Zweierpotenz als Eingabegröße) ausgehen. Eine Verallgemeinerung zu *second-generation-wavelets*, die für beliebige Zerlegungen zu verwenden sind, ist jedoch inzwischen für viele Wavelet-Arten vorgestellt, vergleiche Sweldens [Swe94] sowie Sweldens und Schröder [SS96].

Gestalt erhaltende Interpolationskonzepte In der Computergrafik und insbesondere im *computer aided design* (CAD) finden Gestalt erhaltende Interpolationstechniken Anwendung, vergleiche Peña [Peñ99]. Diese Techniken sind auch für die Repräsentation von Bewegung interessant, da die Kritierien der Gestalt-Erhaltung, die Interpolationen vorgegeben werden können, auch für die Repräsentation von Bewegung nützlich sind. Insbesondere können solche Interpolationstechniken mehrfach stetig ableitbare Kurven generieren, die dennoch nur lokal von den definierenden Fakten abhängen und zudem nur begrenzt von der linearen Interpolation dieser Fakten hinsichtlich Richtung und Entfernung abweichen. Weitere Eigenschaften (einer Faktenmenge zu einer Bewegung), die durch Gestalt erhaltende Interpolation korrekt wiedergegeben bzw. bewahrt werden können, sind Konvexität, Monotonie und Positivität. Zahlreiche Literatur ist insbesondere zu Spline-basierten, Gestalt erhaltenden Interpolationen erschienen, für eine Übersicht vergleiche Goodmann [Goo02]. Ein Ansatz, der dem für Spline-Wavelets geschilderten Ansatz der Darstellung durch ein Hierarchie beschränkter Wellenfunktionen folgt, wurde von Goodman und Ong [GO05] vorgestellt. Generell basieren die Gestalt erhaltenden Interpolationen auf bereits vorgestellten, parametrisierten Interpolationsformen wie ν-Splines oder Exponentialsplines. Die Erstellung einer Gestalt erhaltenden Interpolation erfolgt häufig durch ein iteratives Redefinieren der Parameter einer solchen Interpolation, bis eine Gestalt erhaltende Interpolationsfunktion bestimmt ist. Somit sind Gestalt erhaltende Interpolationen mit einem erhöhten Konstruktionsaufwand verbunden. Es ist anwendungsabhängig zu entscheiden, ob die Zusicherung von Gestalt erhaltenden Eigenschaften der Bewegungsrepräsentationen diesen erhöhten Berechnungsaufwand rechtfertigt.

ist. Fehlerhafte Gewichte in einer Wavelet-Darstellung wirken sich zudem - entsprechend ihrer Hierarchiestufe - globaler aus als fehlerhafte Gewichte beispielsweise in der Basisdarstellung durch B-Splines.

2.3 Repräsentation zeitvarianter Operationsergebnisse

In diesem Kapitel wird die Abgeschlossenheit des in Abschnitt 2.1.2 vorgestellten Datenmodells gegenüber (beliebig konkatenierten) gelifteten Operationen erörtert. Eine solche vollständige Abgeschlossenheit gilt als offenes, aber viel beachtetes Problem in der einschlägigen Literatur, wie bereits in Abschnitt 2.1.1 belegt wurde. In Abschnitt 2.3.1 wird die Zielsetzung der Abgeschlossenheit an Hand von Anwendungsbeispielen eingehender motiviert. Zudem wird ausgeführt, warum die gewünschte Abgeschlossenheit – ohne dafür die Forderung nach Größenbeschränktheit aus Abschnitt 2.1.3 zu verletzen — nur erreicht werden kann, wenn eine nicht-exakte Darstellung von Operationsergebnissen akzeptiert wird. In Abschnitt 2.3.2 wird eine entsprechende Lösung dieses Problems vorgestellt, die die Abgeschlossenheit der Modellierung des in dieser Arbeit vorgestellten Rahmenwerks sichert und dabei die Forderung nach Größenbeschränktheit von Operationsergebnissen mit Hilfe einer geeigneten Approximation erfüllt.

2.3.1 Problemstellung und Motivation

In der Notation des in Abschnitt 2.1.2 beschriebenen Modells verlangt die oben beschriebene Zielsetzung die Abgeschlossenheit der zeitvarianten Datentypen `mpoint`$\langle d \rangle$ und `mreal`$\langle d \rangle$ gegenüber gelifteten Operationen. Die folgenden Beispiele illustrieren die bei der Anwendung solcher Operationen auftretenden Probleme:

Beispiel 1: Berechne die Distanz `mdistance`(mp_1, mp_2) von zwei Instanzen mp_1 und mp_2 des Typs `mpoint`$\langle d \rangle$.

Sind mp_1 und mp_2 durch lineare Trajektorien repräsentiert, so ist das Resultat in obigem Beispiel eine Wurzel eines Polynoms zweiten Grades. Erwig *et al.* [Er99] erkannten dieses Problem und Forlizzi *et al.* [Fo00] erweiterten daraufhin ihr lineares diskretes Modell, genauer den Datentyp `mreal`, so dass für dessen Instanzen nicht nur stückweise lineare Funktionen, sondern zusätzlich stückweise polynomiale Funktionen zweiten Grades (und wahlweise auch deren Wurzeln) zugelassen sind. Diese Erweiterung des Typs `mreal` ist nicht ausreichend, wenn Ergebnisse der Distanzoperation als Operanden in weiteren Berechnungen verwendet werden:

Beispiel 2: Berechne die iterierte (bzw. konkatenierte) Distanz `mdistance`(`mdistance`(mp_1, mp_2), `mdistance`(mp_3, mp_4)) für vier Instanzen mp_1, mp_2, mp_3, mp_4 des Typs `mpoint`$\langle d \rangle$.

Der von Forlizzi *et al.* erweiterte zeitvariante Datentyp ist unter anderem gegen die Bildung von Distanz, Addition oder Durchschnittsbildung nicht abgeschlossen. Während die Autoren derart iterierte Anwendungen von Operationen wie der Distanzbildung ausschließen, wählen Grumbach *et al.* [GRS98] eine noch restriktivere Anwendbarkeit der Distanzoperation für eine Modellierung, die nicht auf abstrakten Datentypen, sondern auf linearen Constraints basiert: Die in dieser Modellierung

einzig zugelassene Art der Distanzabfrage vergleicht (potenziell zeitvariante) Distanzwerte mit einem Schwellwert; die Rückgaben dieser Abfragen beschränken sich somit auf boolesche Werte, so dass insbesondere keine nicht-linearen Constraints als Ergebnisse von Distanzabfragen resultieren können.

Die Weiterverarbeitung von Distanzergebnissen wie in Beispiel 2 ist jedoch für praktische Anwendungen relevant: Stellt das Ergebnis aus Beispiel 1 z.B. die Distanz zweier Flugzeuge im Formationsflug dar, so ist die iterierte Anwendung der Distanz — wie in Beispiel 2 dargestellt — für den Vergleich zweier Formationsflüge (je zweier Flugzeuge) dienlich. Analog ließen sich durch iterierte Anwendung der Distanz in einer physikalischen Simulation zum Beispiel die Entfernungsentwicklungen zweier Partikel von einem (stationären oder beweglichen) Gravitationskern vergleichen.

Offensichtlich lassen sich die Ergebnisse aus Beispiel 2 im Allgemeinen nicht erneut als Wurzel eines Polynoms zweiten Grades darstellen. Selbst wenn man die Datentypen erneut adäquat erweitern würde, wäre bei iterierter Anwendung der Distanzoperation deren Abgeschlossenheit erneut verletzt. Des Weiteren verlangt der Ausdruck in Beispiel 2 eine Variante der Distanzoperation, die als Eingabe Instanzen des Typs `mreal` akzeptiert, oder eine vorherige Anwendung des Casting-Konstruktors, der aus diesen Instanzen Objekte des Typs $\texttt{mpoint}\langle 1\rangle$ generiert, vergleiche Kapitel 2.1.2. Das Zulassen eines solchen Konstruktors impliziert jedoch, dass nicht nur der Typ `mreal`, sondern auch der Typ $\texttt{mpoint}\langle 1\rangle$ zu erweitern wäre.

Trotz des Aufwandes scheint die letztgenannte Alternative die sinnvollere: Auch die Konstruktion neuer Instanzen vom Typ $\texttt{mpoint}\langle 1\rangle$ aus zeitvarianten Distanzen (wie in Bespiel 1) ist in vielen Bereichen, in denen die zu repräsentierenden Bewegungen diskret vermessen werden, praxisrelevant: Distanzbasierte Systeme zur Bewegungserfassung, z.B. GPS oder Sonartracking, rekonstruieren eine Position aus Distanzmessungen, die an einer festen Anzahl von Messstationen durchgeführt werden. Da diese Distanzen — für sich genommen — relevant sind, sollten sie als zeitvariante Daten interpretiert, d. h. insbesondere interpoliert, und als Instanzen des Typs `mreal` gespeichert werden können. Eine Interpolation dieser Messzeitpunkte ist sogar unerlässlich, wenn im Allgemeinen nicht alle Messstationen synchron, sondern zu unterschiedlichen Zeitpunkten Distanzen erfassen: Die Rekonstruktion einer Position zu einem gegebenen Zeitpunkt ist nur dann möglich, wenn zu diesem Zeitpunkt eine Distanzangabe von allen (bzw. hinreichend vielen) Messstationen vorliegt; im Falle einer asynchronen Messung liegen zu einem Zeitpunkt nicht hinreichend viele Distanzmessungen vor, so dass interpolierte Distanzwerte einzelner Messstationen zur Rekonstruktion verwendet werden müssen.[20] Eine so rekonstruierte Bewegung ist also bereits ein Ergebnis einer Operation, deren Operanden zeitvariante Distanzen (und damit Instanzen des Typs `mreal`) sind.

[20] Vom letztgenannten Fall der *asynchronen* Messung durch verschiedene Messstationen ist bei vielen Szenarien der Datengewinnung auszugehen. In einigen dieser Szenarien ist dies schon durch die fehlende Möglichkeit der Synchronisation der messenden Stationen begründet. Bei insbesondere distanzbasierten Messtechniken wie GPS sind zudem die exakten Zeitpunkte, zu denen ein Datum gemessen wird, von der Laufzeit von Signalen und damit von der Position bzw. der Bewegung des beobachteten Objektes selbst abhängig und können daher nicht synchron gewählt werden — dies macht sich insbesondere bei Messtechniken mit langen Singallaufzeiten, wie etwa dem Sonartracking, bemerkbar.

Ein anderes Anwendungsbeispiel für die iterierte Anwendung der Distanzoperation betrifft die Datenanalyse in Gebieten wie der Verhaltensbiologie. Zeitvariante Distanzen zwischen einem Jungtier und seinen Eltern, deren Bewegungen jeweils durch eine Instanz des Typs $\texttt{mpoint}\langle d\rangle$ darzustellen sind, werden in der Datenanalyse weiterverarbeitet: Unterschiede im Verhalten zweier Jungtiere können etwa anhand der Abweichung der zugehörigen Vater-Mutter-Kind-Distanzen analysiert werden.[21]
Obige Beispiele motivieren eine Modellierung, die die beliebig iterierte und verknüpfte Anwendung von Operationen wie der Distanz zulässt. Zum anderen zeigen sie auf, dass entsprechende Operationsergebnisse nicht allesamt exakt dargestellt werden können — sofern nicht beliebig komplexe und speicheraufwändige Datentypen zur Verfügung stehen. Diese Aussage lässt sich nachweisen, sofern nur etwa die stückweise linearen Funktionen als Teilmenge der für die Bewegungsrepräsentation zur Verfügung stehenden Funktionen vorausgesetzt werden:

Lemma 2.3.1 (Lemma 6.6 in [Blu02]) *Jeder zeitvariante Datentyp, der als Instanzen lineare Funktionen oder polynomiale Funktionen höheren Grades zulässt, ist nicht abgeschlossen unter der iterierten Anwendung der Distanzoperation.*

Eine weiteres und generelles Problem bei der Gewährleistung der Größenbeschränktheit ergibt sich für Ergebnisse nicht-unärer Operationen, sofern die Fakten der Operanden nicht zeitlich synchronisiert vorliegen.

Beispiel 3: Berechne das zeitvariante Mittel der Positionen einer Menge $M = \{mp_1, \ldots, mp_n\}$ von Instanzen des Typs $\texttt{mpoint}\langle d\rangle$.

Das Ergebnis in Beispiel 3 stellt den (zeitvarianten) Schwerpunkt einer Wolke sich bewegender Objekte dar. Alle bedingenden Fakten aller Operanden müssen in der Ergebnisrepräsentation berücksichtigt, d. h. interpoliert werden, damit diese exakt ist. Ist für jeden Operanden mp_i die Anzahl $|F(mp_i)|$ der ihn bedingenden Fakten beschränkt durch s, so gilt für das exakte zeitvariante Ergebnis mr_3 der Berechnung aus Beispiel 3: $s \leq |F(mr_3)| \leq n \cdot s$. Die rechte Ungleichung ist scharf, wenn alle Operanden durch s Fakten definiert sind und alle Fakten aller Operanden jeweils unterschiedliche Zeitpunkte betreffen.[22]

Beispiel 4: Berechne das zeitvariante Mittel der paarweisen Distanzen einer Menge $M = \{mp_1, \ldots, mp_n\}$ von Instanzen des Typs $\texttt{mpoint}\langle d\rangle$.

Das Resultat mr_4 aus Beispiel 4 reflektiert beispielsweise die (zeitvariante) mittlere Anziehung zwischen geladenen Partikeln in einer physikalischen Simulation oder zwischen Tieren einer Gruppe, deren soziales Verhalten zu analysieren ist. Für das Ergebnis mr_4 gilt analog zu Beispiel 3: $s \leq |F(mr_4)| \leq n \cdot s$

[21] Bei Vuilleumier [Vui03] findet sich ein komplexeres und ausführlicher diskutiertes Anwendungsbeispiel aus der Verhaltensbiologie, in dem zeitvariante Distanzen zu Analysezwecken weiterverarbeitet werden.

[22] Bei Erwig *et al.* [EGSV99] wurde diese Problematik bereis ausführlich beschrieben.

Bemerkung 2.3.2 Es ist zu beachten, dass die Verletzung der Größenbeschränktheit aus Beispiel 3 als weniger schwerwiegend als die in den Beispielen 1 und 2 dokumentierte angesehen werden kann, da nur die Repräsentationsgröße des Ergebnisses betroffen ist; das Ergebnis selbst ließe sich zumindest in jedem konkreten Fall mit entsprechendem Speicheraufwand und dem Repräsentationskonzept der Operanden darstellen, sofern dieses für alle Operanden einheitlich ist und die Repräsentationen dieses Konzeptes einen Vektorraum bilden, also abgeschlossen gegen Addition und skalare Multiplikation sind. Dies ist für vorgestellte Konzepte wie der stückweise linearen Interpolation oder der Spline-Interpolation erfüllt.[23]
Die Abgeschlossenheit unter der (iterierten) Distanzoperation hingegen ist insofern schwieriger zu erfüllen, als dass deren Ergebnisse im Allgemeinen von höherer algebraischer Komplexität als die Operanden sind; eine exakte Repräsentation würde daher nicht nur einen (gegenüber den Operanden) erhöhten Speicheraufwand, sondern auch die interne Darstellung algebraisch komplexerer Funktionen erfordern.

Die Erörterung in diesem Kapitel veranschaulicht zum einen, dass kein Repräsentationskonzept die Ergebnisse aller relevanten gelifteten Operationen exakt darstellen kann: Schon allein die wiederholte Anwendung der Distanzoperation führt zu unbeschränktem Wachstum der Komplexität und Repräsentationsgröße der (exakten) Resultate. Zum anderen verletzt die Anwendung jeglicher gelifteter Operationen (insbesondere wenn diese eine beliebige Anzahl an Argumenten akzeptieren) die Forderung nach Größenbeschränktheit, sofern alle Operandenfakten berücksichtigt werden sollen und diese nicht zeitlich synchronisiert vorliegen.
Die Abgeschlossenheit von zeitvarianten oder allgemein funktionalen Datentypen gegenüber der Distanz wird, wie bereits erwähnt, auch in anderen Arbeiten diskutiert, vergleiche [EGSV99, MSI02, KKP00]. Die nachfolgend vorgestellte Lösung ist allerdings die einzige insofern universell anwendbare, als dass sie von Repräsentationstypen, d. h. von der Art der Funktionen, die das Datenmodell (bzw. der zeitvariante Datentyp `mreal`) bereitstellt, abstrahiert.

2.3.2 Approximative Repräsentation von Operationsergebnissen durch Faktenrekrutierung

Die oben beschriebene Verletzung der Größenbeschränktheit kann vermieden werden, wenn bei der Konstruktion von Operationsergebnissen, sofern notwendig, eine Größenreduktion vorgenommen wird. Zielsetzung muss sein, bei dieser Reduzierung die wesentlichen Charakteristika des exakten Operationsergebnisses zu bewahren. Der im Folgenden beschriebene Ansatz gewinnt eine approximative Repräsentation des Ergebnisses durch Interpolation derjenigen Fakten, die als die relevantesten (des exakten Ergebnisses) angesehen werden. Eine entsprechende Vorschrift zur Auswahl dieser Fakten sei fortan als *Rekrutierungsstrategie* bezeichnet. Da eine solcherart gewonnene Ergebnisrepräsentation wieder als Operand in Operationsaufrufen eingehen kann, werden ihre definierenden Fakten entsprechend ihrer Relevanz bzw. Vertrauenswürdigkeit klassifiziert. Die (für die Einfachheit der Darstellung auf zwei

[23] Problematisch ist allerdings, wenn die Operanden von unterschiedlichem Repräsentationstyp sind. Dies wird in den Kapiteln 2.3.2 und 3 genauer ausgeführt.

Relevanzklassen beschränkte) Klassifizierung kennzeichnet die zu einem bedingenden Faktum gehörigen Zeitpunkte entweder als „Punkt Erster Klasse (EP)“ oder „Punkt Zweiter Klasse (ZP)“. Diese Terminologie ermöglicht eine Konkretisierung der Forderung nach der Größenbeschränktheit von zeitvarianten Operationsergebnissen aus Kapitel 2.1.3:

Definition 2.3.3 Bezeichne *op* einen p-ären Operator und $\{mp_1, \ldots, mp_n\}$ die Menge seiner Operanden. Die Forderung nach *Größenbeschränktheit* ist für *op* erfüllt, wenn die Repräsentationsgröße des Resultats $op(mp_1, \ldots, mp_n)$ durch $c \cdot E_{max}$ beschränkt ist, wobei E_{max} die Anzahl der EPs des Operanden mit den meisten EPs bezeichnet.

Obige Definition entspricht der Forderung aus Abschnitt 2.1.3, wenn sich die Größe jeder Repräsentation durch die Anzahl seiner EPs multipliziert mit einer (globalen) Konstante beschränken lässt. Wie dies (und die Forderung nach Größenbeschränktheit) mit Hilfe von Rekrutierungsstrategien erfüllt werden kann, wird im Folgenden dargestellt.[24]

Bei der Rekrutierung von Fakten, die das größenbeschränkte Operationsergebnis definieren, sollten die Relevanz der Fakten der Operanden einbezogen werden. Für die Einordnung der Relevanz der Fakten von Operanden sollte berücksichtigt werden, dass die gemessenen (oder vorgegebenen) Werte die einzigen exakt bekannten Daten einer Bewegung sind und daher als *per se* relevant zu erachten sind. Bei der Konstruktion einer Bewegungsrepräsentation durch eine Menge definierender Fakten werden diese daher als EPs markiert.

Definierende Fakten der Repräsentation eines Operationsergebnisses können hingegen jeweils als EP oder als ZP klassifiziert sein. Die Kennzeichnung eines Ergebnisfaktums als ZP ist beispielsweise dann sinnvoll, wenn für den Zeitpunkt dieses Faktums nur für *einen* Operanden ein definierendes Faktum vorliegt — während für den Zeitpunkt eines anderen Ergebnisfaktums Fakten *aller* Operanden vorliegen. Konsequenterweise sollte das letztere Faktum bzw. sein Zeitpunkt als relevanter als der erstere gekennzeichnet sein, da seine Information verlässlicher ist.

Die Rekrutierung von *Kandidaten* für Fakten, die durch ein (approximiertes) Operationsergebnis zu interpolieren sind, nutzt Relevanzwerte $v(t)$, die jedem Kandidatenzeitpunkt t zugewiesen werden. In der folgenden näheren Beschreibung des Sammelns und Bewertens von Kandidatenzeitpunkten t wird vereinfachend davon ausgegangen, dass alle Operanden auf demselben Zeitintervall $[t_b, t_e]$ definiert sind:

Schritt 1: Für die beiden begrenzenden Zeitpunkte $t = t_b$ and $t = t_e$ setze $v(t) = 5 \cdot p$. Damit werden die Intervallendpunkte in jedem Fall zu EPs des Operationsergebnisses.

Schritt 2: Sammle alle Zeitpunkte t, an denen mindestens ein Operand ein definierendes Faktum besitzt. Weise jedem solchen Zeitpunkt t jeweils den Wert

[24] Die Forderung nach Größenbeschränktheit aus Definition 2.3.3 ist auch für die Operation der Konstruktion von `mpoint`$\langle d \rangle$-Instanzen aus n definierenden Fakten erfüllt, wenn der entsprechende Konstruktor als n-ärer Operator verstanden und jeder Zeitpunkt eines übergebenen Faktums als EP interpretiert wird.

$v(t) = 3 \cdot ek(t) + zk(t)$ zu, wobei *ek(t)* (bzw. *zk(t)*) die Anzahl der Operanden angibt, die einen EP (bzw. ZP) an t besitzen.

Schritt 3: [**Optional**] Sammle Zeitpunkte t entsprechend einer spezifischen strategischen Heuristik. Weise jedem solchen Zeitpunkt t wahlweise einen der folgenden Werte zu

- $v(t) = 4 \cdot p$: Die Heuristik, z.B. äquidistantes Sampling, überlagert die Klassifizierung durch die Operanden aus Schritt 2.
- $v(t) = 2 \cdot p$: Die Heuristik respektiert die Klassifizierung von Zeitpunkten, an denen viele Fakten der Operanden vorliegen.
- $v(t) = 0$: Die Heuristik rekrutiert nur zusätzlich zu den in Schritt 1 und 2 gesammelten Punkten, falls die Interpolation weiterer Punkte die Größenbeschränktheit des Operationsergebnisses nicht verletzt.

Aus den in den Schritten 1-3 gesammelten Kandidatenzeitpunkten t werden die insgesamt E_{max} (dem Wert $v(t)$ nach) wichtigsten Zeitpunkte als die EPs des Ergebnisses rekrutiert und entsprechend gekennzeichnet. Die $(c-1) \cdot E_{max}$ nächstwichtigsten Kandidatenzeitpunkte werden als dessen ZPs rekrutiert. Die Repräsentation des Operationsergebnisses wird durch Interpolation der zu den EPs und ZPs korrespondierenden Werte des Ergebnisses erstellt.[25]

Durch das Verwenden einer jeweiligen Heuristik und Gewichtung in Schritt 3 ergeben sich verschiedene Rekrutierungsstrategien: Der optionale Schritt 3 kann etwa genutzt werden, um Charakteristika des exakten Ergebnisses, z.B. lokale Extremalstellen, für die Repräsentation auf jeden Fall zu berücksichtigen. Verzichtet man auf diesen dritten Schritt, so wird das Operationsergebnis allein auf Grundlage der vertrauenswürdigsten Werte des exakten Ergebnisses berechnet.

Beispiel: Zur näheren Veranschaulichung von Rekrutierungsstrategien und insbesondere des Einsatzes von Heuristiken in Schritt 3 sei das in Abbildung 2.1 skizzierte Beispiel zur Berechnung eines Operationsergebnisses erläutert. Zu berechnen sei die zeitvariante Distanz zweier sich bewegender Objekte, deren (eindimensionale) Bewegungen durch f_{mp_1} und f_{mp_2} repräsentiert sind. Für das Beispiel sind folgende Parameter festgehalten:

- Die globale Konstante c aus der Forderung nach Größenbeschränktheit ist als $c := 4/3$ gewählt.
- Die Heuristik in Schritt 3 der Rekrutierung identifiziert lokale Extrema und bewertet die zugehörigen Zeitpunkte t jeweils mit $v(t) = 4 \cdot n$.
- Sind über die so in den Schritten 1-3 gesammelten Punkte hinaus weitere Punkte zu rekrutieren, so werden diese gleichverteilt aus den bislang nicht rekrutierten Punkten ausgewählt und als ZPs gekennzeichnet.

[25] Eine Klassifizierung der bedingenden Fakten in mehr als zwei Klassen lässt sich analog realisieren.

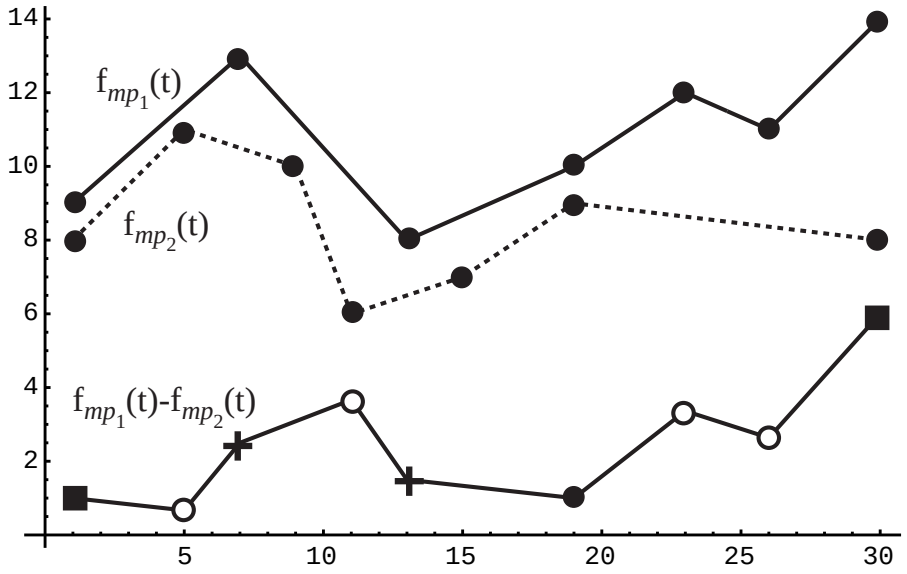


Abbildung 2.1: Beispielhafte Illustration einer Rekrutierungsstrategie.

Für die Einfachheit der Darstellung des Beispiels seien alle definierenden Fakten der Operanden als EPs angenommen. Somit gilt $E_{max} = 7$. Die maximale Anzahl E von Fakten, die durch die Repräsentation des Operationsergebnisses interpoliert werden können, ist somit $F_{max} := E_{max} \cdot 4/3 \approx 9$.

In Schritt 1 der Rekrutierung werden die Intervall-Endpunkte $t = 1$ and $t = 30$ (in der Abbildung dargestellt durch Quadrate) mit einem Relevanzwert von $v(t) = 5 \cdot 2 = 10$ (und somit als EPs) markiert. In Schritt 2 wird dem Zeitpunkt $t = 19$ (dargestellt durch einen schwarzen Kreis) der Wert $v(t) = 3 \cdot 2 = 6$ zugewiesen, da beide Operanden diesen als EP besitzen. Alle weiteren Zeitpunkte zu definierenden Fakten der Operanden werden mit $v(t) = 3$ bewertet, da sie EPs nur für einen Operanden darstellen. Die Heuristik in Schritt 3 identifiziert in diesem Beispiel die Zeitpunkte (dargestellt durch offene Kreise), die zu lokalen Extrema des exakten Operationsergebnisses korrespondieren, und bewertet diese jeweils mit $v(t) = 4 \cdot n = 8$.

Die Zeitpunkte mit den $E_{max} = 7$ höchsten Relevanzwerten sind somit die Intervallendpunkte, die vier Extremalstellen sowie der Zeitpunkt, der zu einem EP beider Operanden korrespondiert. Diese Zeitpunkte werden als EPs markiert. Die noch zu bestimmenden $F_{max} - E_{max} = 2$ ZPs des zu erstellenden Operationsergebnisses werden aus den bewerteten Kandidatenzeitpunkten ($t = 7, 9, 13, 15$) gleichverteilt ausgewählt, so dass hier jeder zweite (in aufsteigender chronologischer Ordnung) berücksichtigt wird (dargestellt durch Kreuze).

Durch das Approximieren entsprechend einer Rekrutierungsstrategie ist die Forderung nach Größenbeschränktheit offensichtlich erfüllbar, da die Anzahl der definierenden Fakten und seiner EPs von der verwendeten Strategie bestimmt werden kann.

Bemerkung 2.3.4 Die Wahl des (für die Anwendung festen) Parameters c erlaubt einen anwendungsspezifischen Kompromiss zwischen Präzision und Repräsentationsgröße von Operationsergebnissen. Die Genauigkeit der Ergebnisrepräsentation hängt

zudem davon ab, welches Repräsentationskonzept für die Interpolation der definierenden Fakten des Operationsergebnisses verwendet, also welche Interpolationsform benutzt wird. Es sei hier erwähnt, dass für einzelne feste (nicht konkatenierte) Operationen, z.B. für die Distanz, die *exakte* Repräsentation der Ergebnisse erzwungen werden kann, nämlich durch geeignete Wahl der Konstante c sowie der Festlegung eines geeigneten Repräsentationskonzeptes für die Ergebnisse der jeweiligen Operationen. Dabei ist zu beachten, dass ein Repräsentationstyp beispielsweise für Ergebnisse der eindimensionalen Distanz und für stückweise lineare Operanden zu definieren ist, vergleiche Forlizzi *et al.* [FGNS00] für die Beschreibung eines solchen Typs und seiner internen Darstellung. Für jeden weiteren Repräsentationstyp, der Bewegung anders als stückweise linear darstellt, wäre jedoch ein entsprechendes Repräsentationskonzept zur Generierung und Speicherung von Distanzergebnissen von Instanzen dieses Typs bereitzustellen. Ferner sind auch diese Konzepte im Allgemeinen nicht ausreichend, wenn die Distanz von heterogenen Operanden, d. h. von Bewegungsrepräsentationen verschiedenen Typs, exakt dargestellt werden soll.

Bemerkung 2.3.5 Das Konzept der Rekrutierungsstrategien kann mit dem Gebiet der Kurvensimplifizierung in Zusammenhang gebracht werden. Die Kurvensimplifizierung als Teilgebiet der algorithmischen Geometrie beschäftigt sich mit der Approximation bzw. der Vereinfachung von Kurven einer bestimmten Gestalt, zumeist von Polylinien. Entsprechende Algorithmen werden zumeist so konzipiert, dass sie bei der Simplifizierung eine gewisse Optimalität garantieren. *Min-#*-Algorithmen sind durch die Angabe einer maximal zulässigen Abweichung der simplifizierten von der ursprünglichen Kurve parametrisiert und berechnen eine solche zulässige Simplifizierung mit minimaler Repräsentationsgröße. *Min-ϵ*-Algorithmen hingegen werden durch eine Obergrenze für die Repräsentationsgröße der Simplifizierung parametrisiert und liefern aus diesen zulässigen Simplifizierungen eine solche, die die Abweichung von der ursprünglichen Kurve minimiert.

Das beschriebene Gerüst für Rekrutierungsstrategien hebt sich von gebräuchlichen Kurvensimplifizierungsalgorithmen ab: zum einen durch die unabhängig von den (für Operanden und Ergebnis verwendeten) Repräsentationstypen, d. h. Kurvenarten, zum anderen durch eine vergleichsweise schnelle Berechnung der Simplifizierung.[26] Ein wesentlicher Vorteil der Rekrutierungsstrategien ist zudem das Konzept der Klassifizierungsmarkierung: Dieses erlaubt Fakten mit verschiedenen Metainformationen zu versehen, was sich bei iterierter Simplizierung und bei iterierter Verknüpfung von Simplifizierungen ausnutzen lässt, wie sie für Anfragen an räumlichtemporale Daten üblich sind. Es ist dabei zu beachten, dass das Konzept der Klassifizierungsmarkierung auch in klassische Kurvensimplifizierungsalgorithmen integriert werden kann: Generell ist für den Simplifizierungsfehler, d. h. für die Abweichung der simplifizierten von der ursprünglichen Kurve, ein Maß vorzugeben; ein solches Maß kann so definiert werden, dass die Relevanz von Fakten mit berücksichtigt wird — bzw. die Kurvenpunkte, die eine größere Verlässlichkeit besitzen, stärker berück-

[26] Während Rekrutierungsstrategien wie die beschriebene simplifizierte Resultate in einer Laufzeit bestimmen, die linear in der Repräsentationsgröße des exakten bzw. ursprünglichen Resultates ist, weisen Kurvensimplifizierungsalgorithmen, die eine Optimalitätsbedingung erfüllen, eine superlineare Laufzeit — schon für einfache Kurven wie stückweise lineare im zweidimensionalen Raum.

sichtigt werden. Einen Überblick über klassische Simplifizierungsalgorithmen gibt die Diplomarbeit von Kötterheinrich [Köt04]. In dieser wird insbesondere auf die Verwendbarkeit der Algorithmen für die Simplifizierung von Trajektorien und zeitvarianten Operationsergebnissen eingegangen. Dunkelmann [Dun05] untersucht in ihrer Diplomarbeit, wie sich die in Bewegungsrepräsentationen enthaltenen Simplifizierungsfehler auf die Ergebnisse von Anfragen auswirken, die an derartige Bewegungsrepräsentationen gestellt werden, vergleiche auch die Ausführungen zur Approximation von Trajektorien in Kapitel 2.1.3.

2.4 Realisierung und Evaluation

Ein Prototyp des hier vorgestellten Rahmenwerks zur Repräsentation von mobilen Objekten und zur Verarbeitung von Bewegungen ist implementiert und in ein objektorientiertes Datenbankmanagementsystem erfolgreich integriert worden. Im Folgenden wird die dieser Implementierung zu Grunde liegende Modellierung vorgestellt. Die für diese Modellierung (und die geleistete Implementierung) zentralen Zielsetzungen sind nachfolgend zusammengefasst:

- Bereitstellung verschiedener Repräsentationstypen bzw. Trajektorienarten und verschiedener Konzepte zur Repräsentation von Bewegungen.
- Bereitstellung von verschiedenen Konstruktions- und Auswertungsalgorithmen für eine jeweilige Trajektorienart.
- Handhabbarkeit verschiedener, praxisrelevanter Arten von definierenden Fakten.
- Abgeschlossenheit gegen beliebige Verkettung von arithmetischen Operatoren, insbesondere wenn diese auf Bewegungsrepräsentationen operieren.
- Bereitstellung verschiedener Rekrutierungsstrategien.

Nach der folgenden Beschreibung der objektorientierten Modellierung des Rahmenwerkes wird in Abschnitt 2.4.2 seine Implementierung und die Anbindung an einen objektorientierten Datenbankkern namens Goodac, der für die Entwicklung von Geo-Anwendungen konzipiert wurde, erläutert. Abschließend wird eine erste praktische Evaluation der Effizienz des Rahmenwerks (bzw. seiner Implementierung) und insbesondere des Mehraufwandes für die Verwendung komplexerer statt stückweise linearer Repräsentationstypen erläutert.

2.4.1 Modellierung

Der zentrale Teil der Modellierung des Rahmenwerkes ist in Form eines Klassendiagramms in Abbildung 2.2 dargestellt. Ein mobiles Objekt wird durch eine Instanz der Klasse `mpoint`$\langle d \rangle$ abgebildet. Zu einem solchen Objekt ist eine Instanz der abstrakten Klasse `Function` assoziiert. Diese Instanz enthält eine interne Repräsentation der Trajektorie des mobilen Objektes und einen Verweis auf eine `FunctionRules`-Instanz.

Die Logik zur Erstellung und Auswertung einer Trajektorie wird von Ableitungen der Klasse `FunctionRules` bereitgestellt. Jede dieser Unterklassen ist Realisierung eines Repräsentationskonzeptes im Sinne der Definition aus Kapitel 2.1.3 und stellt einen Konstruktions- sowie einen Auswertungsalgorithmus durch Implementierung der (in `FunctionRules` abstrakten) Methoden `evaluate()` bzw. `createTrajectory()` bereit. Neben diesen Funktionalitäten sind durch `FunctionRules`-Instanzen auch Parameter zur Erstellung einer Trajektorie gekapselt: So können beispielsweise bei Spline-Interpolationen verschiedene Randbedingungen bei der Erstellung vorgegeben werden; bei ν-Spline-Interpolationen können Parameter die quantifizierte Fähigkeit eines mobilen Objektes zu spontanen Richtungsänderungen abbilden.

Ein wesentlicher Grund für die separate Modellierung der Klassen `Function` und `FunctionRules` besteht darin, dass mehrere Repräsentationskonzepte (wie z.B. Spline-Interpolationen mit verschiedenen Randbedingungen und auch ν-Splines) denselben Repräsentationstyp, d. h. dasselbe Speicherformat für die interne Darstellung, verwenden können. Ein solcher Repräsentationstyp ist durch eine `Function`-Ableitung abgebildet. Diese Modellierung von Repräsentationstypen bietet den Vorteil, an Hand der Klassenzugehörigkeit erkennen zu können, ob zwei Trajektorien dasselbe Speicherformat besitzen. Diese Information ist für die Verarbeitung von Trajektorien, etwa bei Anwendung der Distanzoperation oder anderen arithmetischen Verknüpfungen, dienlich, was in Kapitel 3 näher erläutert wird.[27]

Ein mobiles Objekt vom Typ `mpoint`$\langle d \rangle$ enthält neben dem Verweis auf seine kontinuierliche Bewegungsrepräsentation zusätzlich Referenzen auf die zu seiner Bewegung bekannten, definierenden Fakten. Diese Fakten sind als Instanzen der Klasse `InTimeData` aggregiert und bestehen aus zumindest einem Zeitpunkt und einem Byte für eine Klassifizierungsmarkierung, die von Rekrutierungsstrategien verwendet werden kann. Die in Abschnitt 2.3.2 geschilderte Beispielstrategie verwendet dieses Attribut, um EPs von ZPs zu unterscheiden. Die Klasse `InTimeData` kann abgeleitet werden, um verschiedene Formate von Fakten abzubilden. Die Ableitung `InTimePositionData` dient beispielsweise der Darstellung von Fakten, die aus einer Positionsangabe zu einem Zeitpunkt bestehen. Weitere Unterklassen sind notwendig, um weitere diskrete Informationen zu Zeitpunkten zu modellieren, etwa (gerichtete bzw. ungerichtete) Geschwindigkeit oder Beschleunigung.

Die Methode `getInTimeData()` der Klasse `mpoint`$\langle d \rangle$ liefert die aggregierten `InTimeData`-Instanzen zurück und realisiert somit die in Abschnitt 2.1.3 geforderte Abrufbarkeit der definierenden Fakten. Abhängig von der konkreten Unterklasse von `Function` sind Teile der definierenden Fakten (wie Positionsangaben) nicht explizit vorgehalten, sondern werden durch Aufrufe der Methode `evaluate()` der Klasse `Function` rekonstruiert: In diesem Fall sind die zum mobilen Objekt aggregierten Fakten Instanzen der Basisklasse `InTimeData` und halten Zeitpunkte t vor, die durch Auswertung der Bewegungsrepräsentation zu Fakten $(t, f(t))$ vervollständigt werden können.

[27] Beispielsweise Splines und ν-Splines sind jeweils segmentweise durch Polynome eines einheitlichen Grades definiert. Daher ist ein geeignetes Speicherformat definierbar, das sowohl Spline- als auch $\nu-$Spline-Interpolationen darstellen kann. Dies kann beispielsweise für Addition oder Durchschnittbildung von Spline- und ν-Spline-Trajektorien ausgenutzt werden, da mit dem Speicherformat der Argumente auch die Ergebnisse dieser Operationen beschrieben werden können.

Um eine Repräsentation einer Verarbeitung von Bewegungen zu erzeugen, wird eine Rekrutierungsstrategie verwendet. Diese Strategien sind als Instanzen der Klasse `Heuristics` modelliert. Eine solche Instanz liefert einen Algorithmus, wie den in Abschnitt 2.3.2 beispielhaft beschriebenen, um die Fakten zu bestimmen, welche durch die Ergebnisrepräsentation zu interpolieren sind. Neben dem `Heuristics`-Objekt ist für die Konstruktion von Operationsergebnissen zusätzlich eine `Bindings`-Instanz erforderlich, die das Repräsentationskonzept bzw. eine Ableitung von `FunctionRules` festlegt, die zur Interpolation der ausgewählten Fakten verwendet werden soll. Verschiedene Formen von Rekrutierungsstrategien und Bindungsregeln können durch Unterklassen von `Heuristics` bzw. `Bindings` abgebildet werden. Für jede Anwendung des Rahmenwerks sollte jedoch ein globales `Heuristics`-Objekt verbindlich festgelegt sein, welches wiederum eindeutig ein `Bindings`-Objekt assoziiert.

Arithmetische Operationen, die sich in das in Kapitel 2.1.2 vorgestellte Typsystem integrieren lassen und die — in ihrer gelifteten Version — zeitvariante Typen als Eingabe erwarten, sind als Ableitungen der Klasse `Operation` realisiert. Eine solche Ableitung implementiert eine Methode `evalOp()`, die für nicht zeitvariante Operanden das Ergebnis der Operation zurückliefert. Zusätzlich ist eine Methode `checkSignature()` zu implementieren, die überprüft, ob die Liste der Operanden der Signatur der Operation entspricht. Wenn als Resultat der Operation ein zeitvariantes Objekt zu konstruieren ist, so geschieht dies durch die Methode `doOperation()` der global zu verwendenden Instanz von `Heuristics`. Dieser Methode werden ein `Operation`-Objekt sowie die Argumente der Operation übergeben. Bei der Ausführung der Methode werden folgende Schritte durchgeführt:

1. Es wird über Aufruf von `checkSignature()` überprüft, ob die Typen der Operanden mit der Signatur der Operation übereinstimmen.

2. Die durch das `Heuristics`-Objekt gekapselte Rekrutierungsstrategie wird ausgeführt.

   (a) Zunächst werden Fakten (bzw. deren Zeitpunkte) gesammelt, die das Resultat beeinflussen könnten. Hierfür werden unter anderem Fakten der Argumente durch Aufruf der `Heuristics`-Methode `getConstrainingFacts()` betrachtet. Die Konstruktion der Fakten des Ergebnisses geschieht durch die Methode `evalOp()` des übergebenen `Operation`-Objektes. Diese Methode stützt sich auf Methoden der Operanden zur Funktionsauswertung ab, um die Bewegungsszustände der involvierten Objekte zu spezifizierten Zeitpunkten festzustellen.

   (b) Je nach Rekrutierungsstrategie werden weitere Fakten gesammelt, zu deren Zeitpunkten nicht zwingend ein Faktum eines Operanden existieren muss. Auch hierfür wird die Methode `evalOp()` verwendet.

   (c) Entsprechend der Bewertungen der gesammelten Fakten sowie weiteren Kriterien wählt die Rekrutierungsstrategie die Fakten aus, die durch das Operationsergebnis zu interpolieren sind.

3. Das vom globalen `Heuristics`-Objekt referenzierte `Bindings`-Objekt ist für die abschließende Erstellung der Interpolationsfunktion des Operationsergebnisses verantwortlich. Hierfür wird ein geeignetes Repräsentationskonzept, d. h. eine `FunctionRules`-Ableitung ausgewählt (und eine konkrete Instanz eventuell geeignet parametrisiert). Der Konstruktionsalgorithmus dieser Instanz erzeugt schließlich ein Objekt der zur `FunctionRules`-Ableitung gehörigen Unterklasse von `Function`.

Das Operationsergebnis wird durch eine Instanz des Typs `mreal` abgebildet, welcher analog zum Datentyp `mpoint`$\langle d \rangle$ eine Repräsentation eines zeitvarianten Datums modelliert, also Verweise auf eine Repräsentation vom Typ `Function` sowie auf definierende Fakten enthält.

2.4.2 Realisierung

Oben vorgestellte Modellierung des Rahmenwerks ist als Erweiterung des GIS-Datenbankkerns GOODAC [BVH96, Voi98] in der Programmiersprache `C++` implementiert. GOODAC wurde für die Entwicklung von GIS-Anwendungen konzipiert und ist basierend auf dem objekt-orientierten Datenbanksystem OBJECTSTORE [LLOW91] realisiert. Zusätzlich wurde als Ergänzung der Implementierung des Rahmenwerkes die Anwendung GRAPHICTOOL zur grafischen Darstellung von Repräsentationen von Bewegungen und zeitvarianten Operationsergebnissen entwickelt, um die Beurteilung verschiedener Repräsentationskonzepte und Rekrutierungsstrategien visuell zu unterstützen.[28] In Abbildung 2.3 ist ein Screenshot dieser Anwendung zu sehen, in dem eine kreisförmige Bewegung zum einen durch eine stückweise lineare, zum anderen durch eine Spline-basierte Trajektorie repräsentiert und visualisiert ist.

2.4.3 Praktische Evaluation

Um zu belegen, dass der im vorgestellten Rahmenwerk realisierte Ansatz, verschiedene Konzepte zur Bewegungsrepräsentation zu integrieren, praktisch einsetzbar und ohne dramatische Laufzeiteinbußen zu verwenden ist, sind Experimente durchgeführt worden, die insbesondere die Kosten für die Verwendung nicht-linearer Trajektorien evaluieren. In Tabelle 2.1 werden Konstruktions- und Auswertungszeiten für stückweise lineare und kubische B-Spline-Trajektorien — jeweils konstruiert mit zufällig gewählten, zu interpolierenden Fakten — gegenübergestellt. Die in Tabelle 2.1 aufgelisteten Ergebnisse wurden auf einer Sun Enterprise 250 mit zwei 400 MHz Prozessoren und 1.5 GB RAM gewonnen und über 5 Durchläufe gemittelt. Es ist zu beachten, dass ein nicht unwesentlicher Teil der Laufzeit für das Allozieren von persistenten GOODAC-Objekten und für den Zugriff auf diese verwendet wird; die relative Performanz verschiedener Repräsentationstypen kann sich ändern, wenn nur transiente Objekte benutzt werden, wie z.B. in Hauptspeicherdatenbanken. Die „Konstruktionszeit“ für die stückweise linearen Trajektorien ergibt sich fast vollständig aus dem

[28] Das GRAPHICTOOL wurde zu großen Teilen von Katrin Boege und Christian Breimann realisiert. Eine genauere Beschreibung des Funktionsumfanges und der Details dieser Implementierung findet sich in der Diplomarbeit von Boege [Boe02].

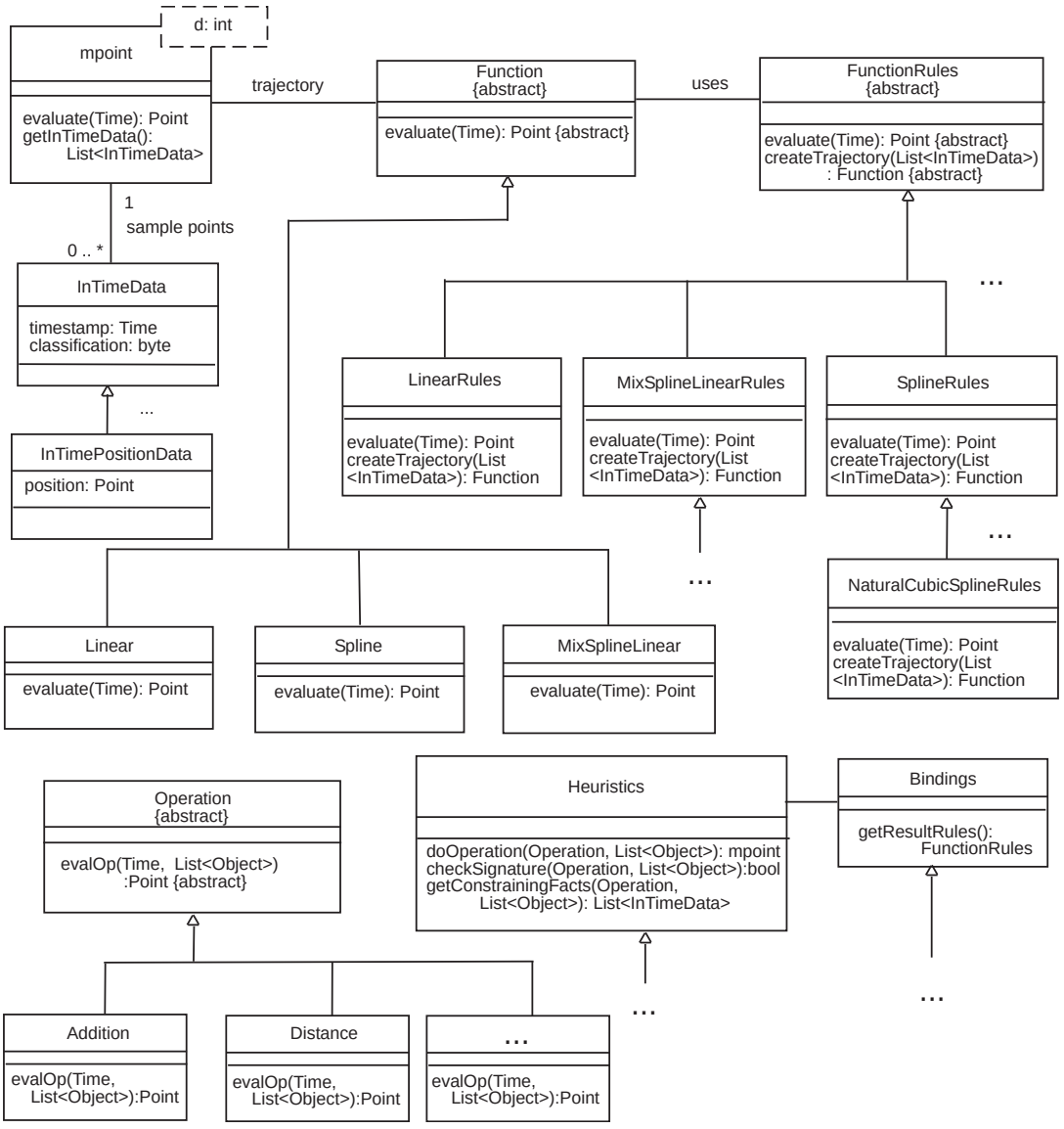


Abbildung 2.2: Klassendiagramm zur Repräsentation mobiler Objekte.

Tabelle 2.1: Laufzeitvergleiche für Konstruktion und Auswertung kubischer Spline- und stückweise linearer Trajektorien.

Fakten	Konstruktionen			1000 Evaluationen		
	Linear	Spline	Faktor	Linear	Spline	Faktor
40	0.314	0.510	1.61	0.793	0.933	1.33
160	0.693	1.219	1.76	0.892	1.174	1.32
320	1.461	2.722	1.86	0.964	1.307	1.36
640	5.783	8.814	1.52	1.014	1.518	1.50

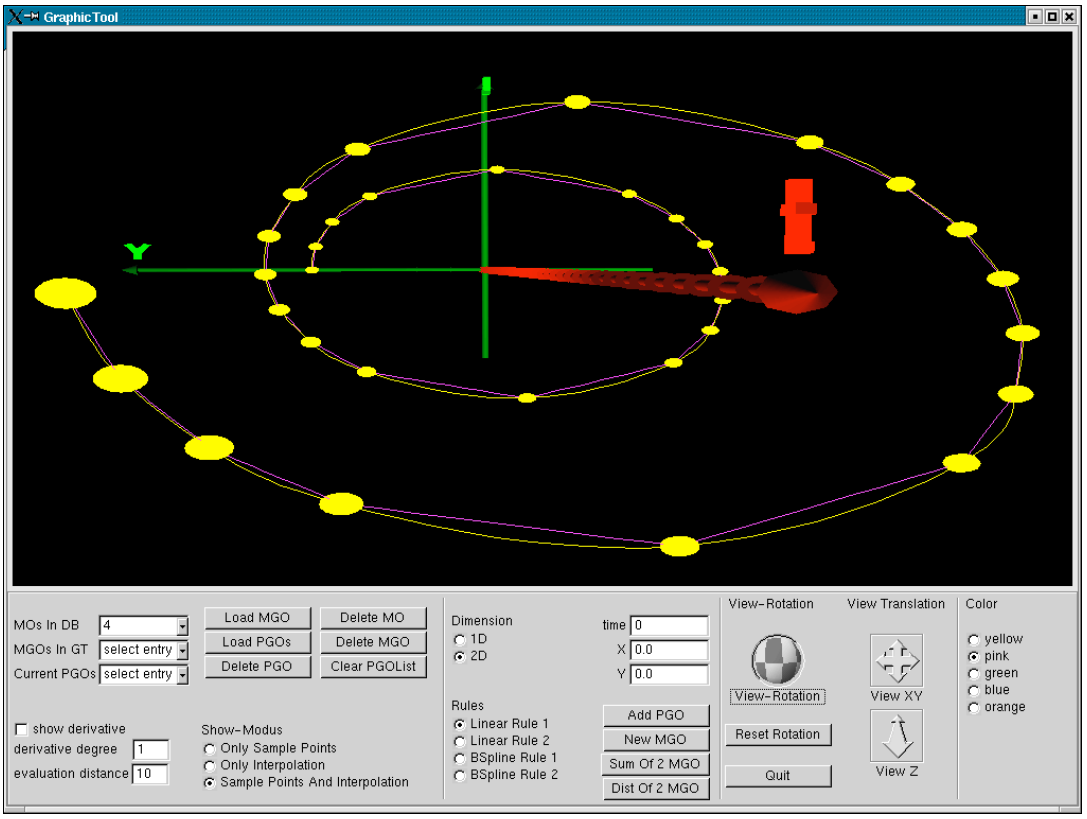


Abbildung 2.3: Das GraphicTool zur Visualisierung von Bewegungsrepräsentationen.

Allozieren von Goodac-Objekten, die den Fakten der Trajektorie entsprechen. Die Differenz zwischen den Konstruktionszeiten für lineare und Spline-Trajektorien entspricht daher in etwa dem Aufwand für die Berechnung der Spline-Interpolation.[29] Die Entwicklung der praktischen Laufzeiten für Konstruktion und Auswertung mit der Anzahl der Fakten spiegeln im Wesentlichen die in Abschnitt 2.2.2 hergeleiteten Komplexitäten wider.

Innerhalb der gewählten Umgebung ist somit der zusätzliche Aufwand, der aus der Verwendung des nicht-linearen Repräsentationstyps resultiert, moderat: Sowohl Konstruktions- als auch Evaluationszeit sind für kubische B-Spline- gegenüber stückweise linearen Trajektorien um weniger als den Faktor 2 erhöht — sogar für eine hohe dreistellige Anzahl an zu interpolierenden Fakten.[30] Beim Wiederholen der Experimente für je 10 Evaluationen bestätigten sich die Beobachtungen: Die mit einem Faktor 100 skalierten Resultate sind nahezu identisch mit den in Tabelle 2.1 dargestellten Ergebnissen.

Der Mehraufwand an Speicherplatz, der sich für die explizite Darstellung von B-Spline-Trajektorien gegenüber der impliziten Darstellung von stückweise linearen Trajektorien ergibt, entspricht pro Bewegungsrepräsentation dem Speicheraufwand von vier zusätzlichen definierenden Fakten, wie in Abschnitt 2.2.2 bereits aus-

[29] Die absolute Berechnungszeit ist wiederum stark durch die Verwendung der Goodac-spezifischen Basisdatentypen beeinflusst, d. h. erhöht.

[30] Insbesondere für solche „großen“ Trajektorien scheint der Aufwand für die persistente Datenhaltung diesen Faktor noch zu verringern.

geführt.[31]

Die Realisierbarkeit und die Laufzeiteffizienz für weitere Anfragetypen (abseits der Abfrage einer Position eines mobilen Objektes zu einem spezifizierten Zeitpunkt) werden im folgenden Kapitel 3 für verschiedene Typen zur Bewegungsrepräsentation genauer untersucht.

[31] Werden Sub-Spline-Interpolationen statt einer klassischen Spline-Interpolation verwendet, ergibt sich der erhöhte Speicheraufwand ebenfalls aus den theoretischen Betrachtungen wie in Abschnitt 2.2.2 beschrieben.

Kapitel 3

Analysieren und Verarbeiten von heterogen typisierten Bewegungsrepräsentationen

Im vorangegangenen Kapitel wurde motiviert, dass die realistische Beschreibung von Bewegungen Repräsentationstypen bzw. Trajektorienarten erfordert, deren Verwendung jeweils zu erhöhtem Berechnungs- und Implementierungsaufwand führt. Dies wirft die Frage auf, ob und wie Anfragen und Analyseoperationen innerhalb eines Systems, das die Bewegngen mobiler Objekte realistisch darstellt, effizient beantwortet bzw. durchgeführt werden können. Weiterhin wurde im vorangegangenen Kapitel argumentiert, dass für verschiedene Arten von Bewegungen jeweils verschiedene Typen von Bewegungsrepräsentationen geeignet sind, d. h. verschiedene mathematische Beschreibungen von Bewegung verwendet werden sollten. Dies impliziert die weitergehende und in diesem Kapitel erörterte Fragestellung, ob und wie effizient sich Algorithmen zur Anfragebearbeitung und zur Analyse von Bewegungen realisieren lassen, so dass diese generisch, d. h. möglichst unabhängig von den Repräsentationstypen der involvierten Bewegungen, verwendbar sind [BHSV06].

3.1 Zielrichtung

Die Betrachtungen zur Laufzeiteffizienz verschiedener Bewegungsrepräsentationstypen in Kapitel 2 beschränkten sich auf die Aufgaben der Konstruktion und Auswertung von Bewegungsrepräsentationen bzw. Trajektorien. Für verschiedene, Bewegung verarbeitende Anwendungen müssen jedoch wesentlich komplexere algorithmische Aufgaben bewältigt werden; entsprechend enthalten die in Abschnitt 2.1.1 referenzierten Datenmodelle jeweils eine Vielzahl an Analyseoperationen und Anfragetypen. Im Anwendungsbeispiel des Flugverkehrmanagements sind dies beispielsweise raumzeitliche Bereichsabfragen, die zeitvariante Sortierung von Flugzeugen nach räumlichen Kriterien wie ihrer Höhe sowie die Erkennung von möglichen Kollisionen oder Unterschreitungen von kritischen Distanzen zwischen Flugzeugen. Wie bereits in der Einleitung erläutert, ist für solche Anwendungen die Verwendung nicht nur von einem, sondern von vielfältigen Bewegungsbeschreibungstypen ratsam, da die Bewegungen der zu betrachtenden mobilen Datenobjekte unterschiedliche Cha-

rakteristika aufweisen. Somit ist das Ziel dieses Kapitels motiviert, Lösungen zu komplexen algorithmischen Aufgabenstellungen wie den oben genannten anzubieten, ohne dabei die Verwendung eines einzelnen konkreten Bewegungsbeschreibungstypen vorauszusetzen.

Es wird an Hand von Beispielen — und ausgehend von diesen verallgemeinernd — aufgezeigt, dass dieses Ziel durch angemessene Erweiterung des in Kapitel 2 vorgestellten Rahmenwerks erreichbar ist: Es sind Lösungen der oben genannten algorithmischen Herausforderungen innerhalb des Rahmenwerks implementierbar, die für Instanzen jeder konkreten Unterklasse der abstrakten Klasse `Function` anwendbar sind; diese Lösungen akzeptieren zudem als Eingaben Bewegungsrepräsentationsmengen, die heterogen bezüglich der Typen ihrer einzelnen Elemente sind. Die Bereitstellung solcherart generischer algorithmischer Lösungen kann durch Generalisierung von Algorithmen erfolgen, die originär nur für einen konkreten Bewegungsrepräsentationstyp, z.B. stückweise lineare Kurven, formuliert wurden. In diesem Kapitel wird ausgeführt, in welcher Art und Weise eine solche Generalisierung bekannter Bewegungen verarbeitender Algorithmen erfolgen kann und wie diese Generalisierungen innerhalb des vorgestellten Rahmenwerks zu modellieren sind. Das Grundkonzept hierfür besteht darin, dass sich Anfragen bzw. Algorithmen für die Bewegungsverarbeitung ausschließlich auf die minimalistische, von Bewegungsrepräsentationstypen zu erfüllende Schnittstelle abstützen, d. h. auf die „Basisanfrage“ nach der Position eines mobilen Objektes zu einem spezifizierten Zeitpunkt. Somit bliebe diese Basisanfrage idealerweise die einzige, die spezifisch für einen jeden Bewegungsrepräsentationstyp zu realisieren wäre.

Eine Strategie, die Korrektheit solcher generalisierter Algorithmen zu gewährleisten, ergibt sich, wenn zu jeder einzelnen repräsentierten Bewegung zumindest eine Bewegungsrestriktion, wie eine oberen Schranke für Geschwindigkeit oder Beschleunigung, als bekannt vorausgesetzt werden kann. Die Beschreibung und Integrierung dieser Strategie und der zu Grunde liegenden Kenntnisse der Restriktionen in das vorgestellte Rahmenwerk ist Hauptgegenstand dieses Kapitels. Als Konsequenz erlaubt das entsprechend modifizierte Rahmenwerk die Erweiterung um Bewegungsrepräsentationstypen derart, dass die im Rahmenwerk integrierten, generalisierten Algorithmen ohne Modifikation das verlässliche Operieren auch auf Bewegungsrepräsentationen des neuen Typs ermöglichen.

3.2 Aufgabenstellungen der Verarbeitung von Bewegungen aus verwandten Forschungsfeldern

Das Abrufen des Bewegungsstatus eines mobilen Objektes zu einem spezifizierten Zeitpunkt ist über die minimalistische Schnittstelle der Klasse `Function` des hier vorgestellten Rahmenwerkes möglich; die Beantwortung von Operationen mit hiervon abweichender Signatur oder von Anfragen, für deren Beantwortung mehrere mobile Objekte zu betrachten sind, bleibt jedoch zu ergänzen. Insbesondere bleiben auch zu `evaluate(t)` orthogonale Operationen zu realisieren, die Zeitpunkte statt Positionen ermitteln. Eine komplette Auflistung der Aufgabenstellungen, die für Bewegungen verarbeitende Systeme relevant sind, scheint jedoch nicht möglich. Es sei

hier unter anderem auf das in Abschnitt 2.1.2 in Teilen wiedergegebene Operationensystem von Güting*et al.* [GBE+00] verwiesen, das eine Vielzahl von Anfragetypen listet, die in Bewegungen verarbeitenden Datenbanken praxisrelevant sind.

Nachfolgend werden weitere Bewegungen verarbeitende Operationen und Problemstellungen umrissen, zu denen in verwandten Forschungsfeldern Algorithmen (sowie Modelle zur Algorithmenentwicklung) vorgestellt wurden. Solche Algorithmen gehen zumeist von der Verwendung eines konkreten Bewegungsrepräsentationstyps aus. Im weiteren Verlauf dieser Arbeit werden beispielhaft Generalisierungen zu solchen Algorithmen angegeben, die ihre Bewegungsrepräsentationstyp-unabhängige Anwendung durch Abstützung auf die minimalistische Schnittstelle des Rahmenwerks ermöglichen.

Kollisionswarnung und -erkennung Die Meldung von sowie die Warnung vor Kollisionen gehört in vielen Bewegungen analysierenden (und koordinierenden) Anwendungen zu den wesentlichen Aufgabenstellungen; einen Überblick über die entsprechend vielfältige Literatur hierzu bieten beispielsweise Lin und Manocha [LM04, LMCG97]. Viele dieser Methoden sind speziell für Echtzeitszenarien konzipiert. In diesen werden die aktuellen Bewegungszustände von Objekten analysiert — zumeist um die zukünftigen Bewegungen der Objekte zu antizipieren [HAFG95, Kah91, KY00]. Die Zielsetzung dieser Methoden ist, alle kritischen Konstellationen (wie Kollisionen) zwischen den beobachteten Objekten zu erkennen und hierfür möglichst wenig Anfragen an die Bewegungszustände der zu betrachtenden Objekte zu stellen. Da für Echtzeitanwendungen ausgelegt, nutzen diese Methoden keine Bewegungsrepräsentationen, die über einen zeitlichen Verlauf definiert sind; folgerichtig treffen sie auch keine Annahmen über den Typ bzw. die funktionale Beschreibung solcher Repräsentationen.

Sind die zu beobachtenden, sich bewegenden Objekte mit einer Ausdehnung modelliert, so erstellen und verwenden viele der entsprechenden Methoden hierarchische Dekompositionen der Objekte [GNRZ02, KSS00, BEG+04]. In dieser Arbeit wird neben der Kollisionserkennung die folgende Form der Kollisionswarnung näher betrachtet: Als kritische Ereignisse sind nicht (bzw. nicht nur) Kollisionen, sondern Unterschreitungen einer kritischen Distanz δ zu melden. Diese Aufgabenstellung der Kollisionswarnug für punktförmige Objekte entspricht zugleich der der Kollisionserkennung für kugelförmig ausgedehnte Objekte: Eine Unterschreitung einer Distanz von δ für zwei sich bewegende Objekte ohne Ausdehnung entspricht einer Kollision zweier (sich gleicherart bewegender) Objekte, deren Ausdehnung als Kugel mit Radius $\delta/2$ modelliert ist. Eine Vielzahl solcher Aufgabenstellungen im Kontext räumlich-temporaler Datenhaltung und -analyse für konzentrisch erweiterte, punktförmige Objekte wird von Trajcevski *et al.* [TWHC04] betrachtet.[1]

Die Methodik von Hayward *et al.* [HAFG95] und das konzeptionell verwandte Modell von Kahan [Kah91] nutzen zur Kollisionserkennung Bewegungsrestriktionen wie Beschränkungen von Geschwindigkeit und Beschleunigung, die für jedes be-

[1] Die räumliche Erweiterung der sich bewegenden Objekte ist bei Trajcevski *et al.* durch die Zielsetzung motiviert, Unsicherheiten bezüglich der Positionen der Objekte zu modellieren, um alle möglichen Konfigurationen der (in beschränktem Maße unpräzise) abgebildeten Realität zu erfassen.

trachtete Objekt als bekannt vorausgesetzt werden. Hierdurch kann für Paare von Objekten bestimmt werden, zu welchem nächsten Zeitpunkt eine Kollision möglich wird. Die Algorithmen zur Kollisionserkennung identifizieren hierüber die Objekte, für die eine Kollision in näherer Zukunft zu erwarten ist und die daher genauer zu beobachten sind. Sind zu den betrachteten Objekten keine Beschränkungen ihres Bewegungsverhaltens bekannt, kann die Korrektheit der Ergebnisse dieses Ansatzes jedoch nicht garantiert werden. Ähnliche, Bewegungsrestriktionen voraussetzende Konzepte sind zur räumlich-temporalen Indizierung von Objekten von Prabhakar *et al.* [PXK+02] und zur Behandlung von Ungenauigkeiten und Ungewissheiten von zu verwaltenden räumlich-temporalen Daten von Cheng *et al.* [CKP04], von Pfoser und Jensen [PJ99, PT01] sowie von Trajcevski *et al.* [TWHC04] vorgestellt worden.

Die Darstellung von Konzepten und Sachverhalten zur Generalisierung von Algorithmen erfolgt in dieser Arbeit insbesondere an Hand der Beispielaufgabe der Kollisionserkennung für Objekte, deren Bewegung in nur einer Dimension betrachtet wird. Zusätzlich wird punktuell auf die verallgemeinerte Aufgabenstellung der Kollisionserkennung in höheren Dimensionen eingegangen.

Kinetische Datenstrukturen Zu einer Vielzahl von algorithmischen Problemstellungen im Kontext der Bewegungsverarbeitung und -analyse sind mit *kinetischen Datenstrukturen* geeignete Lösungen präsentiert worden, vergleiche [Bas99, BG99]. Eine kinetische Datenstruktur (kurz: KDS) für eine Menge sich bewegender Objekte hält eine zeitvariante Beschreibung einer kombinatorischen Eigenschaft dieser Menge, beispielsweise ihrer konvexen Hülle, aufrecht. Eine Übersicht über Ergebnisse aus dem Bereich kinetischer Datenstrukturen liefert Guibas [Gui98].

Obwohl kinetische Datenstrukturen die Bewegungen der betrachteten Objekte als durch stetige Funktionen repräsentiert voraussetzen, ändert sich die durch eine kinetische Datenstruktur aufrecht zu erhaltende kombinatorische Eigenschaft dieser Objekte nur zu diskreten Zeitpunkten. Die Bestimmung dieser Zeitpunkte erfolgt durch Identifizierung der Nullstellen von so genannten *Zertifikatfunktionen*; solange keine Vorzeichenwechsel dieser Funktionen erfolgt sind, bleibt die aktuelle Beschreibung der kombinatorischen Eigenschaft der Objekte korrekt. Solche Zertifikatfunktionen sind jeweils durch eine feste Anzahl der sich bewegenden Objekte bzw. ihrer Bewegungsrepräsentationen parametrisiert und ihre Vorzeichenwechsel werden durch Bestimmen ihrer Nullstellen identifiziert. Im Folgenden sei die Nullstellenbestimmung zu einer Zertifikatfunktion auch als *Primitivoperation* oder *Primitiv* bezeichnet.

In Abbildung 3.1 ist die Aufrechthaltung der kinetischen konvexen Hülle bzw. des kinetischen Paares mit geringstem Abstand jeweils für eine Menge von punktförmigen Objekten, deren Bewegungen durch Funktionen f_i repräsentiert werden, skizziert. Die konvexe Hülle dieser Punkte ändert sich genau dann, wenn die Orientierung dreier dieser Objekte, von denen zwei sich auf der konvexen Hülle befinden, sich ändert. Ein solches Ereignis fällt mit dem Vorzeichenwechsel einer Determinantenfunktion zusammen, die durch die drei betrachteten Objekte bzw. ihre Bewegungsrepräsentationen parametrisiert ist. Ebenso ändert sich das Paar mit geringstem Abstand innerhalb einer Menge sich bewegender Objekte genau dann, wenn sich das Vorzeichen der Differenz der Distanzen des Paares mit dem bislang geringsten Ab-

stand sowie eines neuen Paares (mit fortan geringstem Abstand) ändert, vergleiche jeweils Basch [Bas99]. Die zu realisierenden Primitive für kinetische Datenstrukturen zu den genannten Problemstellungen sind die Nullstellenbestimmung der Determinantenfunktion bzw. der Differenz der Distanzen zweier Bewegungsrepräsentationen.

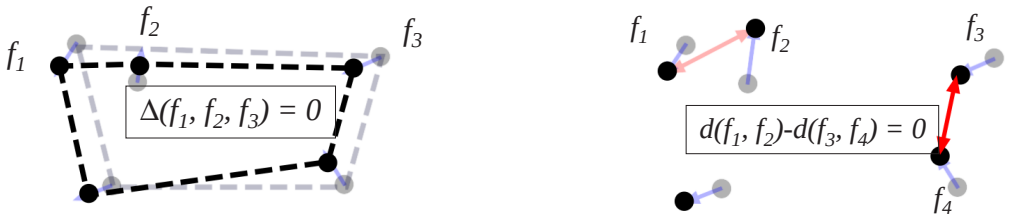


Abbildung 3.1: Aufrechterhaltung der kinetischen Datenstrukturen der konvexen Hülle (links) und des Paares mit minimalen Abstand (rechts) durch Nullstellenbestimmung.

Hauptsächlich um für kinetische Datenstrukturen eine aussagekräftige Effizienzanalyse zu ermöglichen, wird in der Literatur zu kinetischen Datenstrukturen die Annahme zu Grunde gelegt, dass eine jede Bewegungsrepräsentation in jeder Dimension durch eine polynomiale Funktion in der Zeit zu beschreiben ist. Allerdings schätzen maßgebliche Mitbegründer des Forschungsfeldes der kinetischen Datenstrukturen die Beschränkung auf die Klasse der Polynome als wahrscheinlich zu restriktiv ein, um Bewegungen für praktische Anwendungen hinreichend realistisch darzustellen.[2] Die nur bedingte Eignung von Polynomen zur Bewegungsrepräsentation wird in Abschnitt 2.2.3 sowie vor dem Hintergrund der Verarbeitung von Bewegungen genauer in Abschnitt 3.4.1 erläutert.

Modellierungen mobiler Datenobjekte Der bereits in Kapitel 2 genauer betrachtete Forschungsbereich räumlich-temporaler Datenbanken hat zahlreiche Ansätze zur Modellierung mobiler Datenobjekte hervorgebracht, vergleiche Kapitel 2 sowie etwa [FGNS00, GBE+00, CKP04, PT01, TWHC04, YdC95]. Lema *et al.* [LFG+03] beschreiben detailliert Realisierungen der Operationen des in Abschnitt 2.1.2 skizzierten Modells von Güting *et al.* [GBE+00]. Diese Realisierungen sind allerdings nur für stückweise lineare Bewegungsrepräsentationen konzipiert. Mount *et al.* [MNP+04] präsentierten ein Rahmenwerk, das das Konzept der kinetischen Datenstrukturen für Echtzeitszenarien nutzbar macht: Anders als im Datenmodell, das üblicherweise der Analyse kinetischer Datenstrukturen zu Grunde liegt, wird keine komplette Kenntnis über die Bewegungen der Objekte vorausgesetzt, sondern die zukünftigen Positionen von Objekten werden geschätzt.[3] Diese Schätzungen von einzelnen Positionen werden aktualisiert, wann immer die Schätzungen nicht mehr

[2] Die von Basch in seiner Doktorarbeit getroffene Aussage im genauen Wortlaut: „Clearly, the model of motion we have adopted in this thesis — all coordinates are polynomial functions of time — is too restrictive to be of much use in applications, although it is perfectly adequate for theoretical purposes.“ [Bas99, p. 103]

[3] Der entsprechende Ansatz fußt auf dem von Kahan [Kah91] vorgestellten Modell.

zuverlässig genug sind, um mit einer hohen Wahrscheinlichkeit die Korrektheit der Beschreibung der aufrecht zu haltenden kombinatorischen Eigenschaft anzunehmen. Die Korrektheit kann hingegen nur zugesichert werden, wenn Bewegungsrestriktionen zu allen Objekten bekannt sind und die Schätzungen konservativ auf Grundlage dieser Restriktionen gewonnen werden. Dies wird jedoch von Mount *et al.* nicht näher ausgeführt; stattdessen wird das Arbeiten mit nicht konservativen, aus Extrapolation von bisherigem Bewegungsverhalten gewonnenen Schätzungen beschrieben.

Das von Mount *et al.* referenzierte Modell von Kahan ist zudem für die Modellierung von räumlich-temporalen Daten mit Unsicherheiten aufgegriffen worden und unter Aspekten der wettbewerblichen Analyse (*engl.: competitive analysis*) von *online*-Algorithmen[4] untersucht worden, vergleiche Bruce [BHKR05] und die dort enthaltenen Referenzen.

Ein Forschungsgebiet, das ebenfalls Implikationen für die Algorithmik mobiler Objekte bietet, stellen die bereits in Abschnitt 3.2 beschriebenen Constraint-Datenbanken dar. Deren Anfragebearbeitung durch Lösen von Ungleichungssystemen lässt sich auf durch Funktionen dargestellte Daten und insbesondere auf Bewegungen anwenden, was in Abschnitt 3.6.3 genauer beleuchtet wird. Zudem lässt sich, speziell für Bewegungsdaten, diese Form der Anfragebearbeitung mit alternativen Ansätzen, etwa mit *plane-sweep*-Techniken aus der algorithmischen Geometrie, wie sie im folgenden Abschnitt erläutert werden, kombinieren, was von Su *et al.* [SXI01] genauer ausgeführt wird.

Geometrische Segmentschnittalgorithmen Für die Kollisionserkennung für durch eindimensionale Funktionen in der Zeit repräsentierte Bewegungen können Algorithmen zur Bestimmung der Schnittpunkte von Kurven bzw. Segmenten im zweidimensionalen Raum verwendet werden.

Das plane-sweep-Verfahren[5] von Bentley und Ottmann [BO79] verarbeitet Liniensegmente im (t, y)-Raum von links nach rechts (d. h. aufsteigend entlang der als zeitlich zu interpretierenden Dimension). Dieser Algorithmus nutzt aus, dass zeitlich unmittelbar vor einem Schnittpunkt zwei Segmente hinsichtlich ihrer y-Koordinaten benachbart sein müssen. Bei Antreffen eines neuen Segmentes wird dieses daher auf Schnittpunkte nur mit den (bis zu zwei) bzgl. der y-Koordinate benachbarten Segmenten überprüft.[6] Analog werden zwei Segmente auf Schnitt getestet, wenn ein dazwischen liegendes Segment „verschwindet“. Dies ist im linken Teil der Abbildung 3.2 für das Zeitintervall $[t_a, t_b]$ veranschaulicht. Zusätzlich werden nach Antreffen eines Schnittpunktes die beteiligten Segmente mit ihren („nach“ dem Schnittpunkt aktuellen) neuen Nachbarsegmenten auf Schnitt getestet.

Der Algorithmus von Bentley und Ottmann lässt sich auch für die Aufgabe der Aufrechterhaltung einer *kinetischen sortierten Liste*, wie sie von Basch [Bas99] formuliert wurde, verwenden. Auch für diese Problemstellung sind die Zeitpunkte zu

[4]Eine Einführung in das Gebiet der online-Algorithmen und der wettbewerblichen Analyse bieten Phillips und Westbrook [PW99].

[5]Eine Erläuterung des plane-sweep-Paradigmas sowie weiterer Techniken der algorithmischen Geometrie geben Preparata und Shamos [PS85].

[6]Die beiden Endpunkte eines Segmentes entsprechen dem zeitlichen Beginn und Ende der Bewegungsrepräsentation, als das das Segment verstanden werden kann.

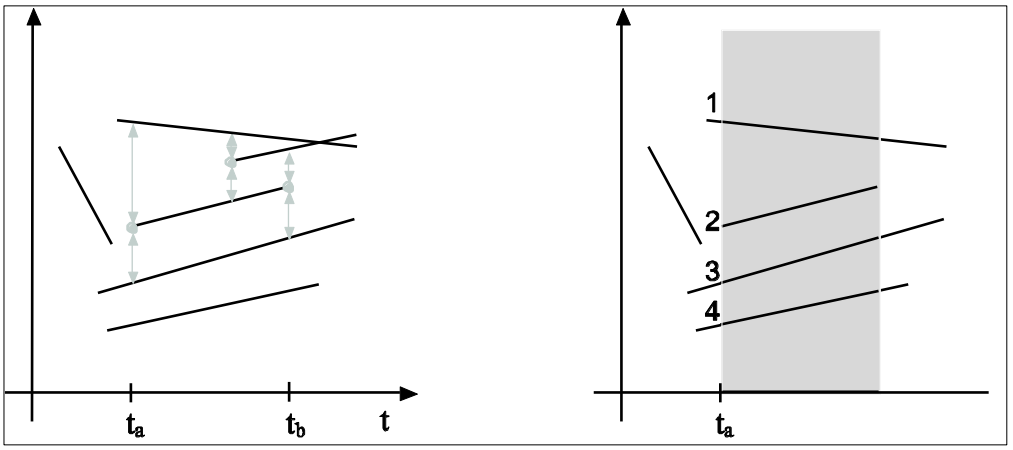


Abbildung 3.2: Bestimmung der Schnittpunkte von Liniensegmenten.

identifizieren, an denen zwei Objekte der Eingabe dieselbe Position im eindimensionalen Raum, d. h. bezüglich ihrer y-Koordinate, einnehmen. In Abbildung 3.3 sind diese Zeitpunkte für eine Beispieleingabe sich bewegender Objekte durch Kreise gekennzeichnet. Wie bereits für kinetische Datenstrukturen im Generellen erläutert, fallen diese Ereignisse mit den Nullstellen von Zertifikatfunktionen — hier der Differenz zweier Bewegungsrepräsentationen f und g der Eingabe — zusammen.

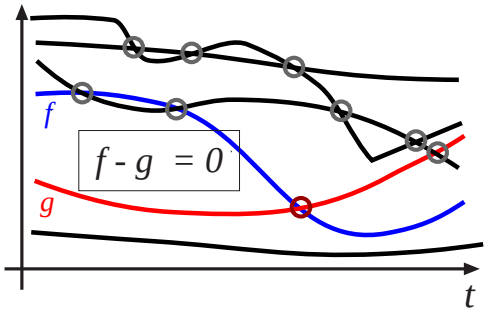


Abbildung 3.3: Ereignisse der Aufrechterhaltung einer kinetischen sortierten Liste, bzw. Kollisionen zwischen sich im Eindimensionalen bewegenden Objekten.

Kinetische Datenstrukturen bestimmen Veränderungen nur in chronologischer Reihenfolge, rückblickende Betrachtungen sind hingegen problematisch. Aus diesem Grund sollten für die Verarbeitung von Bewegungen auch andere Hilfsmittel in Betracht gezogen werden, etwa Verfahren zur Segmentschnittpunktbestimmung, die nach dem *divide & conquer*-Prinzip [PS85] arbeiten. Ein solcher Algorithmus ist der Algorithmus von Balaban [Bal95], der im Unterschied zum Verfahren von Bentley und Ottmann von einer zeitlich globalen Sicht auf die Segmente ausgeht. Der Ablauf des Verfahrens lässt sich im Wesentlichen auf die im rechten Teil der Abbildung 3.2 dargestellte Situation zurückführen. Der Algorithmus betrachtet alle Segmente, die zum Zeitpunkt $t = t_a$ definiert sind und y-geordnet vorliegen müssen. Als Ausgabe

erzeugt der Algorithmus nun eine y-geordnete Folge aller Segmente, die zum Zeitpunkt $t = t_b$ definiert sind, wobei gleichzeitig alle Schnittpunkte bestimmt werden, die im Zeitintervall $[t_a, t_b]$ auftreten. Den Algorithmen von Balaban sowie von Bentley und Ottmann ist das Prinzip gemein, immer nur die Paare von „benachbarten“ Segmenten auf Schnittpunkte zu testen.

3.3 Einsatz und Modellierung von Primitiven für die Verarbeitung vielfältiger Bewegungsrepräsentationen

In diesem Kapitel wird eine Erweiterung des in Kapitel 2 beschriebenen Rahmenwerks beschrieben. Diese ermöglicht die Integration von effizienten Algorithmen für Aufgabenstellungen, wie den in den vorangegangenen Abschnitten 3.1 und 3.2 angeführten, ohne dass eine individuelle Anpassung dieser für einzelne im Rahmenwerk integrierte Bewegungsrepräsentationstypen notwendig wird. Die Beschreibung dieser Erweiterung wird in Abschnitt 3.5 fortgesetzt, in dem für derartige Algorithmen Eingaben betrachtet werden, zu deren Elementen zusätzliche Informationen in Form von Bewegungsrestriktionen vorliegen.

3.3.1 Isolierung der Primitivoperationen in Bewegungen verarbeitenden Algorithmen

Für die Modellierung von Algorithmen zur Bewegungsverarbeitung bietet sich ein klassischer „black box“-basierter Ansatz an:[7] Ein in das Rahmenwerk integrierter Algorithmus zur Bewegungsverarbeitung interagiert mit Elementen der Eingabe, d. h. mit konkreten Bewegungsrepräsentationen, ausschließlich durch gekapselte Aufrufe von Primitivoperationen. Für eine Vielzahl solcher Algorithmen können die Primitivoperationen, auf die sie sich derart abstützen, mit folgender Signatur formuliert werden, wie bereits für kinetische Datenstrukturen von Basch [Bas99] ausgeführt und in Abschnitt 3.2 skizziert: Die Eingabe einer Primitivoperation besteht aus einer festen Anzahl von Bewegungsrepräsentationen und liefert einen einzelnen Zeitpunkt zurück. Für viele Primitivoperationen korrespondiert dieser Zeitpunkt zu einer Veränderung einer kombinatorischen Eigenschaft der Eingabe; für Primitive, die in kinetischen Datenstrukturen eingesetzt werden, fällt dieser Zeitpunkt mit der Invalidierung der aktuellen Beschreibung der kombinatorischen Eigenschaft, die durch die jeweilige kinetische Datenstruktur aufrecht zu erhalten ist, zusammen.

Wesentlich für die angestrebte Integration solcher Algorithmen und ihre Anwendbarkeit für *alle* integrierten Bewegungsrepräsentationstypen ist, dass die Entscheidung, welche konkrete — eventuell von den Repräsentationstypen der übergebenen

[7] Ein *Black Box*-System bezeichnet ein System, zu dem Charakteristika von Ein- und Ausgabe bekannt sind, dessen interne Funktionsweise aber unbekannt bleibt bzw. nicht publiziert ist. Das *Black Box*-Prinzip hat im Software Engineering vor allem durch die Verwendung von Schnittstellendefinitionen für funktionale Aufgaben Einzug gehalten, besitzt aber einen weit darüber hinaus gehenden Anwendungsbereich, den erstmals Ashby [Ash56, Kapitel 6] zu umreißen versuchte, vergleiche für einen weiteren Versuch auch `http://en.wikipedia.org/wiki/Black_box_%28systems%29`.

Objekte abhängige — Realisierung einer solchen Primitivoperation ausgeführt wird, nicht vom Algorithmus festgelegt ist. Stattdessen befragt ein solcher im Rahmenwerk integrierter Algorithmus vor Aufruf eines Primitivs, auf das er sich abstützt, eine Instanz der Klasse `Decider`, die daraufhin eine geeignete Realisierung dieses Primitivs identifiziert und bereit stellt.[8] Diese Entscheidung, bzw. die Eignung einzelner verfügbarer Realisierungen, wird vom `Decider`-Objekt bezüglich der Operanden des Aufrufs, d. h. unter Berücksichtigung der konkreten Bewegungsrepräsentationen, evaluiert. Der Ablauf eines Primitivaufrufs ist schematisch im Diagramm in Abbildung 3.4 dargestellt.

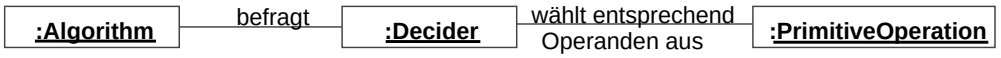


Abbildung 3.4: Primitivauswahl durch ein `Decider`-Objekt zum Zwecke der generalisierten Anwendung von Bewegung verarbeitenden Algorithmen.

Beispiel Kollisionserkennung Für die Aufgabenstellung der eindimensionalen Kollisionserkennung hat das Primitiv, auf das sich entsprechende Algorithmen abstützen, die Kollisionen (bzw. Schnittpunkte) zwischen nur *zwei* Datenobjekten zu identifizieren. Für die der Kollisionserkennung ähnlichen Aufgabenstellungen der Aufrechterhaltung einer kinetischen sortierten Liste und des Segmentschnittproblems ist dies bereits formuliert und ausgenutzt worden, vergleiche Basch [Bas99] bzw. Boissonnat und Snoeyink [BS99] sowie Boissonnat und Vigneron [BV02].

Bemerkung 3.3.1 Lösungen des Segmentschnittproblems (und damit der eindimensionalen Kollisionserkennung) sind scheinbar bereits in praxistauglichen Algorithmenpaketen wie der `Arrangement_2`-Klasse in CGAL[9] und für verschiedene Repräsentationstypen realisiert: Diese `C++`-Klasse — und mit ihr die Bestimmungen der Schnittpunkte innerhalb einer Menge von Kurven — ist generisch implementiert: Ein Template-Parameter gibt den Typ der t-monotonen Kurven an, die als Eingabe akzeptiert werden.

Die entscheidende Einschränkung einer solchen Implementierung ist, dass nur *homogene* Eingabemengen, d. h. nur Kurven gleichen Typs, beispielsweise stückweise lineare Kurven, bearbeitet werden können. Die Auflösung dieser Einschränkung, d. h. die Integration von Algorithmen (wie für die Kollisionserkennung) auch für *heterogen* typisierte Eingaben, ist eine zentrale Zielsetzung in dieser Arbeit.

[8] Der Auswahlprozess, den ein `Decider`-Objekt zu leisten hat, kann in zwei Funktionalitäten eingeteilt werden: Zum einen hat dieser Prozess zumindest eine, aber möglichst alle Realisierungen zu identifizieren, die sich für einen konkreten Primitivaufruf und die ihm übergebenen Parameter eignen. Zum anderen sollte (im Sinne der Anfrageoptimierung) die geeignetste Realisierung identifiziert werden.

[9] Einen Überblick über die CGAL-Bibliothek für robuste geometrische Algorithmen liefern Fabri *et al.* [FGK+00].

Kapselung von Primitivaufrufen Nachfolgend ist durch Pseudo-Code ein Beispielaufruf oben beschriebenen Primitivs innerhalb eines Algorithmus zur Kollisionserkennung für zwei Bewegungsrepräsentationen f und g skizziert.

```
if ( f.collidesWith(g, begin, end) ) {  /* Do something. */  }
```

Eine Schwachpunkt dieser Modellierung wäre, dass jeder Typ von Bewegungsrepräsentationen, d. h. jede Konkretisierung der abstrakten Klasse `Function`, eine eigene Realisierung des verwendeten Primitivs in Form einer — zudem polymorphen— Methode `collidesWith()` bereitstellen müsste.[10] Geeigneter scheint es, die Kenntnis über die im Rahmenwerk implementierten Bewegungsrepräsentationstypen und die jeweils geeigneten Primitivrealisierungen von den Klassendefinitionen der einzelnen Repräsentationstypen zu entkoppeln. Dies wird durch Kapselung der Auswahl einer geeigneten Primitivrealisierung — entsprechend allen Argumenten des Primitivaufrufs — durch ein `Decider`-Objekt erreicht. Ein entsprechend modifizierter Primitivaufruf, der die Idee aus Abbildung 3.4 umsetzt, ist nachfolgend angegeben.

```
Function              f = ..., g = ...;
CollisionPrimitive cp = myCollisionDecider.poll(f, g);

if ( cp.find(f,g, begin, end) != null) { /* Do something. */ }
```

3.3.2 Modellierung arithmetischer Verknüpfungen von Bewegungsrepräsentationen

Für viele Algorithmen, die Bewegungen verarbeiten, können die Primitive, die es zu kapseln gilt, wie erwähnt als Nullstellenbestimmung zu einer Funktion, die sich als arithmetische Verknüpfung von Bewegungsrepräsentationen ergibt, formuliert werden. Als Verknüpfungen sind bislang die Differenz in obigem Beispiel sowie in den bereits in Abschnitt 3.2 genannten Problemstellungen die Distanz, eine Determinantenberechnung sowie eine Komposition aus Differenz und Distanz betrachtet worden. Im Folgenden sei als arithmetische Komposition oder *arithmetische Verknüpfung* eine Verarbeitung von Bewegungsrepräsentationen durch arithmetische Operationen bezeichnet;[11] für solche Operationen seien insbesondere die Addition und die Multiplikation zugelassen.[12] Für eine arithmetische Verknüpfung ist auch die Verwendung von Konstanten — und somit auch die skalare Addition und Multiplikation — zulässig.

[10] Konkreter müsste die bereitgestellte Methode für jeden Repräsentationstyp jeweils so implementiert werden, dass sie für Argumente jeden Typs von Bewegungsrepräsentation operabel wäre.

[11] In einer Verknüpfung sind insbesondere auch konstante Terme als Summanden oder Multiplikatoren zulässig.

[12] Die Operation der Distanz lässt sich bedingt insofern durch Multiplikation und Addition darstellen, als dass für die Bestimmung ihrer Nullstellen die quadrierte Distanz verwendet werden kann; diese wiederum ist durch Verknüpfung der oben angegebenen Operationen darzustellen. Probleme können sich bei der Weiterverarbeitung von Distanzen — anlog zu den Ausführungen in Abschnitt 2.3.1 — ergeben. Hierauf wird nochmals in Abschnitt 3.6.3 eingegangen.

Für die Kollisionsbestimmung für eine Menge im eindimensionalen Raum sich bewegender Objekte ist das relevante Primitiv die Nullstellenbestimmung der Differenzfunktion zweier Bewegungsrepräsentationen: Die Nullstellen von $f - g$ entsprechen genau den Kollisionen der Objekte, deren Bewegungen durch f bzw. g repräsentiert sind. Die im Folgenden beschriebene Modifikation des in Kapitel 2 vorgestellten Rahmenwerkes erlaubt die Integration eines Primitivs, das generisch für verschiedene arithmetische Verknüpfungen (angewendet auf konkrete Bewegungsrepräsentationen) deren Nullstellen bestimmt. Ein Aufruf eines solchen generischen Primitivs durch einen Algorithmus zur Kollisionserkennung ist nachfolgend in Pseudo-Code angegeben:

```
Difference          diff = new Difference(f, g);
ZeroFinderPrimitive zfp  = myZeroFinderDecider.poll(diff);

if (zfp.find(diff, begin, end) != null) { /* Do something. */ }
```

Es wird nicht wie im Pseudo-Code aus Abschnitt 3.3.1 ein spezielles, im eindimensionalen Raum Kollisionen zweier Objekte findendes Primitiv aufgerufen, sondern ein generisches Primitiv, dem die zwei Argument-Bewegungsrepräsentationen, verknüpft durch die arithmetische Differenzfunktion, übergeben werden. Für die Kollisionserkennung im Mehrdimensionalen beispielsweise kann der Aufruf desselben generischen Primitivs mit durch die Distanzfunktion verknüpften Bewegungsrepräsentationen erfolgen.

Die beschriebene generische Modellierung von Primitiven lässt sch in das in Abschnitt 2.4.1 beschriebene Modell des Rahmenwerks integrieren, da die Schnittstelle der Klasse `Function` minimalistisch konzipiert ist: Somit können nicht nur einzelne Bewegungsrepräsentationen, sondern auch arithmetische Verknüpfungen von solchen Repräsentationen als Instanzen der Klasse `Function` modelliert werden; die arithmetischen Verknüpfungen — instanziiert durch ihre Argumente — sind ebenfalls durch Funktionen in der Zeit beschrieben und „übernehmen“ relevante Eigenschaften der Argumente wie Stetigkeit und Auswertbarkeit.[13] Ein gegenüber der Darstellung in Abschnitt 2.4.1 entsprechend erweitertes Klassendiagramm ist in Abbildung 3.5 dargestellt.

Eine konkrete Bewegungsrepräsentation ist nun Instanz der Unterklasse `InterpolationFunction` der Klasse `Function`, während eine c-stellige arithmetische Verknüpfung von solchen Bewegungsrepräsentationen Instanz der Subklasse `composedOfFunction` ist. Eine solche Instanz enthält zum einen Referenzen auf ihre Argumente, d. h. auf die durch sie verknüpften Bewegungsrepräsentationen $f_1, \ldots, f_c$, zum anderen bietet sie eine Methode `getCompound(`$op_1, \ldots, op_c$`)` an, die die entsprechende arithmetische Verknüpfung für nicht zeitvariante Argumente $op_1, \ldots, op_c$ auswertet. Die zu implementierende Methode `evaluate(t)` stützt sich auf die Methode `evaluate(t)` der Argumentrepräsentationen ab und berechnet das Ergebnis

[13] Bei der Implementierung einiger arithmetischer Verknüpfungen (im allgemeinen Sinne des Wortes) und ihrer funktionalen Darstellung ist allerdings insofern Vorsicht geboten, als dass etwa die Verknüpfung der Division durch eine nullstellenbehaftete Funktion zu einer unstetigen Verknüpfung führt.

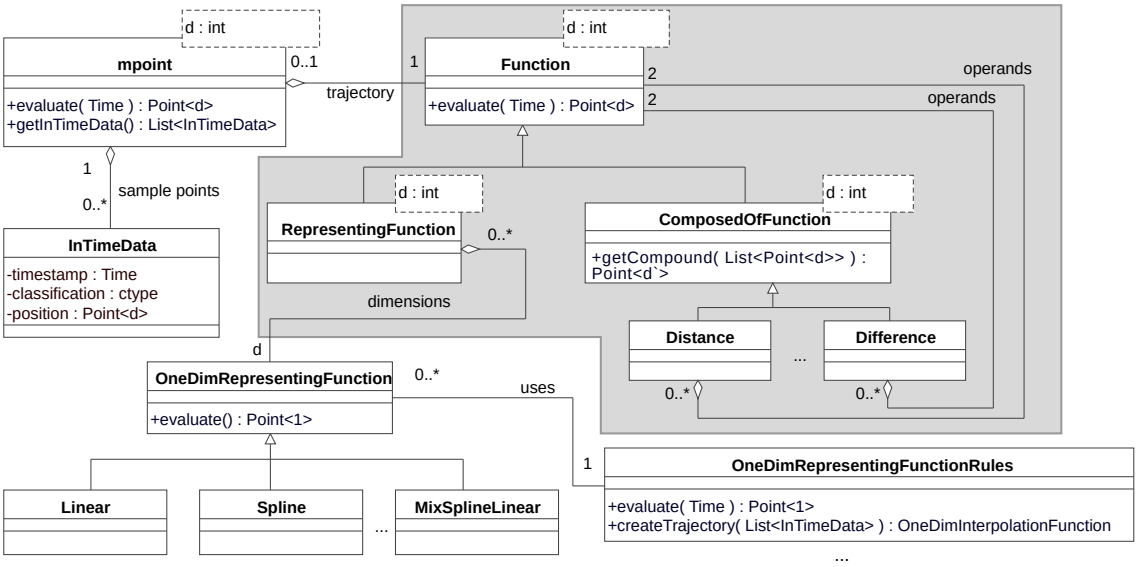


Abbildung 3.5: Klassendiagramm des erweiterten Rahmenwerks aus Kapitel 2; die Erweiterungen sind hervorgehoben.

von deren Verknüpfung durch Aufruf von `getCompound(evaluate(`f_1`), ..., evaluate(`f_c`))`.

Eine Konkretisierung der in diesem Kapitel vorgestellten Modellierung wird in Abschnitt 3.6 geliefert, nachdem zunächst Primitivrealisierungen, die Bewegungsrestriktionen der beteiligten Objekte ausnutzen, ausführlich vorgestellt wurden. Die Modellierung und Realisierung der Auswahl von Primitivrealisierungen durch `Decider`-Objekte wird in Abschnitt 3.6.2 genauer beschrieben. In den beiden nachfolgenden Abschnitten 3.4 und 3.5 wird zunächst die Einsetzbarkeit von (semi)-algebraischen und nmerischen sowie von Bewegungsrestriktionen nutzenden, numerischen Verfahren zur Nullstellenbestimmung für die Realisierung von Primitiven diskutiert.

3.4 Entwicklung und Einsatz von klassischen Primitivrealisierungen

Probleme wie das Auffinden von Kollisionen sind, wie soeben skizziert, auf das Problem der Bestimmung der Nullstellen einer arithmetischen Verknüpfung von Bewegungsrepräsentationen, d. h. von stetigen Funktionen, reduzierbar. Zur Nullstellenbestimmung existieren zahlreiche Techniken, die sich in numerische, algebraische und algebraisch-numerische Verfahren untergliedern lassen. Im Folgenden werden kurz algebraische und algebraisch-numerische und anschließend rein numerische Verfahren vorgestellt, da letztere unabhängig von den konkreten mathematischen Details der betrachteten Funktionen, d. h. auch unabhängig von Bewegungsrepräsentationstypen, operieren können.

3.4.1 Primitivrealisierung durch algebraische Nullstellenbestimmung

In vielen Bereichen der algorithmischen Verarbeitung mobiler Objekte, etwa im Bereich der kinetische Datenstrukturen, wird von polynomialen Bewegungsrepräsentationen ausgegangen [Bas99, Gui98]. Um die Nullstellen der Differenz zweier Polynome von beispielsweise kubischem Grad zu finden, kann ausgenutzt werden, dass der Polynomring und insbesondere die additive Gruppe der Polynome drittes Grades gegenüber der Differenzoperation, wie auch etwa gegenüber der Durchschnittsbildung oder der Translation, abgeschlossen sind. Folgerichtig existieren für Polynome dritten Grades zahlreiche algebraische Verfahren zur Nullstellenbestimmung, vergleiche Bronstein *et al.* [BSM99]. Für die Wahl von Polynomen zur Bewegungsrepräsentation scheint zu sprechen, dass derartige algebraische oder auch algebraisch-numerische Methoden zur Nullstellenbestimmung nicht nur existieren, sondern teilweise bereits in Softwarepaketen implementiert sind, vergleiche den Überblick von Schirra [Sch00]. Insbesondere ist für die Bibliothek geometrischer Algorithmen namens CGAL [FGK+00] eine Erweiterung vorgestellt worden, die eine Integration kinetischer Datenstrukturen sowie zahlreicher algebraischer Nullstellen berechnender Methoden zur Realisierung von Primitiven bereitstellt [GKR04].

Der Aufwand für algebraische Nullstellenbestimmungen wächst allerdings stark mit dem Polynomgrad (oder allgemeiner: mit der algebraischen Komplexität) der zu untersuchenden Funktionen [GK99]. Ein weiterer Nachteil rein algebraischer Nullstellenbestimmungen ist, dass Nullstellen auch außerhalb des (in einem konkreten Aufruf) zu untersuchenden Eingabebereiches der zu betrachtenden Funktion gesucht bzw. gefunden werden: Insbesondere werden Nullstellen im Komplexen und außerhalb des zeitlichen Definitionsbereiches der Bewegung, die durch die zu betrachtende (algebraisch auf der gesamten Zeitachse definierten) Funktion repräsentiert ist, bestimmt [GK99]. Dieses Problem ist für stückweise polynomiale, beispielsweise Spline-basierte, Bewegungsrepräsentationen noch signifikanter: Für jedes einzelne Segment der Bewegungsrepräsentation, das die Bewegung in einem für den Aufruf relevanten zeitlichen Bereich darstellt, ist eine Nullstellenbestimmung durchzuführen. Dies ist bei einer Darstellung durch viele Segmente, etwa durch eine Interpolation vieler zeitlich nahe beieinander liegender diskreter Positionsdaten, zeitaufwändig; die Wahrscheinlichkeit ist jeweils gering, dass eine gefundene Nullstelle tatsächlich im Zeitintervall, für das das Segment die Bewegung repräsentiert, liegt und somit hinsichtlich der gestellten Anfrage relevant ist.

Problematisch ist die algebraische Nullstellenbestimmung zudem bei Anwendung auf Verknüpfungen wie der Distanz: Diese beinhaltet eine Wurzeloperation, die allerdings für die Nullstellenbestimmung ignoriert werden kann, da die Nullstellen der quadratischen mit der der eigentlichen Distanzfunktion übereinstimmen. Eine Radizierung wird allerdings notwendig, wenn die Nullstellen einer Weiterverarbeitung von Distanzen bestimmt werden sollen — etwa zur Bestimmung des Paares mit minimalem Abstand, vergleiche die Abschnitte 3.2 und auch 2.3.1. Aber schon die Quadrierung einzelner Terme der Argumente führt dazu, dass sich für übergebene polynomiale Funktionen der Polynomgrad der quadrierten Distanzfunktion verdoppelt und der Zeit- und Implementierungsaufwand für die algebraische Null-

stellenbestimmung sich erhöht.

Der grundlegendste Nachteil der algebraischen Nullstellenbestimmung ist jedoch — im Hinblick auf die Zielsetzung dieses Kapitels — ihre sehr beschränkte Eignung für heterogene Eingaben: Eine algebraische Nullstellenbestimmung fordert einen festen, einheitlichen Typ für *alle* Operanden, d. h. einen einheitlichen (algebraischen) Typ aller einer Verknüpfung übergebenen Bewegungsrepräsentationen. Zudem wären im Rahmenwerk algebraische Nullstellenbestimmungen nicht nur für einen jeden dieser Typen zu realisieren, sondern vielmehr für eine jede Kombination bzw. Verknüpfung solcher Typen durch jede implementierte arithmetische Verknüpfung: Sind beispielsweise die Kollisionen (bzw. die Nullstellen der Differenz) einer durch ein Meyer-Wavelet [Chu92] und einer durch ν-Splines repräsentierten Bewegung aufzufinden, so wäre eine algebraische Nullstellenbestimmung für einen neuen Repräsentationstyp zu realisieren, da die Differenz dieser beiden Repräsentationen nicht vom Repräsentationstyp eines der beiden Argumente ist. Für die Berechnung der Distanz mehrdimensionaler Repräsentationen dieser Typen wäre eine noch aufwändigere algebraische Nullstellenbestimmung zu realisieren.[14]

Auf Grund der erstgenannten Nachteile rein algebraischer Verfahren zur Nullstellenbestimmung werden häufig hybride Verfahren eingesetzt: Solche semi-algebraischen Verfahren nutzen ebenfalls die algebraische Repräsentation einer Funktion, grenzen aber auf iterative Weise die zu findenden Nullstellen ein. Für diese Eingrenzungen können algebraische Aussagen wie das Theorem von Sturm und die Regel von Descartes verwendet werden, siehe Guibas und Karavelas [GK99] oder Sondern [Son05]. Ein weit verbreitetes Werkzeug zur Eingrenzung von Nullstellen ist die Intervallarithmetik, zu der Kearfott [Kea96] eine Einführung anbietet. Das Werk von Hammer *et al.* [HHKR95] beschreibt eine `C++`-Software-Bibliothek, die Methoden der Intervallarithmetik bereitstellt. Die Realisierung von Primitiven zur Nullstellenbestimmung zeitvarianter Funktionen wird bereits von Guibas und Karavelas im Kontext der Implementierung kinetischer Datenstrukturen erörtert [GK99].

In gleicher Weise wie die rein algebraischen sind auch die semi-algebraischen Methoden als Primitivrealisierungen für die Verarbeitung von heterogen repräsentierten Bewegungen wenig geeignet, da diese ebenfalls die mathematische Gestalt der zu untersuchenden Funktionen explizit betrachten, d. h. zur Nullstellenbestimmung auf die Kenntnis dieser angewiesen sind. Eine praktische Evaluation algebraischer und insbesondere semi-algebraischer Verfahren für den Einsatz zur Aufrechterhaltung kinetischer Datenstrukturen findet sich bei Guibas *et al.* [GKR04]. Eine genauere Vorstellung einiger solcher Verfahren und eine Erörterung ihrer Einsetzbarkeit innerhalb des in dieser Arbeit vorgestellten Rahmenwerks findet sich in der Diplomarbeit von Sondern [Son05, Kapitel 2].

[14] Eine offensichtliche Alternative zur Implementierung einer Vielzahl algebraischer Nullstellenbestimmungen wäre, ein einzelnes Verfahren für eine möglichst umfassende Algebra bzw. Menge von Funktionen zu definieren. Die Effizienz algebraischer Nullstellenbestimmungen resultiert jedoch gerade aus der möglichst genauen Kenntnis und Berücksichtigung der algebraischen Gestalt der konkret zu untersuchenden Funktion, vergleiche beispielsweise Führer [Füh01], so dass dieses einzelne Verfahren entweder nur ineffizient (oder sehr implementationsaufwändig und mit vielen Fallunterscheidungen) zu realisieren wäre. Die algebraische Lösung von Gleichungen verschiedener Algebren wird in den Ausführungen zu Constraint-Logik und -datenbanken in Abschnitt 3.6.3 nochmals behandelt.

3.4.2 Primitivrealisierung durch klassische numerische Nullstellenbestimmung

Klassische numerische Verfahren wie Newtons Methode, die Intervallhalbierung oder die Sekantenmethode beruhen auf iterativer Auswertung der Funktion, deren Nullstellen zu finden sind, vergleiche beispielsweise Suli und Mayers [SM03], Stoer [Sto99] oder Schwarz [Sch97]. Daher sind diese numerischen Verfahren auf fast beliebige arithmetische Verknüpfungen — auch von heterogen typisierten Bewegungsrepräsentationen — anwendbar. Insbesondere können sie problemlos innerhalb des vorgestellten Rahmenwerkes verwendet werden, da sie sich nur auf die minimalistische Schnittstelle für Trajektorien in Form der Methode `evaluate()` abstützen.[15]

Für klassische und viel verwendete numerische Verfahren zur Nullstellenbestimmung (wie den oben genannten) ist ihre Korrektheit ohne spezielle Kenntnisse über die zu untersuchende, als stetig vorausgesetzte Funktion nicht gewährleistet:[16] Die Verfahren können im Einzelfall Nullstellen übersehen oder auch divergieren, d. h. nicht terminieren. Diese Unzuverlässigkeit ist inakzeptabel für viele Anwendungsszenarien, in denen Verknüpfungen von Bewegungsrepräsentationen zu untersuchen sind, wie beispielsweise in der Flugverkehrsüberwachung. Einen Überblick über klassische numerische Verfahren zur Nullstellenbestimmung und eine Diskussion ihrer Einsetzbarkeit für die Verknüpfung von Bewegungen (insbesondere im vorgestellten Rahmenwerk) findet sich in der Diplomarbeit von Puke [Puk03, Kapitel 2.1]. Im folgenden Abschnitt werden numerische, Bewegungsrestriktionen verwendende Verfahren zur Nullstellenbestimmung vorgestellt, die das Auffinden aller Nullstellen garantieren.

3.5 Realisierung von Primitiven unter Ausnutzung von Bewegungsrestriktionen

Algebraische Verfahren zur Nullstellenbestimmung erfordern präzise Kenntnisse über die mathematische Repräsentation der zu untersuchenden Funktionen. Solche Kenntnisse können bzw. sollten im vorgestellten Rahmenwerk nicht uneingeschränkt vorausgesetzt werden, da für dieses die einfache Erweiterbarkeit um Repräsentationstypen ein wesentliches Modellierungsziel ist. Klassische numerische Techniken zur Nullstellensuche hingegen garantieren keine Korrektheit, d. h. Suchläufe übersehen eventuell Nullstellen oder terminieren nicht.

[15] Einige der klassischen numerischen Verfahren profitieren davon, wenn auch Auswertungen von Ableitungen der zu untersuchenden Funktion möglich sind. Die Verfügbarkeit dieser Auswertungsfunktionalität kann an Hand der Typen der übergebenen Argumenttrajektorien der zu untersuchenden Funktion überprüft werden, vergleiche die Modellierungsbeschreibung in Abschnitt 3.6.2. Alternativ können diese Verfahren auch auf nicht explizit ableitbaren Bewegungsrepräsentationstypen operieren, indem numerisch Näherungen für Ableitungswerte durch Bildung der Differnzenquotienten von Auswertungen der nicht abgeleiteten Funktion berechnet werden. Eine Korrektheit einer solchen Nullstellensuche (d. h. das Auffinden aller Nullstellen) kann allerdings im allgemeinen Fall nicht garantiert werden, vergleiche [Sch97].

[16] Überblicke über numerische Verfahren zur Nullstellenbestimmung sowie deren Analyse hinsichtlich Effizienz und Korrektheit finden sich jeweils bei Schwarz [Sch97], Stoer [Sto99] sowie bei Suli und Mayers [SM03].

Es sind Korrektheit garantierende numerische Verfahren konstruierbar, sofern Charakteristika der zu betrachtenden Bewegungen bekannt sind. In diesem Kapitel werden solche Verfahren zur Nullstellenbestimmung für die Realisierung von Primitiven beschrieben. Als Bewegungscharakteristika werden Restriktionen der beteiligten Bewegungen wie Beschränkungen von Geschwindigkeit oder Beschleunigung ausgenutzt.

Kenntnis von Restriktionen in praktischen Anwendungen Die Annahme, dass Bewegungscharakteristika der zu betrachtenden mobilen Objekte bekannt sind, ist durchaus praxisnah; die meisten Bewegungen verarbeitenden Anwendungen haben Kenntnis von zumindest ungefähren Beschränkungen des Bewegungsverhaltens der zu verwaltenden Objekte. Beispielhaft sei hier erneut auf die BADA-Datenbank hingewiesen, die vom europäischem Institut für Flugsicherung EUROCONTROL vorgehalten wird und in der verschiedenste Bewegungscharakteristika von über 250 Flugzeugtypen vermerkt sind [Eur99, NPI+05].

Zusätzlich ist anzumerken, dass die auszunutzenden Beschränkungen durchaus ungenau sein dürfen: Solange sie konservative Abschätzungen darstellen, die also auf jeden Fall von den zu beobachtenden Objekten eingehalten werden, bleiben die Nullstellenbestimmungen korrekt; die Präzision der verwendeten Restriktionen wirkt sich nur (positiv) auf die Effizienz der Nullstellenbestimmung bzw. des Primitivaufrufs aus.

Grundkonzept Sind für eine Bewegung Restriktionen, wie eine obere Schranke für die absolute Geschwindigkeit oder Beschleunigung, bekannt, so implizieren diese Restriktionen Beschränkungen von Ableitungen der Bewegungsfunktionen. Für arithmetische Verknüpfungen einer oder mehrerer Bewegungsrepräsentationen, etwa der Differenz, ergeben sich hieraus Beschränkungen für die Ableitungen des Ergebnisses der Verknüpfung.[17] Die hier beschriebenen Verfahren nähern sich iterativ, ausgehend jeweils von einem Startzeitpunkt t_i, an die im Zeitverlauf nächste Nullstelle der zu untersuchenden Funktion an. Dabei kann auf Grund der zu den beteiligten Bewegungen bekannten Restriktionen der Startzeitpunkt t_{i+1} der jeweils nächsten Iteration so gewählt werden, dass die Existenz einer Nullstelle im Intervall $[t_i, t_{i+1}]$ ausgeschlossen werden kann. Genauere Beschreibungen einiger der im Folgenden vorgestellten Verfahren finden sich bereits in den Diplomarbeiten von Puke [Puk03, Kapitel 3] und Sondern [Son05]. In letzterer werden auch Nachweise der Korrektheit sowie Effizienzanalysen einiger der hier vorgestellten Verfahren ausgeführt. Die Veranschaulichung der einzelnen Verfahren erfolgt hier erneut am Beispiel der Kollisionserkennung für sich im Eindimensionalen bewegende Objekte.

[17] An dieser Stelle sei bemerkt, dass die Restriktionen, die für eine Bewegung angenommen werden, von der Repräsentation dieser Bewegung einzuhalten sind, damit die Nullstellen bestimmenden Verfahren Korrektheit garantieren können. In Abschnitt 2.2.3 wurden unter anderem Repräsentationstypen skizziert, für die Restriktionen als (verbindliche) Parameter bei der Erstellung einer Repräsentation übergeben werden können Für Repräsentationen anderer Typen wäre zu prüfen, ob diese Restriktionen eingehalten werden; gegebenenfalls wären entweder die für die Bewegung postulierten Restriktionen oder die Repräsentation selbst entsprechend zu modifizieren.

3.5.1 Bewegungen mit beschränkter Geschwindigkeit

Die Annahme, dass die durch f und g repräsentierten Bewegungen beschränkte absolute Maximalgeschwindigkeiten aufweisen, impliziert jeweils eine Beschränkung der ersten Ableitung f' bzw. g' durch je eine Konstante G_f bzw. G_g. Im linken Teil von Abbildung 3.6 ist eine Iteration eines Verfahrens skizziert, das solche Beschränkungen ausnutzt: Sind für Objekte, deren Bewegungen durch f und g repräsentiert sind, ihre Positionen zum Zeitpunkt t_i bekannt, so befinden sich die möglichen zukünftigen Aufenthaltsorte der zu f und g assoziierten Objekte in den schattiert dargestellten (raum-zeitlichen) Korridoren. Somit fällt der frühestmögliche Zeitpunkt einer Kollision der Objekte mit dem Zeitpunkt des (zeitlich frühesten) Schnittes der Korridore zusammen. Anschaulich tritt eine Kollision zu diesem frühestmöglichen Zeitpunkt t_{i+1} genau dann ein, wenn die Objekte sich mit maximaler, konstanter Geschwindigkeit aufeinander zu (d. h. auf den Berandungen ihrer Korridore) bewegen. Der entsprechende Zeitpunkt t_{i+1} ergibt sich daher als Lösung folgender linearer Gleichung (dargestellt o.B.d.A. für den Fall $f(t_i) > g(t_i)$):

$$f(t_i) - G_f \cdot (t_{i+1} - t_i) = g(t_i) + G_g \cdot (t_{i+1} - t_i) \tag{3.1}$$

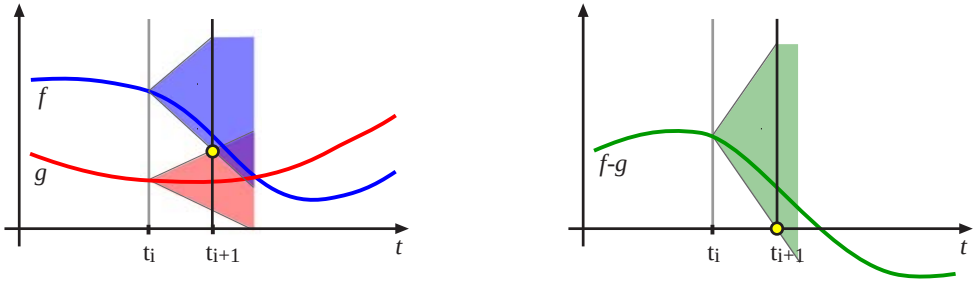


Abbildung 3.6: Iterative Kollisionssuche für geschwindigkeitsbeschränkte Objekte.

Die Auflösung dieser Gleichung nach t_{i+1} liefert folgende Berechnungsvorschrift:

$$t_{i+1} = t_i + \frac{(f-g)(t_i)}{G_f + G_g} \tag{3.2}$$

Im rechten Teil von Abbildung 3.6 ist die Aufgabe der Kollisionserkennung als Nullstellensuche interpretiert dargestellt. Für die Verknüpfung $h(f,g) := -(f,g) = f - g$ ergibt sich aus den Ableitungsbeschränkungen für f und g die Beschränkung $|(f-g)'| \leq G_f + G_g := G_h$; der zu betrachtende Korridor ergibt sich aus der Summe der Beschränkungen der Ableitungen von f und g.

Da eine Nullstelle innerhalb des Intervalles $[t_i, t_{i+1}]$ ausgeschlossen werden kann, kann das Verfahren nun mit t_{i+1} als Startpunkt des nächsten Iterationsschrittes fortgesetzt werden. Hierfür sind erneut die tatsächlichen Positionen der zu f und g assoziierten Objekte zu evaluieren — nun für den Zeitpunkt t_{i+1}. Dieses iterative Verfahren sei im Folgenden als L_{one} benannt, da es die Nullstellen für eine lineare Funktion bestimmt.

3.5.2 Bewegungen mit beschränkter Beschleunigung

Das geschilderte iterative Verfahren lässt sich adaptieren, so dass Beschleunigungs- statt Geschwindigkeitsrestriktionen der Objekte ausgenutzt werden. Hierfür müssen die zu betrachtenden Bewegungsrepräsentationen allerdings folgende, zusätzliche Voraussetzungen erfüllen:

- Die erste Ableitung der betrachteten Bewegungsrepräsentationen muss ausgewertet werden können, d. h. zu einem frei wählbaren Zeitpunkt t_i muss neben der Position auch der Geschwindigkeitsvektor der repräsentierten Bewegung bestimmbar sein.
- Die betrachteten Bewegungsrepräsentationen müssen stetig ableitbare Funktionen sein; eine nicht stetige Ableitung einer Bewegungsrepräsentation würde einen Widerspruch zur Beschleunigungsbeschränkung der repräsentierten Bewegung bedeuten.

Innerhalb des Rahmenwerks wird die Voraussetzung der Ableitbarkeit in der Schnittstelle `Derivable` gekapselt, die von Unterklassen der Klasse `Function` implementiert werden kann. Diese Schnittstelle besteht aus einer Methode `evaluateFirstDerivative()` zur Auswertung des Geschwindigkeitsvektors einer Bewegung.[18]

Ein klassisches Verfahren Die jeweils nächste Näherung t_{i+1}, die eine Nullstellenfreiheit in $[t_i, t_{i+1}]$ garantiert, ergibt sich anschaulich aus der Annahme, die Objekte bewegten sich ab dem Zeitpunkt t_i mit ihrer maximalen Beschleunigung aufeinander zu. Mathematisch ergibt sich dieser Zeitpunkt als Lösung der folgenden quadratischen Gleichung (dargestellt o.B.d.A. für den Fall $f(t_i) > g(t_i)$):

$$f(t_i)+f'(t_i)\cdot(t_{i+1}-t_i)-B_f\cdot(t_{i+1}-t_i)^2 = g(t_i)+g'(t_i)\cdot(t_{i+1}-t_i)+B_g\cdot(t_{i+1}-t_i)^2 \quad (3.3)$$

Die resultierende Berechungsvorschrift für t_{i+1} lautet:

$$t_{i+1} = t_i+\frac{1}{B_f+B_g}\left((f-g)'(t_i)+\sqrt{(f-g)'^2(t_i)+2\cdot(B_f+B_g)\cdot(f-g)(t_i)}\right) \quad (3.4)$$

Eine Iteration dieses Verfahrens, das im Folgenden als P_{one} benannt sei, da es die Betrachtung einer Parabel erforderlich macht, ist in Abbildung 3.7 skizziert.

Eine Variante für retrospektive Anwendungen Eine alternative und potentiell effizientere Variante des Verfahrens P_{one} ist in Abbildung 3.8 skizziert: Hier wird nicht der ab dem Zeitpunkt t_i frühestmögliche Zeitpunkt einer Nullstelle bestimmt, sondern der frühest mögliche Zeitpunkt einer *zweiten* Nullstelle. Dieser ergibt sich anschaulich aus der Annäherung beider Objekte mit maximaler Beschleunigung und einem anschließenden entgegengesetzten Manöver, so dass die Objekte sich gerade

[18] Eine alternative Kapselung der Ableitungsfunktionalität wird in Abschnitt 3.6 beschrieben.

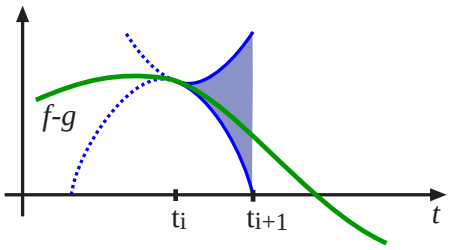


Abbildung 3.7: Iterative, online-fähige Kollisionssuche für beschleunigungsbeschränkte Bewegungen.

berühren, ihre Flugbahnen sich aber nicht kreuzen. Mathematisch ergibt sich dieser Zeitpunkt t_{i+1} als Extremalstelle einer weiteren Parabel, die mit der bereits beschriebenen einen Berührpunkt gemeinsam hat, in dem die zweite die erste Parabel mit übereinstimmender erster Ableitung fortsetzt. Die resultierende angenommene Bewegung beider Objekte ist während der beiden aufeinanderfolgenden Manöver mit stetiger Richtung und Geschwindigkeit abgebildet. Die zweite Parabel wird eindeutig dadurch bestimmt, dass ihr Extremum mit ihrer (zweifachen) Nullstelle zusammenfällt.

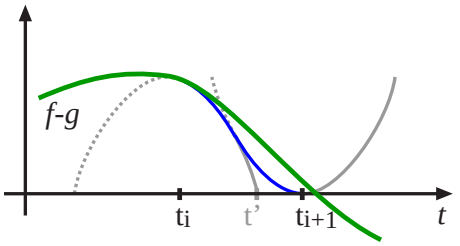


Abbildung 3.8: Eine alternative Technik zur Kollisionssuche für beschleunigungsbeschränkte Bewegungen.

Ergibt die anschließende Evaluation von f und g zum Zeitpunkt t_{i+1}, dass sich das Vorzeichen von $f - g$ nicht geändert hat, so besitzt diese Funktion in $[t_i, t_{i+1}]$ keine Nullstelle und die iterative Suche wird mit t_{i+1} fortgesetzt. Im alternativen, in Abbildung 3.9 dargestellten Falle liegt genau eine Nullstelle in $[t_i, t_{i+1}]$ (genauer: in $[t_{left}, t_{i+1}]$).[19]

Ein weiterer Sonderfall kann von den beiden bereits beschriebenen Fällen unterschieden werden. Ist eine Kollision auch durch ein *sofortiges* Gegenmanöver beider Objekte mit maximaler Beschleunigung nicht zu vermeiden, so befindet sich die unvermeidliche Nullstelle zwischen den Nullstellen der zum sofortigen Gegenmanöver korrespondierenden Parabel und t_{left}, der Nullstelle der entgegengesetzt gerichteten Version dieser Parabel.[20] Die Suche nach weiteren Nullstellen kann an einem Zeit-

[19] Da diese Nullstelle die einzige im Intervall $[t_i, t_{i+1}]$ ist, besteht zu deren näheren Eingrenzung die Option, klassische numerische Verfahren wie beispielsweise die Intervallhalbierung [Sto99] einzusetzen, deren Korrektheit garantiert werden kann, wenn die Anzahl der zu findenden Nullstellen bekannt ist.

[20] Auch für die Eingrenzung dieser Nullstelle können klassische numerische Verfahren wie bei-

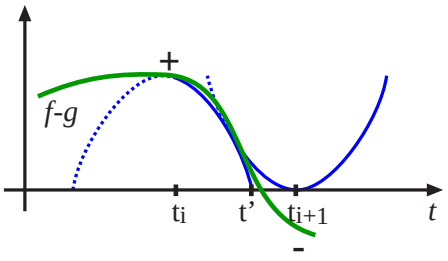


Abbildung 3.9: Nullstelleneingrenzung der alternativen Technik.

punkt t_{i+1} fortgesetzt werden, ohne eine solche zu übersehen. Der spätest mögliche Zeitpunkt hierfür ergibt sich als zweite Nullstelle der zum Ausweichmanöver korrespondierenden Parabel, da vor diesem Zeitpunkt keine weitere Kollision der Objekte auftreten kann.

Das beschriebene Verfahren, das anschaulich auf der Betrachtung zweier Parabeln basiert, sei fortan als P_{two} benannt. Die Berechnungsvorschrift für den Zeitpunkt t_{i+1} ergibt sich entsprechend obiger Fallunterscheidung wie folgt (o.B.d.A. für den Fall $f(t_i) > g(t_i)$):

$$t_{i+1} = t_i \; + \tag{3.5}$$
$$\begin{cases} \frac{1}{B_f+B_g}\left((f-g)'(t_i) \quad +\sqrt{2}\cdot\sqrt{(f-g)'^2(t_i)+2\cdot(B_f+B_g)\cdot(f-g)(t_i)}, \right. \\ \qquad\qquad \text{falls } (f-g)'(t_i) \geq \sqrt{2\cdot(B_f+B_g)\cdot(f-g)(t_i)} \\ \frac{1}{B_f+B_g}\left((f-g)'(t_i) \quad -\quad \sqrt{(f-g)'^2(t_i)-2\cdot B_f+B_g\cdot(f-g)(t_i)}, \right. \\ \qquad\qquad \text{sonst (Kollision unvermeidbar)} \end{cases}$$

Einsetzbarkeit von P_{one} und P_{two} Anders als die Verfahren L_{one} und P_{one}, die bereits für Echtzeitanwendungen wie *online*-Kollisionserkennung und -warnung beschrieben wurden, vergleiche Hayward *et al.* [HAFG95], ist P_{two} nur bedingt in solchen Szenarien einsetzbar: Mit P_{two} können zwar Nullstellen bzw. Kollisionen in Echtzeitanwendungen gezählt werden; das nähere zeitliche Eingrenzen *nach* dem Feststellen einer Nullstelle erfordert allerdings das Evaluieren von Positionen der betrachteten Objekte in *nicht* chronologischer Reihenfolge. Dies ist nicht in Echtzeit-, aber in retrospektiven Anwendungen möglich. Somit kann das Verfahren P_{two} zur exakten Bestimmung von Nullstellen nur für die rückblickende Analyse oder für die Planung von Bewegungen eingesetzt werden.

Im Folgenden seien Verfahren wie L_{one} und P_{one}, die sich streng iterativ und einseitig Nullstellen nähern, als *online*-fähig bezeichnet. Verfahren wie P_{two} hingegen, die die Bereiche von Nullstellen beidseitig eingrenzen, seien als *offline*-Verfahren bezeichnet.

spielsweise Intervallhalbierung [Sto99] eingesetzt werden.

3.5.3 Effizienzanalysen zu Primitivrealisierungen

Kostenmodellierung von Nullstellenbestimmungen In der Algorithmenanalyse wird häufig von den Kosten für die Bestimmung von Nullstellen (oder Schnittpunkten von Kurven) insoweit abstrahiert, als dass für diese Operationen nur Einheitskosten veranschlagt werden; dies gilt sowohl für die Analyse von Algorithmen aus den Bereichen der kinetischen Datenstrukturen als auch für geometrische Algorithmen wie für das Segmentschrittproblem, vergleiche beispielsweise Guibas [Gui98] sowie Boissonnat und Snoeyink [BS99].

Da im beschriebenen Rahmenwerk die Eingabesegmente bzw. Bewegungsrepräsentationen im Allgemeinen von komplexer Gestalt und Nullstellenbestimmungen aufwändig sind, scheint eine genauere Kostenmodellierung und -analyse der Primitivoperationen zur Nullstellen- bzw. Schnittpunktbestimmung notwendig. In der Diplomarbeit von Sondern [Son05] findet sich in den Kapiteln 4 und 5 eine Diskussion geeigneter Kostenmodelle zur Effizienzanalyse von Primitivoperationen sowie eine Übersicht über Parameter bei Primitivaufrufen, die sich auf die Laufzeitkosten der Primitivrealisierungen auswirken. Im Folgenden wird zunächst eine Analyse der bis hierhin beschriebenen Primitivrealisierungen, weclche Bewegungsrestriktionen nutzen, an Hand zum einen ihrer Konvergenzgeschwindigkeit zun anderen ihrer Schrittweite (pro Iteration) in zusammengefasster Form wiedergegeben.

Konvergenzgeschwindigkeiten von Primitivrealisierungen

Zur Konvergenz von Fixpunktiterationen Viele numerische Verfahren zur Nullstellenbestimmung und auch die oben geschilderten, Bewegungsrestriktionen nutzenden Verfahren gehören zu den (reellwertigen und eindimensionalen) Fixpunktiterationen, auch Fixpunktverfahren genannt. Diese sind durch eine Iterationsfunktion $\Phi(t)$ (abbildend von $\mathbb{R}$ nach $\mathbb{R}$) beschrieben, die iterativ für Eingaben $t_{i+1} := \Phi(t_i)$ aufgerufen wird.[21] Fixpunktverfahren können bezüglich ihrer Konvergenzordnung, d. h. bezüglich der asymptotischen Geschwindigkeit, mit der sie sich einem Fixpunkt s nähern, verglichen werden, siehe beispielsweise Stoer [Sto99]. Formal ist die Konvergenzordnung eines Fixpunktverfahrens wie folgt definiert:

Definition 3.5.1 Ein Fixpunktverfahren besitzt eine *Konvergenzordnung* von mindestens $p \geq 1$, falls für die von ihm erzeugte Folge von Iterationspunkten $(t_i)_{i \in N}$ für eine positive reelle Zahl K (mit $K < 1$ für den Fall $p = 1$) gilt:

$$\limsup_{i \to \infty} \frac{||t_{i+1} - s||}{||t_i - s||^p} = K$$

Die reelle Zahl K wird als *asymptotische Fehlerkonstante* bezeichnet.

Die Entfernung vom Fixpunkt nimmt bei einem Verfahren mit Konvergenzordnung 1 linear, bei einem Verfahren mit Konvergenzordnung 2 bereits quadratisch mit der Anzahl i der durchgeführten Iterationen ab. Während die Konvergenzordnung

[21] Das Verfahren P_{two} kann als Fixpunktverfahren angesehen werden, leistet aber mehr als ein solches, da es sich nicht nur auf einen Fixpunkt zu bewegt, sondern einen solchen eingrenzt und gleichzeitig eine Näherung für einen möglichen weiteren Fixpunkt bestimmt.

den polynomialen Grad der Konvergenzgeschwindigkeit angibt, bestimmt sich der asymptotische Vorfaktor (bzw. der Proportionalitätsfaktor) der Konvergenz (bzw. eine konservative Abschätzung für einen solchen) durch die asymptotische Fehlerkonstante; für ein Verfahren der Konvergenzordnung p gilt für „späte“ Iterationen i (im schlechtesten Falle): $|s - t_{i+1}| \approx K \cdot |s - t_{i+1}|^p$.

Konvergenzgeschwindigkeiten der Bewegungsrestriktionen nutzenden Verfahren Das Verfahren L_{one} besitzt eine lineare Konvergenz wie auch einfache numerische Methoden zur Nullstellenbestimmung, wie zum Beispiel die Intervallhalbierung und das Verfahren der Regula Falsi, vergleiche Stoer [Sto99]. Die Verfahren P_{one} und LP_{one} konvergieren quadratisch — wie auch das häufig verwendete numerische Newtonverfahren — und somit sogar schneller als etwa die numerische Sekantenmethode (deren Konvergenzordnung etwa 1,618 beträgt), vergleiche erneut Stoer [Sto99]). Den beiden letztgenannten Methoden ist mit P_{one} und LP_{one} gemein, dass für die Berechnung der Iterationspunkte nicht nur die zu untersuchende Funktion, sondern auch ihre erste Ableitung auszuwerten ist. Die Verfahren L_{one}, Regula Falsi und die Intervallhalbierung verwenden keine Ableitungsauswertungen.

Die Konvergenzgeschwindigkeiten für die vorgestellten, Bewegungsrestriktionen nutzenden Verfahren sowie für die zitierten numerischen Verfahren sind in Tabelle 3.1 aufgelistet. Es ist zu beachten, dass die asymptotische Fehlerkonstante bei allen Bewegungsrestriktionen nutzenden Verfahren unter anderem von den Restriktionswerten abhängt, d. h. von den bekannten Schranken für Geschwindigkeit und/oder Beschleunigung der betrachteten Objekte. Auch für effiziente klassische Verfahren zur Nullstellenbestimmung hängt die Fehlerkonstante von dem Verhalten der betrachteten Funktion (genauer: von ihren Ableitungswerten) an der Nullstelle s ab. Für die Bewegungsrestriktionen nutzenden Verfahren impliziert jedoch auch eine grobe (d. h. betragsmäßig große) Schranke für Geschwindigkeit oder Beschleunigung eine größere Fehlerkonstante, also eine langsamere Konvergenz. Diese Verlangsamung ist jeweils linear in der Größe der vorliegenden Schranken.

Es ist zu beachten, dass die Konvergenzgeschwindigkeit der vorgestellten Verfahren ihre Effizienz nur zu Teilen beschreibt, nämlich nur im Falle der Existenz einer Nullstelle s und dann nur in der unmittelbaren Nähe von s.

Schrittweiten

Der globale Zeitaufwand einer Anwendung eines iterativen Verfahrens ergibt sich als Summe über die Kosten der einzelnen Suchiterationen. Ein weiteres Kriterium, um iterative Verfahren zur Nullstellenbestimmung bezüglich ihrer Effizienz zu vergleichen, ist daher ihre Schrittweite. Wie die Konvergenzgeschwindigkeiten sind auch die Schrittweiten der hier vorgestellten Verfahren abhängig von den Restriktionswerten für Geschwindigkeit bzw. Beschleunigung, siehe hierzu die Tabelle 3.2.

In retrospektiven Anwendungen verspricht P_{two} Effizienzvorteile gegenüber P_{one}: Während die Berechnungskosten einer Iteration (d. h. zur Bestimmung eines Zeitpunktes t_{i+1}) für beide Verfahren annähernd gleich groß sind, weist P_{two} eine generell größere Schrittweite auf, wie in Korollar 3.5.2 präzisierend formuliert.[22]

[22]Die Berechnungskosten einer Iteration von P_{one} bzw. P_{two} ergeben sich aus den Gleichungen

Tabelle 3.1: Konvergenz iterativer Verfahren zur Nullstellenbestimmung ([Son05]).

Verfahren	Konvergenzordnung	asymp. Fehlerkonstante
L_{one}	1	$\lvert\frac{h'(s)+G_h}{G_h}\rvert$
P_{one}	2	$\lvert\frac{h''(s)+B_h}{2\cdot(h'(s))}\rvert$
Intervallhalbierung	1	$\frac{1}{2}$
Regula Falsi	1	$\lvert\frac{h''(s)}{2\dot{h}(s)}\cdot(t_0-s)\rvert$, falls h konkav
Sekantenmethode	1,618	$\lvert\frac{h''(s)}{2\cdot h'(s)}\rvert^{\frac{1}{p}}$
Newtonverfahren	2	$\lvert\frac{h''(s)}{2\cdot(h'(s))}\rvert$

Korollar 3.5.2 *Die Schrittweite $\Delta_{P_{\text{two}}}$ des Verfahrens P_{two}ist generell größer als die von P_{one}:*

$$\Delta_{P_{\text{one}}} \;<\; \Delta_{P_{\text{two}}} \;\leq\; (\sqrt{2}+1)\cdot\Delta_{P_{\text{one}}}$$

Für den Fall $(f-g)(t_i)\cdot(f-g)'(t_i)<0$, d. h. wenn die Objekte sich zum Zeitpunkt t_i aufeinander zu bewegen, gilt sogar:

$$\sqrt{2}\cdot\Delta_{P_{\text{one}}} \;<\; \Delta_{P_{\text{two}}} \;\leq\; (\sqrt{2}+1)\cdot\Delta_{P_{\text{one}}}$$

Bemerkung 3.5.3 Der Effizienzvergleich des online-fähigen Verfahrens P_{one} mit dem offline-Verfahren P_{two} legen nahe, dass für retrospektive Anwendungen andere Verfahren als für Echtzeitanwendungen am geeignetsten sind.[23] Während sich die für Echtzeitanwendungen benötigten online-fähigen Verfahren häufig als spezielle Formulierungen von Lösungen aus dem Bereich der gebräuchlichen Optimierungsmathematik ergeben, worauf in den Abschnitten 3.5.4 und 3.6.3 näher eingegangen wird, belegen das Verfahren P_{two} und seine Analyse, dass die effizienzoptimierte Primitivrealisierung für *retrospektive* Anwendungen eine eigenständigere algorithmische Aufgabenstellung darstellt.

3.4 bzw. 3.5. Eine Dominanz dieser Kosten durch die (für P_{one} und P_{two} identischen) Kosten für die Auswertungen der beiden Bewegungsrepräsentationen ist je nach Anwendungskontext unterschiedlich stark, vergleiche auch Kapitel 2. Die weiteren Kosten einer Iteration sind für P_{one} und P_{two} nahezu identisch: Für P_{two} fallen zusätzliche Kosten nur für Fallunterscheidung (durch Größenvergleich zweier vorliegender Werte) sowie für eine skalare Multiplikation (mit dem Wert $\sqrt{2}$) an.

[23] Diese Aussage bezieht sich auf das exakte Bestimmen von Nullstellen (von arithmetischen Verknüpfungen von Bewegungsrepräsentationen). Für das *Zählen* von Nullstellen lassen sich, wie erläutert, auch in Echtzeitanwendungen offline-Verfahren wie P_{two} einsetzen.

Tabelle 3.2: Schrittweiten iterativer Verfahren zur Nullstellenbestimmung.

Verfahren	Schrittweite
L_{one}	$\lvert\frac{G_h}{h(t_i)}\rvert$
P_{one}	$\frac{1}{B_h} \cdot \left(h'(t_i) + \sqrt{f'^2(t_i) + 2 \cdot B_h \cdot f(t_i)} \right)$
P_{two}	$\begin{cases} \frac{1}{B_h}\left(h'(t_i) + \sqrt{2}\sqrt{f'^2(t_i) + 2B_h f(t_i)} \right) & \text{, falls } f'(t_i) \geq -\sqrt{2B_h f(t_i)} \\ \frac{1}{B_h}\left(h'(t_i) - \sqrt{f'^2(t_i) + 2B_h f(t_i)} \right) & \text{, sonst} \end{cases}$

3.5.4 Verallgemeinerungen

Die beschriebenen, Restriktionen nutzenden Techniken zur Primitivrealisierung lassen sich verallgemeinern — hinsichtlich weiterer arithmetischer Verknüpfungen sowie hinsichtlich der Operabilität für weitere (Kombinationen von) Restriktionstypen. In diesem Abschnitt werden die Möglichkeiten zur Generalisierung untersucht, ihre Verwendung skizziert und insbesondere ihre Einschränkungen herausgearbeitet.

Bewegungen mit heterogen typisierten Restriktionen

Die bislang beschriebenen Verfahren zur Nullstellensuche gehen davon aus, dass die auf Kollisionen zu untersuchenden Bewegungen eine Restriktion derselben Größe — entweder der Geschwindigkeit oder der Beschleunigung — besitzen. In diesem Fall lässt sich auch für die auf Nullstellen zu untersuchende Funktion h eine Beschränkung G_h direkt aus Beschränkungen $G_{f_1}, \ldots, G_{f_c}$ berechnen. Die Berechnung einer solchen Beschränkung G_h folgt nicht nur für die Differenz, sondern auch für andere arithmetische Verknüpfungen denselben Regeln, die auch der Intervallarithmetik zu Grunde liegen, vergleiche Kearfott [Kea96], Hammer *et al.* [HHKR95] sowie Guibas und Karavelas [GK99], und insofern ließen sich Restriktionen G_h für arithmetische Verknüpfungen h automatisiert berechnen. Komplexer gestaltet sich die Ausnutzung von Restriktionen der Argumente, wenn diese Restriktionen unterschiedlichen Typs sind.

Beispiel Kollisionserkennung: Sind zwei Bewegungen (repräsentiert durch f und g) auf Kollisionen zu untersuchen, für die keine Restriktionen gleichen Typs, sondern beispielsweise nur Beschränkungen G_f und B_g gelten, sind die zu den Beschränkungen G_f und B_g korrespondierenden Ungleichungen

$$f(t_i) + G_f \cdot (t - t_i) \leq f(t) \leq f(t_i) + G_f \cdot (t - t_i)$$

und

$$g(t_i) + g'(t_i) \cdot (t - t_i) - B_g \cdot (t - t_i)^2 \leq g(t) \leq g(t_i) + g'(t_i) \cdot (t - t_i) + B_g \cdot (t - t_i)^2$$

genauer zu betrachten. Aus diesen ergibt sich ein Zeitpunkt t_{i+1}, für den Kollisionsfreiheit im Intervall $[t_i, t_{i+1}]$ gesichert ist, wie folgt (o.B.d.A. dargestellt für den Fall $f(t_i) > g(t_i)$):

$$t_{i+1} = t_i + \frac{1}{B_g}\left(G_f - g'(t_i) + \sqrt{(G_f - g'(t_i))^2 - 2 \cdot B_g \cdot g(t_i)}\right) \tag{3.6}$$

Die Vorschrift zur Bestimmung des frühestmöglichen Kollisionszeitpunktes t_{i+1} resultiert also auch im Falle heterogen restringierter Bewegungen aus dem Lösen eines Ungleichungssystems; für eine Kombination von Geschwindigkeits- und Beschleunigungsrestriktionen, wie hier dargestellt, besteht dieses zu lösende System aus quadratischen Ungleichungen.

Auch für die Nullstellenbestimmung jeder anderen arithmetischen Verknüpfung h ergibt sich die Herausforderung, ein Ungleichungssystem — und seine Auflösung nach t_{i+1} — zu bestimmen, so dass zwischen t_i und t_{i+1} Nullstellenfreiheit gesichert ist (bzw. allgemeiner: so dass die Anzahl der Nullstellen zwischen t_i und t_{i+1} bekannt ist). Wird zusätzlich gefordert, dass t_{i+1} als der späteste dieser Zeitpunkte zu bestimmen ist, stellt diese Herausforderung eine Optimierungsaufgabe mit Randbedingungen dar. Ein solche Optimierungsaufgabe ist parametrisiert durch die arithmetische Verknüpfung sowie durch die Restriktionsmengen der jeweiligen Operanden; ihre Lösung resultiert in einer Berechnungsvorschrift für t_{i+1} wie in Gleichung 3.6 exemplarisch für die Verknüpfung der Differenz und eine Restriktion der Geschwindigkeit und des einen und einer Restriktion der Beschleunigung des anderen Operanden dargestellt. Wie solche Lösungen von Optimierungsaufgaben automatisiert gewonnen und im vorgestellten Rahmenwerk modelliert werden können, um eine einfache Erweiterung um Primitivrealisierungen zu ermöglichen, wird in Kapitel 3.6.3 skizziert.

Bewegungen mit mehreren Restriktionen

Liegen für eine oder mehrere zu verarbeitende Bewegungen nicht nur eine, sondern mehrere Restriktionen vor, etwa sowohl für die Geschwindigkeit als auch für die Beschleunigung, so können diese gesamtheitlich ausgenutzt werden.

Beispiel Kollisionserkennung: Hayward *et al.* [HAFG95] stellten eine (im Folgenden LP_{one} benannte) Optimierung der Realisierung von P_{one} vor, die neben Beschleunigungsrestriktionen auch Geschwindigkeitsrestriktionen der auf Kollisionen zu untersuchenden, sich bewegenden Objekte berücksichtigt. Der Korridor möglicher zukünftiger Positionen solcher Objekte entspricht einer Parabel, die zum frühesten Zeitpunkt, an dem das Objekt Maximalgeschwindigkeit erreicht, in eine lineare Funktion übergeht.

Auch das Verfahren P_{two} lässt sich entsprechend zu einem Verfahren LP_{two} abwandeln, welches (wie LP_{one} gegenüber P_{one}) eine gegenüber P_{two} vergrößerte Schrittweite aufweist, vergleiche Sondern [Son05, Kapitel 2.3.3, 4.1 und 4.2]. Das Verfahren LP_{one} weist dieselbe Konvergenzgeschwindigkeit (hinsichtlich Konvergenzordnung und asymptotischer Fehlerkonstante) wie P_{two} auf. Einzige Ausnahme bildet die Konvergenz gegen eine Nullstelle s der zu betrachtenden Funktion h mit $h'(s) = G_h$. In diesem Fall weist LP_{one} die Konvergenzgeschwindigkeit von L_{one} auf.

Bemerkung 3.5.4 Es ist zu bemerken, dass die beschriebenen Verfahren LP_{one} und LP_{two} korrekt, aber bezüglich ihrer Schrittweite suboptimal sind: Der Zeitpunkt t_{PL} des Überganges von einer parabelförmigen zu einer linearen Korridorbegrenzung ergibt sich für $f - g$ konservativ, aber nicht notwendigerweise exakt; es ist möglich, dass entweder f oder g bereits zu einem früheren Zeitpunkt Maximalgeschwindigkeit erreicht haben, so dass entweder f oder g bereits vor dem Zeitpunkt t_{PL} eine lineare Korridorbegrenzung besitzen. Um dies zu berücksichtigen, ist nicht nur die Betrachtung von $f - g$ und der Geschwindigkeitsrestriktion G_{f-g}, sondern die *separate* Betrachtung von f und g und der jeweils geltenden Maximalgeschwindigkeiten notwendig. Die separate Betrachtung der Restriktionen *einzelner* Operanden einer arithmetischen Verknüpfung von Bewegungsrepräsentationen ermöglicht somit eine Optimierung iterativer Nullstellensuchverfahren wie LP_{one} und LP_{two} bezüglich ihrer Schrittweite.

Weitere Restriktionstypen

Sind Restriktionen weiterer Typen, wie etwa eine Beschränkung der Beschleunigungsänderung, bekannt, so lassen sich analog zu den bereits besprochenen Restriktionstypen Verfahren zur Nullstellensuche auf arithmetischen Verknüpfungen solcherart beschränkter Bewegungen konstruieren.

Für Restriktionen mit oberer und unterer Schranke (zum Beispiel eine Beschleunigungsbeschränkung mit separaten Schranken für den Beschleunigungs- und den Bremsvorgang) gibt Sondern in ihrer Arbeit konkrete Modifikationen der Verfahren L_{one}, P_{one}, P_{two}, LP_{one} an [Son05, Kapitel 2.3.4]. Weitere praxisrelevante Restriktionstypen umfassen richtungs- bzw. dimensionsabhängige Restriktionen (etwa separate Geschwindigkeitsschranken für ein Flugobjekt und seine horizontalen bzw. vertikalen Bewegungen) sowie kontextabhängige Restriktionen (beispielsweise für sich in Verkehrsnetzen bewegende Fahrzeuge).

Verallgemeinerung für weitere arithmetische Verknüpfungen von Bewegungsrepräsentationen

Wie bereits erwähnt, lassen sich Strategien zur Nullstellensuche nicht nur für die Differenz $-(f, g)$ zweier Bewegungsrepräsentationen f und g, sondern allgemeiner für arithmetische Verknüpfungen h von Bewegungsrepräsentationen $f_1, \ldots, f_c$ anwenden.

Beispiel Kollisionserkennung in höheren Dimensionen: Wird die Aufgabenstellung der Kollisionserkennung auf in mehrdimensionalen Räumen sich bewegende Objekte erweitert, korrespondiert eine Kollision zweier Objekte mit einer Nullstelle ihrer Distanzfunktion.[24] Die folglich zur Kollisionserkennung auf Nullstellen zu untersuchende arithmetische Verknüpfung ist somit die Distanz $dist(f, g)$, angewendet auf zwei Bewegungsrepräsentationen f und g. Aus Bewegungsrestriktionen der

[24] Im Folgenden wird unter der Distanz implizit die euklidische Distanz verstanden. Die folgenden Aussagen zur Distanzfunktion sind jedoch auf beliebige Distanzen, die eine Metrik (vergleiche Forster [For99] für eine formale Definition) definieren, verallgemeinerbar.

Operanden f und g lassen sich wie im eindimensionalen Fall Restriktionen für die Distanzfunktion $dist(f, g)$ ableiten.

Für Beschränkungen G_f und G_g der absoluten Geschwindigkeiten ergibt sich durch Ausnutzung der Cauchy-Schwarzschen Ungleichung bzw. durch Regeln der Intervallarithmetik eine Beschränkung $G_{dist(f,g)} := G_f + G_g$, vergleiche [Son05, Lemma 3.1.1]. Für Beschleunigungsbeschränkungen der Operanden ergibt sich hingegen nur eine *untere* Schranke $B^+_{dist(f,g)}$ für die Distanz von f und g [Son05, Lemma 3.1.2]. Die Nichtexistenz einer oberen Schranke hängt mit der Positivität der Distanzfunktion zusammen: Bewegen sich zwei Objekte mit konstanter Richtung und Geschwindigkeit aneinander vorbei, so verhält sich die zweite Ableitung der Distanz dieser Punkte zum Zeitpunkt der geringsten Entfernung der Objekte umgekehrt proportional zu dieser Entfernung; im Falle einer Kollision der Objekte besitzt sie entsprechend eine Singularität. Eine untere Schranke $B^-_{dist(f,g)}$ wäre nur dann herzuleiten, wenn zusätzlich ein Mindestabstand der Objekte garantiert wäre.

Das Fehlen der oberen Schranke für $B_{dist(f,g)}$ impliziert, dass offline-Verfahren wie P_{two}, die Beschleunigungsrestriktionen oder Restriktionen noch höherer Ableitungen der Bewegungsrepräsentationen nutzen, nicht anwendbar sind. Online-fähige Verfahren wie L_{one} oder P_{one} lassen sich hingegen in gleicher Weise wie für die Differenz auch für die mehrdimensionale Distanz anwenden, wie auch bereits von Hayward *et al.* [HAFG95] beschrieben. Hayward *et al.* gaben zudem ein weiteres Beschleunigungsrestriktionen nutzendes Verfahren an, das gegenüber P_{one} eine größere Schrittweite für die mehrdimensionale Kollisionserkennung aufweist. Dieses nutzt aus, dass eine frühestmögliche Kollision zweier sich im Mehrdimensionalen bewegender Objekte sich nicht aus einem direkten Aufeinanderzusteuern der Objekte ergibt; vielmehr bezieht dieses Verfahren die aktuelle (d. h. die an t_i evaluierte) Bewegungsrichtung und -geschwindigkeit der Objekte mit ein, um eine optimale Richtung der Beschleunigung für eine frühestmögliche Kollision zu bestimmen. Ein derart optimiertes Verfahren lässt sich offensichtlich nur unter Betrachtung der individuellen Restriktionen der Operanden f und g, nicht aber durch Verwenden einer festen Restriktion $B_{dist(f,g)}$, realisieren.

Beispiel Kollisionswarnung: Sämtliche bislang vorgestellte Primitivrealisierungen lassen sich nicht nur für die Kollisionserkennung, sondern auch für die Kollisionswarnung in der in Abschnitt 3.2 verwendeten Formulierung adaptieren. Hierbei sind Differenz- bzw. Distanzfunktion nicht auf Nullstellen, sondern auf Stellen mit dem Funktionswert δ zu untersuchen. Als Primitiv dient daher die Nullstellenbestimmung für die arithmetische Verknüpfung der Differenz- bzw. Distanzfunktion, ergänzt um einen Summanden vom Betrage δ.

Die Kollisionswarnung ist gegenüber der Kollisionserkennung die in mancher Hinsicht einfacher bzw. auf vielfältigere Weise zu lösende Aufgabe. Insbesondere kann zu ihrer Lösung (in eingeschränkter Formulierung) eine Adaption von offline-Verfahren wie P_{two} auch im Mehrdimensionalen verwendet werden; Unterschreitungen einer Distanz δ können hiermit erkannt werden, da *vor* der Unterschreitung die Bedingung für eine untere Schranke $B^-_{dist(f,g)}$ erfüllt ist — allerdings würde die genauere weitere Beobachtung eines solchen Objektpaares für eine Zeitspanne, in der sich die Objekte weiter (bzw. beliebig nah) annähern, ein online-fähiges Verfahren wie P_{one}

erfordern.

Beispiel Kollisionserkennung und -warnung für ausgedehnt modellierte Objekte: Wie bereits in Abschnitt 3.2 ausgeführt, entspricht die Kollisionswarnung für punktförmige Objekte der Kollisionserkennung für kugelförmig ausgedehnte Objekte. Für die Kollisionserkennung lassen sich auch offline-Verfahren wie P_{two} einsetzen, wenn die betrachteten Objekte mit einer Ausdehnung etwa in Kugel- oder Polyedergestalt modelliert sind. Für durch Polyeder modellierte Objekte beliebiger Dimension basieren die zu betrachtenden Primitive auf Halbraumtests bezüglich der berandenden Facetten der Objekte, wie sie in ähnlicher Form auch zur Aufrechterhaltung von Datenstrukturen wie der kinetischen konvexen Hülle [Bas99] verwendet werden. Die Effizienz der Bewegungsrestriktionen nutzenden Verfahren und insbesondere der offline-Verfahren nimmt dabei mit der Ausdehnung des Objektes (in Relation zu den Größen der Bewegungsbeschränkungen) ab. Dies ist anschaulich darin begründet, dass für kleinere Objekte zwei aufeinanderfolgende Kollisionen der Berandungen solcher Objekte, d. h. ein Verschmelzen und anschließendes Trennen zweier sich bewegender Objekte, in kürzerem Zeitabstand erfolgen kann. Dieser Sachverhalt ist in Analogie zu der im vorigen Paragraphen für kugelförmig ausgedehnte Objekte erläuterten Aussage zu sehen, für die eine untere Schranke $B^{-}_{dist(f,g)}$ umgekehrt proportional mit dem Radius der Objekte wächst. Somit ist der Zeitraum, in dem eine einzelne Nullstelle eingegrenzt werden kann, entsprechend klein. Dies wiederum führt zu kleinen Schrittweiten bei der Berechnung des nächsten Iterationspunktes t_{i+1}. Sind die Objekte punktförmig modelliert, so sind offline-Verfahren nicht anwendbar.[25] Der Berechnungsaufwand zur Nullstellenbestimmung nimmt mit der Komplexität der Berandung der zu betrachtenden Objekte zu. Dies gilt für Bewegungsrestriktionen nutzende wie auch für alternative Verfahren, vergleiche Lin *et al.* [LMCG97] für eine Übersicht über Strategien und Verfahren zur Kollisionserkennung, sowie über Eingabeparameter und -charakteristika, die die Effizienz solcher Verfahren beeinflussen; eine genauere Beschreibung wesentlicher Verfahren zu dieser Aufgabenstellung finden sich bei Kirkpatrick *et al.* [KSS00], Guibas *et al.* [GNRZ02] und Basch *et al.* [BEG^{+}04]. Zudem existieren zur verwandten Aufgabenstellung der Kollisionswarnung (sowohl für ausgedehnte als auch für punktförmige Objekte) nicht-triviale Algorithmen, die eine effiziente Laufzeit hinsichtlich der Komplexität der Anzahl sowie der Berandungsbeschreibungen der betrachteten Objekte bieten, vergleiche Agarwal *et al.* [AS00]. Dies ist insofern bemerkenswert, da für die Kollisionserkennung punktförmiger Objekte keine Algorithmen bekannt sind, die bezüglich der Anzahl der zu betrachtenden Objekte eine nicht-triviale Laufzeitkomplexität aufweisen; die Ausnahme bilden Bewegungen im eindimensionalen Raum, für die effiziente Segmentschnittalgorithmen adaptierbar sind, vergleiche Abschnitt 3.2.

Mehrere Verfahren zur mehrdimensionalen Kollisionswarnung und -erkennung

[25] Diese Nichtanwendbarkeit gilt zudem für ausgedehnte Objekte in Anwendungskontexten, in denen für Kollisionen eine Interaktion — etwa eine Abstoßung — der kollidierenden Objekte modelliert ist. Eine solche spontane Reaktion der Objekte auf die zu identifizierenden Ereignisse (z.B. Kollisionen) muss in den meisten Aufgabenstellungen zur Bewegungsanalyse nicht berücksichtigt werden; als Beispiele seien hier die Aufrechterhaltung der kinetischen konvexen Hülle oder die Sortierung von sich bewegenden Objekten in einer räumlichen Dimension ihrer Bewegung genannt.

für punktförmige Objekte, die (auch Restriktionen nutzende) Primitive zur Nullstellenbestimmung verwenden, werden auch in der Arbeit von Sondern [Son05] vorgestellt. Die Interaktion der Verfahren mit den Bewegungsrepräsentationen der Eingabe erfolgt durch alleinige Abstützung auf die Primitive zur Nullstellenbestimmung der Differenz und/oder Distanz zweier Bewegungsrepräsentationen. Zusätzlich wird bei Sondern die Abhängigkeit der Effizienz dieser Algorithmen von verschiedenen Eigenschaften der Eingabe sowie von der gewählten Realisierung obiger Primitive erläutert.

3.6 Modellierung von Bewegungsrestriktionen ausnutzenden Primitivrealisierungen

Im vorigen Abschnitt wurde belegt, dass sich Primitive zur Nullstellenbestimmung zuverlässig unter Ausnutzung von verschiedenen Bewegungsrestriktionen realisieren lassen. Diese Primitivrealisierungen arbeiten unabhängig von den Repräsentationstypen der betrachteten Bewegungen, d. h. insbesondere unabhängig von ihren mathematischen Beschreibungen. Einzelne Realisierungen unterscheiden sich hinsichtlich der für die Bewegungen geltenden Bewegungsrestriktionen bzw. sind zumindest durch diese parametrisiert. Für eine Verwendung solcher Primitivrealisierungen innerhalb des vorgestellten Rahmenwerkes ist zu konkretisieren, wie sich folgende Elemente in die in Abschnitt 3.3 vorgestellte erweiterte Modellierung des in dieser Arbeit vorgesetellten Rahmenwerks integrieren lassen:

- Verschiedene Typen von Bewegungsrestriktionen.
- Verschiedene Primitivvarianten, die die Nullstellenbestimmung jeweils einer arithmetischen Verknüpfung (von Bewegungsrepräsentationen) leisten. Ziel ist die Modellierung eines generischen Primitivs, das durch eine arithmetische Verknüpfung parametrisiert ist.
- Verschiedene Bewegungsrepräsentationen nutzende Realisierungen von Primitiven bzw. des generischen Primitivs.
- Ein parametrisierbares und durch ein `Decider`-Objekt abzubildendes Auswahlverfahren, das zu einem Primitivaufruf eine geeignete Realisierung auswählt.
- Eine Verfahren, das die einfache Erweiterbarkeit und Anpassung der Nullstellenbestimmung bei Ergänzung zusätzlicher Instanzen der oben genannten Modellierungselemente ermöglicht und organisiert. Idealerweise sollte diese Verfahren automatisiert für die Modellierungselemente angepasste Primitivrealisierungen erzeugen und in das Rahmenwerk integrieren.

3.6.1 Modellierung von Bewegungsrestriktionen

Jeder Typ von Bewegungsrestriktionen, wie Geschwindigkeits- oder Beschleunigungsbeschränkungen, ist durch eine eigene Klasse abgebildet, die die Schnittstelle `MotionRestriction` implementiert. Instanzen dieser Klassen besitzen als Attribut eine

numerische Angabe der Restriktion, welche über eine Methode `getBounds()` abrufbar ist.[26] Für die Interpretation dieser Angabe ist die Kenntnis über den Typ der Restriktion notwendig; daher beinhaltet die Schnittstelle `MotionRestriction` eine Methode `uniqueRestrictionID`, die diesen Typ zurückgibt.[27]

Um auf Restriktionen von Bewegungsrepräsentationen (und von arithmetischen Verknüpfungen) von außen zuzugreifen, aggregiert jedes Objekt vom Typ `Function` eine oder mehrere Instanzen von Realisierungen der Schnittstelle `MotionRestriction`, welche über Aufruf der öffentlichen Methode `getRestrictions()` abgefragt werden können.

Abbildung 3.10 zeigt die bereits beschriebene Modellierung von Restriktionen. Als konkrete Restriktionstypen sind Beschränkungen von Geschwindigkeit und Beschleunigung, jeweils mit absoluten als auch mit richtungsabhängigen Schranken, dargestellt. Die weiteren dargestellten Restriktionen werden in Kapitel 3.6.2 näher erläutert.

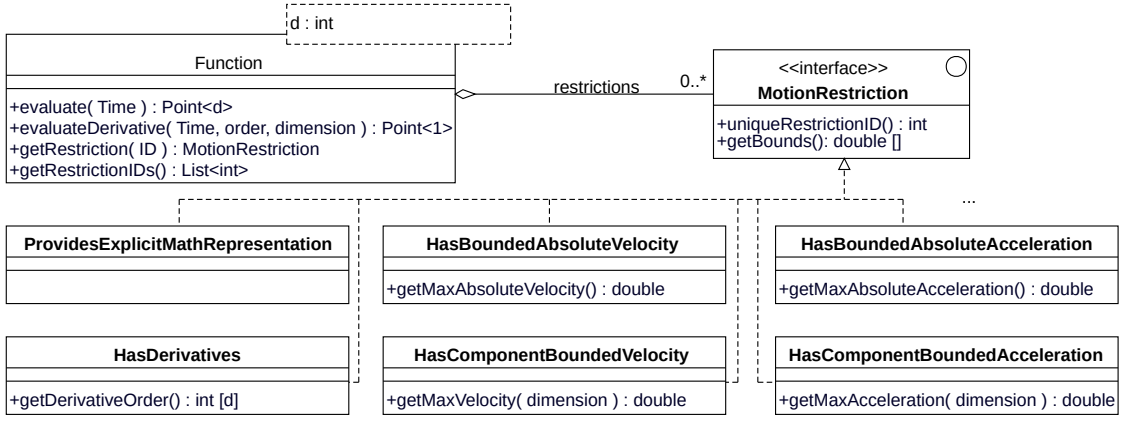


Abbildung 3.10: Modellierung von Restriktionen.

Bemerkung 3.6.1 Semantisch scheint es nahe liegender, Bewegungsrestriktionen zu Instanzen der Klasse `Mpoint` zu aggregieren. Auch die Aggregation von Bewegungsrestriktionen zum Typ des sich bewegenden Objektes, das durch eine Instanz von `Mpoint` repräsentiert wird, scheint sinnvoll: Im Beispiel der Flugverkehrsüberwachung könnten Bewegungsparameter und -restriktionen der einzelnen Flugzeugtypen, wie sie die BADA-Datenbank bereitstellt [Eur99, NPI+05], gekapselt werden; jedem konkreten Flugzeug wäre sein Typ assoziiert, während seine Bewegung nach wie vor durch eine Instanz der Klasse `Mpoint` modelliert wäre. Diese Varianten der Vorhaltung von Bewegungsrestriktionen werden im vorgestellten Rahmenwerk nicht ausgeschlossen. Stattdessen werden alle Bewegungsrestriktionen, die zu einer Instanz

[26] Diese numerische Angabe kann aus mehreren numerischen Werten bestehen, etwa wenn der Restriktionstyp individuelle Schranken für einzelne räumliche Dimensionen, in denen die Bewegung stattfindet, vorsieht. Daher ist diese Angabe als Feld von numerischen Werten realisiert.

[27] In Programmiersprachen wie `Java` ist diese Methode nicht zwingend erforderlich, da zum einen ein Objekt seine Klassenzugehörigkeit bekannt geben kann und zum anderen die Menge Klassen, die eine konkrete Schnittstelle implementieren, abgefragt werden kann, vergleiche [AG96].

von `Mpoint` verfügbar sind, zur Instanz von `Function` aggregiert, die die Bewegung der `Mpoint`-Instanz beschreibt. Die möglicherweise resultierende Redundanz ist auf Grund des relativ kleinen Datenvolumens einer Restriktionsbeschreibung akzeptabel. Der Vorteil dieser Datenhaltung liegt darin, dass auch für Instanzen von `Function`, die eine arithmetische Verknüpfung von Bewegungsrepräsentationen darstellen, Restriktionen aggregiert werden können — und zwar in der gleichen Weise wie für einzelne Bewegungsrepräsentationen.

3.6.2 Modellierung der Auswahl einer Primitivrealisierung

Modellierung von Primitiven und ihrer Realisierungen Die Modellierung von Primitiven, d. h. Verfahren zur Nullstellenbestimmung, ist zu Teilen bereits in Abschnitt 3.3 erläutert. Einige Details der Modellierung konkretisieren sich durch die Option, Bewegungsrestriktionen nutzende Verfahren zur Nullstellenbestimmung zu verwenden. Für die Nullstellenbestimmung ist ein einziges Primitiv modelliert, dem bei Aufruf die auf Nullstellen zu untersuchende Funktion, genauer eine arithmetische Verknüpfung, die Referenzen auf ihre Operanden enthält, übergeben wird. Für einen konkreten Aufruf wird durch ein *Decider*-Objekt eine geeignete Realisierung des Primitivs, das die Nullstellenbestimmung leistet, ausgewählt. Diese Auswahl ist insbesondere von den Restriktionen der Operanden abhängig.

Modellierung von Ableitbarkeit Da für die Ausnutzung einiger Restriktionen wie Beschleunigungsbegrenzungen neben der aktuellen Position auf weitere Daten über den Status der Bewegung wie die aktuelle Geschwindigkeit zugegriffen werden muss, ist die abstrakte Klasse `Function` um die Methode `evaluateDerivative()` erweitert. Ob eine konkrete Bewegungsrepräsentation tatsächlich solche Daten liefert, ist als Restriktionstyp durch die Klasse `hasDerivatives` modelliert.

Verwendung algebraischer Primitivrealisierungen Um anzuzeigen, dass eine Bewegungsrepräsentation ihre mathematische Beschreibung bereitstellt, aggregiert das entsprechende `Function`-Objekt eine Instanz von `ProvidesExplicitMathRepresentation`, vergleiche Abbildung 3.10. Dies ermöglicht die Überprüfung, ob algebraische und semi-algebraische Primitivrealisierungen, siehe Abschnitt 3.4.1, angewendet werden können (als Alternativen zu Bewegungsrestriktionen nutzenden Verfahren). Ein Zugriff auf die in der `Function`-Instanz hinterlegte mathematische Beschreibung kann dann entsprechend dem bekannten *Factory*-Entwurfsmuster erfolgen; alternativ kann hierfür eine programmiersprachenspezifische Zugriffsmethodik wie die `Java` *Reflection*-API, vergleiche Arnold und Gossling [AG96], verwendet werden.

Restriktionen gesteuerte Auswahl von Primitivrealisierungen Durch Modellierung der beiden oben beschriebenen speziellen Restriktionstypen ist es nun möglich, die Auswahl einer Primitivrealisierung allein durch Betrachtung der `MotionRestriction`-Instanzen, die zu den Operanden des Primitivaufrufs aggregiert sind, treffen zu können.

Die beschriebene Kapselung der Kenntnisse über die im Rahmenwerk implementierten Bewegungsrepräsentationstypen und über die jeweils geeigneten Primitivrealisierungen (und damit auch der Verantwortung für die Auswahl geeigneter Primitivrealisierungen für spezifische Primitivaufrufe) ist eine spezielle Lösung des *double-dispatch*-, bzw. des *mulitiple-dispatch*-Entwurfsproblems, vergleiche Gamma *et al.* [GHJV95]. Dieses Problem tritt auf, wenn Resultate einer Operation, und insbesondere der Typ dieser Resultate, vom Typ mehrerer Operanden abhängen. Einige Programmiersprachen wie beispielsweise Smalltalk bieten Mechanismen, um dieses Problem direkt zu lösen. In vielen objektorientierten Sprachen ist hingegen nur *single dispatching* unterstützt, vergleiche Gamma *et al.* [GHJV95, Seite 338]. In *single-dispatch*-Sprachen wie `C++` [Str00] oder `Java` [AG96] kann die zu verwendende Realisierung bei einem Operationsaufruf `object.operate(operand)` nur von der Art der Operation (d. h. vom Methodennamen) sowie vom Typ *eines*, nämliches des aufrufenden, Objektes abhängig gemacht werden; die *konkreten* Typen weiterer Operanden können im allgemeinen Fall nicht (bzw. nur durch explizite Fallunterscheidungen innerhalb der aufgerufenden Methode) in die Auswahl der Realisierung mit einbezogen werden.

Für die Wahl der geeigneten Realisierung einer Primitivoperation zur Nullstellenbestimmung sind hingegen die Typen *aller* betrachteten, sich bewegenden Objekte (bzw. ihrer Bewegungsrepräsentationen) ausschlaggebend, die Operanden der zu untersuchenden Verknüpfung sind. Es existieren Möglichkeiten, Varianten des *double dispatch*-Problems wie die vorliegende auch in *single-dispatch*-Sprachen zu lösen. Ein Beispiel einer solchen Lösung wird von Fussel [Fus00] veranschaulicht. In diesem wird das *double dispatch*-Problem für arithmetische Operationen wie der Addition und für verschiedene Datentypen, die als Operanden solcher Verknüpfungen akzeptiert werden sollen, beschrieben. Der innerhalb des Rahmenwerks abzubildende Fall ist allerdings komplexer und kann daher nicht in analoger Weise behandelt werden: Die Auswahl einer Realisierung einer Primitivoperation hängt nicht nur vom Typ der Operanden ab, sondern von mehreren ihrer Eigenschaften; konkret sind für die Wahl — neben der arithmetischen Verknüpfung selbst — nicht nur die Typen der übergebenen Bewegungsrepräsentationen, sondern ihre aggregierten *Mengen* von `MotionRestriction`-Instanzen relevant. Diese Konkretisierung des *double dispatch*-Problems lässt sich durch Adaption des von Gamma vorgestellten *Visitor*-Entwurfsmusters lösen. Im vorliegenden Fall bedeutete dies die Kapselung jeder Primitivoperation in einer eigenen Klasse. Dieser Aufwand ist insofern reduziert, als dass die in Abschnitt 3.3.1 vorgestellte Modellierung nur ein *einziges*, generisches (Nullstellen bestimmendes) Primitiv vorsieht, das nicht nur durch Bewegungsrepräsentationen, sondern auch durch deren arithmetische Verknüpfung parametrisiert ist.

Des Weiteren sähe das *Visitor*-Entwurfsmuster vor, in jeder Subklasse von *Function* eine Methode `revealOperandInfos()` zu erstellen, die die für den Aufruf der Primitivoperation notwendigen Informationen zu einzelnen Operanden, d. h. zu *Function*-Instanzen, bereit stellt. Scheinbar würde dies die Modularität und Erweiterbarkeit des Rahmenwerks stark einschränken. Die beschriebene Modellierung von Restriktionen erlaubt jedoch, für alle `Function`-Instanzen *dieselbe* Methode `revealOperandInfos()` bzw. `getRestrictions()` aufzurufen; diese gibt lediglich die

zu einer `Function`-Instanz aggregierten Restriktionen zurück. Einzelne Restriktionen werden dabei durch ihren Typ und ihre numerische Angabe beschrieben, die jeweils über `getUniqueID()` bzw. `getBounds()` zugreifbar sind. Diese Methode kann somit innerhalb der abstrakten Klasse `Function` implementiert werden und muss nicht für Subklassen von `Function` konkretisiert werden. Zudem ist zu bemerken, dass die so abgerufenen Informationen der Operanden nur für die *Auswahl* einer geeigneten Realisierung des aufgerufenen Primitivs, nicht aber für die anschließende Arbeit der ausgewählten Primitivrealisierung benötigt werden. Die Kapselung dieser Auswahl erfolgt, wie bereits in Abschnitt 3.3 beschrieben, durch ein *Decider*-Objekt.

Konsequenzen Der so modellierte Auswahlprozess nutzt ausschließlich die Kenntnis über die zur Verfügung stehenden Primitivrealisierungen und über ihre jeweilige Anwendbarkeit für die dem Aufruf des Primitivs übergebene arithmetische Verknüpfung sowie für deren Operanden. Im Falle der Erweiterung des Rahmenwerks um neue Restriktionstypen, arithmetische Verarbeitungen oder Primitivrealisierungen ist eine Modifizierung des Auswahlprozesses angeraten oder notwendig; für eine Modellierung dieses Prozesses durch ein *Decider*-Objekt ist dieses entweder zu modifizieren oder zu ersetzen.

Parallelen zur Ausführungsplanerstellung in traditionellen Datenbanksystemen Der Prozess zur Auswahl einer Primitivrealisierung steht in einer Analogie zur Erstellung eines Ausführungsplanes, wie sie bei der Optimierung der Anfragebearbeitung auch in traditionellen Datenbanksystemen durchgeführt wird:[28] In beiden Kontexten wird eine Anfrage an den Datenbestand hinsichtlich der referenzierten Datenbankinhalte sowie hinsichtlich der Art ihrer Verknüpfung untersucht: In der Anfrageoptimierung etwa im relationalen Datenmodell werden die Datenbankrelationen sowie die (Kompositionen von) Operatoren der relationalen Algebra betrachtet, die in der Anfrage referenziert werden; Ergebnis einer solchen Analyse ist ein Ausführungsplan, der aus einer Abfolge von Operationen besteht, deren Durchführung das Anfrageergebnis möglichst effizient generiert. Im in dieser Arbeit betrachteten Kontext der Nullstellenbestimmung für zeitvariante Funktionen besteht der „Ausführungsplan“ aus der Angabe (und Parametrisierung) einer möglichst geeigneten Primitivrealisierung.

Bemerkung 3.6.2 Aus Sicht des objektorientierten Software-Engineerings stellt sich die Frage, ob die Modellierung des Auswahlprozesses durch ein zentrales *Decider*-Objekt adäquat ist, d. h., ob nicht stattdessen eine Modularisierung des Auswahlprozesses oder eine erweiterbare Vererbungshierarchie von *Decider*-Klassen sinnvoll realisierbar und vorteilhafter wären. Es scheinen jedoch keine wesentlichen Argumente gegen die gewählte, zentrale Modellierung zu sprechen. Diese Einschätzung ist durch die Analogie zur — zumeist zentral und nicht modularisiert implementierten — Ausführungsplanerstellung in traditionellen Datenbanksystemen fundiert. Allerdings existieren durchaus Ansätze mit der Zielsetzung, die Ausführungsplanerstellung (insbesondere objektorientierter) Datenbanksysteme zu modularisieren. Einen

[28] Eine Einführung in die Anfrageoptimierung in Datenbanksystemen findet sich beispielsweise bei Silberschatz *et al.* [SKS05, Kapitel 14].

solchen Ansatz sowie eine Übersicht über die (zumeist nicht zufrieden stellende) Erweiterbarkeit der Anfrageoptimierungskomponenten bestehender Datenbanksysteme wird von Breimann [Bre04] beschrieben.

Es sei zusätzlich auf die vom hier vorgestellten Rahmenwerk bereit gestellte Möglichkeit verwiesen, für eine jede Anwendung das zu verwendende *Decider*-Objekt spezifisch anzupassen bzw. auszutauschen. Die Evaluierung alternativer, eventuell modularerer Ansätze sei hier als ein Aspekt der Fortentwicklung des Rahmenwerks genannt.

3.6.3 Erweiterung durch automatisierte Generierung von Primitivrealisierungen

Das in der Aufzählung zu Beginn des Abschnittes 3.6 letztgenannte Modellierungsziel fordert die einfache Erweiterbarkeit und Anpassung der Nullstellenbestimmung im Falle, dass das System um zusätzliche Typen von Bewegungsrepräsentationen oder Restriktionen oder weitere arithmetische Verknüpfungen ergänzt wird. Hierfür werden im Folgenden Lösungsansätze skizziert, die in eine Realisierung des hier beschriebenen Rahmenwerks einfließen könnten.

Das genannte Implementierungsziel einer möglichst generischen und automatisierbaren Erweiterbarkeit wird für viele Problemstellungen angestrebt. In vielen Fällen steht dieses Ziel jedoch in Konflikt mit dem Ziel der größtmöglichen Effizienz einer Implementierung. Dies trifft punktuell auch für die Modellierung von Primitiven zur Nullstellenbestimmung und ihren individuellen Realisierungen, die Bewegungsrestriktionen ausnutzen, zu. Zwar lassen sich bei Integratation neuer Bewegungsrestriktionstypen und neuer arithmetischer Verknüpfungen geeignete Primitivrealisierungen sogar autmatisiert generieren; diese Generierung ist jedoch algorithmisch nur aufwändig zu realisieren, insbesondere wenn diese Realisierungen eine größtmögliche Effizienz bieten sollen. Die folgende Aufzählung fasst Argumente für die letztgenannte These zusammen. Anschließend werden Strategien und Techniken zur automatischen Generierung von Primitivrealisierungen skizziert.

- Für (insbesondere mehrfach) restringierte Bewegungen bietet die separate Betrachtung der Restriktionen der einzelnen Bewegungen Effizienzvorteile gegenüber der Betrachtung allein der Restriktionen der Verknüpfung h dieser Bewegungen, was in Bemerkung 3.5.4 ausgeführt wird.

- Auch für Verknüpfungen von insbesondere mehrdimensionalen Bewegungen existieren Primitivrealisierungen, die dadurch Effizienzvorteile gewinnen, dass nicht nur die Restriktionen einer Verknüpfung h, sondern die Operanden von h einzeln betrachtet werden. Ein Beispiel hierfür ist das in Kapitel 3.5.4 beschriebene Verfahren von Hayward *et al.* [HAFG95], das Beschleunigungsrestriktionen zur Kollisionserkennung in mehrdimensionalen Räumen nutzt.
  Allgemeiner gesprochen ergeben sich für komplexere Verknüpfungen iterative Primitivrealisierungen mit optimaler Schrittweite nur als Lösungen aufwändiger Optimierungsaufgaben, wie noch ausgeführt werden wird.

- Für Restriktionen nutzende Primitivrealisierungen ergibt sich der jeweils nächste der iterativ zu bestimmenden Zeitpunkte t_{i+1} algebraisch; für die Differenz zweier geschwindigkeitsbeschränkter Bewegungen ergibt er sich als Lösung einer linearen Gleichung. Sind jedoch höhere Ableitungen der Bewegungen restringiert, etwa Beschleunigung oder deren Änderung, so verkompliziert sich auch die Bestimmung des Zeitpunktes t_{i+1} sowie die Gewinnung der Vorschrift zu seiner Bestimmung.

- Die Komplexität der Nullstellenbestimmung wächst auch mit der algebraischen Komplexität der arithmetischen Verknüpfung, deren Nullstellen zu bestimmen sind. Die multidimensionale Distanz beispielsweise verdoppelt den Polynomgrad der Gleichung, über deren Lösung sich das Inkrement bestimmt.

Optionen zur automatisierten Generierung von Primitivrealisierungen Im Folgenden wird diskutiert, welche Forschungsergebnisse und Softwaresysteme sich für eine einfache Erweiterbarkeit des Rahmenwerkes um Restriktionstypen und arithmetische Verknüpfungen nutzbar machen lassen.

Besitzen alle Operanden einer arithmetischen Verknüpfung von Bewegungsrepräsentationen Restriktionen ausschließlich desselben Typs (bzw. derselben Typen), so lassen sich Restriktionen dieses Typs (bzw. dieser Typen), wie bereits beschrieben, auch für das Ergebnis einer arithmetischen Verknüpfung h mit Hilfe von Techniken der Intervallhalbierung bestimmen. Aus diesen Restriktionen für h lässt sich für die Nullstellenbestimmung eine Vorschrift zur Gewinnung von Iterationspunkten t_{i+1} gewinnen.

Bei online-fähigen Verfahren ergibt sich der zulässige Iterationspunkt t_{i+1} durch Auflösung eines Gleichungssystems. Dieses besteht aus den Ungleichungen, die zu den bekannten Restriktionen korrespondieren, der Forderung $t_{i+1} \geq t_i$ (dass also t_{i+1} zeitlich nicht vor t_i liegt) sowie der Forderung, dass zwischen t_i und t_{i+1} kein Vorzeichenwechsel stattfindet.[29]

Der spätestmögliche zulässige Zeitpunkt kann somit als Lösung einer Optimierungsaufgabe mit Randbedingungen verstanden werden. Solche Lösungen sind insbesondere auch zu erhalten, wenn die Operanden Restriktionen unterschiedlichen Typs besitzen, wie in Abschnitt 3.5.4 beispielhaft beschrieben. Ähnliche Optimierungsprobleme mit einem ebenfalls physikalischen Hintergrund werden in der Literatur zu optimalen Steuerprozessen besprochen, vergleiche Bryson und Ho [BH75].

Für das Lösen von Optimierungsaufgaben existieren zuverlässige Softwarepakete, beispielsweise die Bibliothek zur angewandten Optimierung für die Software MATLAB, vergleiche Venkataraman [Ven02]. Allerdings ist die generische Anwendbarkeit zur Primitivrealisierung für arithmetische Verknüpfungen und Restriktionskombinationen zwangsläufig durch die Komplexität der Gleichungssysteme, die eine solche

[29] Im Allgemeinen ist die letztgenannte Forderung nur mit Hilfe von Existenzquantoren auszudrücken. Nur durch Betrachtung des Kontextes der Nullstellenbestimmung ist eine effizientere Abbildung dieser Forderung möglich: Beispielsweise für die Modellierung von Geschwindigkeitsrestriktionen, die durch lineare Funktionen abgebildet werden können, kann diese auch durch einfache Ungleichungen ausgedrückt werden, sofern nicht nur die tatsächliche Bewegung, sondern auch die zu den Restriktionen korrespondierenden Korridore modelliert werden.

Software zu lösen vermag, beschränkt. Analoges gilt für Systeme zur automatisierten Beweisführung, vergleiche Paulson [Pau87], sowie für die Implementationen zur Lösung von Ungleichungen, wie sie in Constraint-Datenbanksystemen zum Einsatz kommen. Wie erwähnt, unterstützen solche System bis dato zumeist nur lineare Constraints. Primitivrealisierungen, die die Beschränkung höher Ableitungen von Bewegungsrepräsentationen (z.B. der Beschleunigung) ausnutzen, wären hiermit nicht zu erhalten; in Abschnitt 2.1.1 sind Ansätze zur Verwendung nicht-linearer Ungleichungen in Constraint-Datenbanken besprochen.

Bemerkung 3.6.3 Auch offline-Verfahren wie P_{two} ließen sich als Resultate von mit Constraints formulierbaren Anfragen erhalten. Hierfür ist der Sachverhalt, dass genau eine Nullstelle zwischen zwei Zeitpunkten t_i und t_{i+1} existiert, durch Constraints zu formulieren, was die Verwendung von Exisitenzquantoren zwingend notwendig macht; die Verwendung von Quantoren führt dazu, dass entsprechende Constraint-Anfragen nur mit deutlich erhöhtem Aufwand durch automatisiertes Contraint-Lösen zu beantworten sind, vergleiche Kuper *et al.* [KPL99]. Letztere Einschränkung kann als ein Hinweis darauf gesehen werden, dass ein automatisiertes und sehr allgemein verwendbares Anfragesystem zur Lösung von Ungleichungen nur bedingt geeignet ist, die speziellen Fragen nach Iterationsvorschriften für die Nullstellenbestimmung unter Ausnutzung von Bewegungsrestriktionen zu beantworten.

Will man auf die aufwändigen und zeitintensiven Techniken zur exakten Lösung solcher Ungleichungssysteme bzw. Optimierungsprobleme verzichten, so können diese Lösungen auch approximiert werden. Hierfür ließen sich erneut Regeln der Intervallarithmetik verwenden, vergleiche wiederum [Kea96, HHKR95, GK99]. Die so zu erhaltenden Schätzungen für den optimalen Iterationszeitpunkt t_{i+1} sind konservativ, so dass die Verfahren zur Nullstellenbestimmung verlässlich bleiben.

Modellierung der automatisierten Realisierung von Primitiven Man könnte ein Sofwaresystem zur automatisierten Lösung von Optimierungsproblemen verwenden, um innerhalb des Rahmenwerks Verfahren zur Nullstellenbestimmung (d. h. Primitivrealisierungen) für diverse arithmetische Verknüpfungen und für Operanden mit diversen Bewegungsrestriktionen. Hierfür hätten Bewegungsrestriktionstypen neben der Methode `getBounds()` eine Methode `getInequalities()` zu realisieren. Eine jeweilige Instanz würde hierüber die Ungleichungen publizieren, durch die sie, d. h. die Bewegungsrestriktion, beschrieben ist. Diese Ungleichungen können so als Nebenbedingungen in das Optimierungsproblem für die Bestimmung von t_{i+1} eingehen. Somit wären bei einem Erweitern des Rahmenwerks um weitere Restriktionen oder arithmetische Verknüpfungen effiziente Primitivrealisierungen (zu den ergänzten Elementen) automatisiert zu erhalten.

Bemerkung 3.6.4 Es ergäbe sich sogar die Option, die Generierung von Primitivrealisierungen nicht bei Erweiterung des Rahmenwerks, sondern zur Laufzeit für jeden Primitivaufruf einzeln vorzunehmen. Effizienter scheint es jedoch, die bereits konstruierten Primitivrealisierungen im System vorzuhalten und bei Bedarf auf diese zuzugreifen.

3.7 Implementierung

Die in diesem Kapitel beschriebenen Konzepte sind größtenteils in eine Erweiterung des in Kapitel 2 beschriebenen Rahmenwerkes eingeflossen. Eine erste Erweiterung des im Geo-Datenbankkernel GOODAC [Voi98] implementierten Rahmenwerks wurde als Machbarkeitsstudie von Iris Puke im Rahmen ihrer Diplomarbeit [Puk03] implementiert. Die dieser Studie zu Grunde liegende Modellierung fokussiert auf die Aufgabenstellung der Kollisionserkennung und ist in einer weiterentwickelten Form bei Blunck *et al.* [BHPV04] beschrieben. Später wurde gemeinsam mit der Diplomandin Joelle Sondern eine Erweiterung einer Implementierung des Rahmenwerkes in der Programmiersprache `Java` [AG96] erstellt. Die dieser zweiten Implementierung zu Grunde liegende Modellierung ist generischer als die der erstgenannten Erweiterung des GOODAC-Systems und fußt auf den in dieser Arbeit (und bei Blunck *et al.* [BHSV06]) geschilderten Konzepten und Modellierungselementen. Als Teil dieser Implementierung sind die in Abschnitt 3.5 beispielhaft vorgestellten, Restriktionen nutzenden Primitivrealisierungen zur Nullstellenbestimmung von arithmetischen Verarbeitungen von Bewegungsrepräsentationen integriert. Weiterhin implementiert sind verschiedene Algorithmen zur eindimensionalen und mehrdimensionalen Kollisionswarnung und -erkennung, die sich auf die genannten Primitive abstützen. Einige dieser Algorithmen sind Varianten des von Bentley und Ottmann realisierten Segmentschnittalgorithmus [BO79].[30]

Eine Skizze über Gegenstände der weiteren Forschung im Kontext des hier vorgestellten Rahmenwerkes und für seine Erweiterung findet sich in der Zusammenfassung dieser Arbeit in Kapitel 7.

[30] Implementierungen der für diese Verfahren verwendeten Datenstrukturen sind zu großen Teilen von Jan Vahrenhold bereit gestellt worden.

Kapitel 4

Ein Überblick über speichereffiziente Algorithmen

Ziel bei der Entwicklung speichereffizienter Algorithmen ist es, neben dem Speicher, der die Eingabe enthält, möglichst wenig zusätzlichen Speicher zu nutzen und dennoch eine optimale oder nahezu optimale Laufzeit zu erhalten. In der Vergangenheit ist mit dem Sinken der Kosten für Speicherressourcen der Aspekt der Speichereffizienz beim Design und bei der Entwicklung von Algorithmen in den Hintergrund getreten [Ben84]. Jedoch ist trotz der weiterhin fallenden Speicherkosten das Interesse an speichereffizienten Algorithmen wieder gestiegen [BIK+04], da die von Anwendungen wie beispielsweise Geo-Informationssystemen [Wor95] zu verarbeitenden Daten dank neuerer Datengewinnungstechniken in größerer Menge und Genauigkeit zur Verfügung stehen. Noch wesentlicher für das erstarkte Interesse an speichereffizienten Algorithmen ist jedoch die rasche Fortentwicklung und Verbreitung von mobilen Endgeräten, wie beispielsweise Mobiltelefonen oder PDAs, die beschränktere Speicherressourcen als stationäre Computer aufweisen. Diese Geräte bieten inzwischen auch komplexere Dienstleistungen an, die die Kenntnis über den Standort solcher mobilen Geräte verwenden, um dem Nutzer für ihn angepasste und relevante Informationen zu liefern. Ein Beispiel für solche *lokationsbasierten Dienste* [HTKR05] ist beispielsweise die Vorauswahl von „*points of interest*“ wie Restaurants oder Hotels nach mehreren wählbaren Kriterien, unter anderem nach der Nähe zum momentanen Standort des Nutzers. Diese Dienste machen die Verarbeitung größerer (und in ihrer Größe eventuell nicht absehbarer) Datenmengen notwendig; Operationen wie Selektion oder Aggregation werden in vielen Szenarien direkt auf den mobilen Endgeräten ausgeführt, deren Speicherressourcen häufig beschränkt sind.[1] Daher hat mit der Bedeutung der lokationsbasierten Dienste auch die Erforschung speichereffizienter Algorithmen an Relevanz gewonnen.[2]

[1] Einschränkend ist hier anzumerken, dass auch für mobile Endgeräte die Kosten für Speicherressourcen stark im Sinken begriffen sind. Jedoch ist bei der Entwicklung lokationsbasierter Dienste ein wesentliches Ziel die zuverlässige Anwendbarkeit für möglichst viele Typen mobiler Endgeräte. Somit scheint die Unterstützung von Geräten erstrebenswert, deren Speicherressourcen stark beschränkt oder unbekannt sind.

[2] Einen Überblick über lokationsbasierte und mobile Dienste sowie über Strategien für deren effiziente Realisierung bieten beispielsweise Höpfner *et al.* [HTKR05]; Szenarien, in denen Datenverarbeitung direkt auf den ressourcenbeschränkten Endgeräten erfolgt, werden in den Kapiteln 5

Sensornetzwerke stellen einen weiteren neuen Anwendungsbereich dar, in dem Datenverarbeitung mit beschränkten Speicherressourcen notwendig wird.[3] Sensornetzwerke haben die Durchführung und Organisation von dezentraler Datengewinnung und -verarbeitung zur Aufgabe: In Sensornetzwerken sind einzelne mit Sensoren versehene Geräte, auch Sensor-Computer genannt, für die Erfassung von Daten sowie deren Analyse und Verbreitung zuständig. Hierzu baut eine Menge solcher Sensor-Computer eigenständig ein Kommunikationsnetzwerk auf und passt dieses bei Ausfall oder Umpositionierung einzelner Teilnehmer geeignet an. Die Hardwareressourcen dieser Sensor-Computer sind begrenzt; unter anderem ist auch der zur Verfügung stehende Speicherplatz beschränkt.[4] Sensor-Computer sammeln in vielen Szenarien kontinuierlich große Datenmengen, deren (wahllose) Weiterleitung bzw. Verbreitung die begrenzten Energieressourcen sowie die Bandbreite des Kommunikationsnetzes strapazieren würde [LM03, LHY+04]. Aus diesem Grunde sollten die einzelnen Sensor-Computer in der Lage sein, Daten vor deren Sendung anforderungsspezifisch und intelligent zu komprimieren und insbesondere zu aggregieren und zu selektieren [DNW05, LHY+04, LM03], vergleiche auch Kapitel 5 dieser Arbeit. Bei der Entwicklung entsprechender Techniken und Algorithmen sind auch die beschränkten Speicherressourcen der Sensor-Computer zu berücksichtigen [SS04, LM03]. Speichereffiziente Algorithmen können daher die Leistungsfähigkeit eines Sensor-Computers erhöhen sowie seine Energiekosten senken und damit seine Einsatzdauer verlängern. Dies impliziert einen vergrößerten Einsatzbereich und eine höhere Zuverlässigkeit von Sensor-Netzwerken.

Zusätzlich können auch Anwendungsfelder, in denen der zur Verfügung stehende Speicher weniger starken Beschränkungen unterliegt, von der Verwendung speichereffizienter Algorithmen profitieren. In nahezu allen modernen Computersystemen existiert eine Speicherhierarchie, die von kleinen, sehr schnell zugreifbaren Caches bis hinunter zu größeren und preiswerteren, aber bezüglich der Zugriffszeit sehr viel langsameren Datenträgern wie Festplatten oder Bandlaufwerken reicht.[5] Ein Algorithmus, der nur wenig zusätzlichen Speicher verwendet, hat das Potential, eine im Mittel größere (Teil-) Menge E an Eingabedaten auf einer konkreten Hierarchiestufe komplett zu bearbeiten. Ein Algorithmus hingegen, der mehr zusätzlichen Speicher benötigt, muss eventuell für die Bearbeitung von E *wiederholt* Teile von E aus Speichermedien der nächst niedrigeren Hierarchiestufe auslesen.[6]

Zudem ist die Untersuchung und Entwicklung speichereffizienter Algorithmen

---

und 7 des Lehrbuches geschildert. Einige dieser Szenarien werden in Kapitel 5 dieser Arbeit näher erläutert.

[3]Ein Überblick über Technik, Logik und Aufgaben von Sensornetzwerken erschließt sich aus den Arbeiten von Akyildiz *et al.* [ASSC02], Mainwaring *et al.* [MCP+02] und Khemapach *et al.* [KDM05]. Auf diese Arbeiten und den Einsatz speichereffizienter Algorithmen in Sensornetzwerken wird in Kapitel 5 noch detaillierter eingegangen.

[4]Die Technik von Sensor-Computern und die Entwicklung ihrer Hardwareressourcen wird in den Arbeiten von Vieira *et al.* [VCdSdM03] und Khemapech *et al.* [KDM05] zusammengefasst.

[5]Einen umfassenden Überblick über moderne Rechnerarchitekturen bieten Patterson und Hennessy [PH02]; Speicherhierarchien und ihre Struktur in verschiedenen Rechnertypen werden in Kapitel 5 des Werkes beschrieben.

[6]Ein Überblick über die Entwicklung von Algorithmen unter der Berücksichtigung von Speicherhierarchien findet sich in einem von Meyer *et al.* [MSS03] herausgegebenen Werk.

im Forschungsgebiet der theoretischen Algorithmik von Interesse, da die Abhängigkeiten zwischen Speicherplatzbedarf und erreichbarer Laufzeitkomplexität für viele Problemstellungen noch nicht zufriedenstellend erforscht sind. In den Kapiteln 5 und 6 wird dies an Problemen der algorithmischen Geometrie verdeutlicht, für die die ersten Algorithmen vorgestellt werden, die bezüglich des im Folgenden beschriebenen Berechnungsmodells sowohl hinsichtlich Laufzeit als auch hinsichtlich Speicherbedarf optimal sind.

Das Berechnungsmodell In dieser Arbeit wird bei der Analyse von Algorithmen hinsichtlich Laufzeit und Speicherbedarf das im Folgenden erläuterte *real random access machine (real RAM)*-Berechnungsmodell [PS85, Knu98a] zu Grunde gelegt. Das *real RAM*-Modell basiert auf dem Modell der *random access machine* [AHU82, Knu98a], die über eine unbegrenzte Anzahl an Speicherzellen verfügt, in denen jeweils eine beliebig große, ganze Zahl gespeichert werden kann. In der Laufzeitanalyse werden für den Zugriff auf eine beliebige Zelle Einheitskosten veranschlagt. Die *real RAM* als Erweiterung der random access machine erlaubt, in einer Zelle (alternativ statt einer ganzen) eine reelle Zahl mit unbeschränkter Genauigkeit zu speichern. Weiterhin ist ein konkretes *RAM*-Modell zusätzlich über die zulässigen Grundoperationen definiert, für deren Anwendung dieselben Einheitskosten wie für einen Speicherzugriff veranschlagt werden. In dieser Arbeit wird als Berechnungsmodell – wie auch in Übersichten zur algorithmischen Geometrie (vergleiche [PS85, dBvKOS97, Lee96]) — das *real RAM*-Modell zu Grunde gelegt, das die Grundoperationen indirekte Adressierung, die binären Vergleichsoperatoren „$<$“, „$=$“ und „$\leq$“ sowie folgende arithmetische Operationen bietet: Addition, Subtraktion, Multiplikation und exakte Division, die Berechnungen von Logarithmen und k-ten Wurzeln, sowie die Auswertung von Exponentialfunktionen. In dieser Arbeit werden Laufzeit und Speicherbedarf von Algorithmen zumeist in Form ihrer asymptotischen Komplexität angegeben; insbesondere wird ein Algorithmus als optimal hinsichtlich Laufzeit oder Speicherbedarf bezeichnet, wenn er diesbezüglich eine optimale asymptotische Komplexität aufweist [CLR01]. Eine Erläuterung des Begriffs der asymptotischen Komplexität sowie eine nähere Motivation und Beschreibung des *real RAM*-Modells, insbesondere für geometrische Algorithmen, findet sich bei Preparata und Shamos [PS85].

Im Folgenden bezeichnet der Begriff *Wort* eine Menge an Speicherzellen, die genügt, um eine durch $\mathcal{O}(\log_2 n)$ binäre Ziffern darstellbare Zahl vorzuhalten, wobei n die Größe der Eingabe des Algorithmus angibt. Somit genügt ein Wort insbesondere, um eine eindeutige Referenz auf ein Element innerhalb der Eingabe vorzuhalten.

Es wird angenommen, dass jedes der Eingabeelemente sich durch eine konstante Anzahl an Wörtern speichern lässt. Der Speicherbedarf eines Algorithmus wird an der Anzahl der Wörter gemessen, die er an *zusätzlichem Speicher*, d. h. an Speicher zusätzlich zum die Eingabe darstellenden Speicher, verwendet.

Ein Algorithmus wird im Folgenden *speichereffizient* genannt, wenn er (in der Eingabegröße) sublinear viele Wörter an zusätzlichem Speicher verwendet. Ein *in-place*-Algorithmus verwendet nur konstant viele Wörter an zusätzlichem Speicher. Für die in dieser Arbeit präsentierten *in-place*-Algorithmen wird davon ausgegangen, dass die Eingabe in Form eines Feldes `A` übergeben wird. Es wird zudem von

Algorithmen die folgende Invariante gefordert: Vor, während und nach Ablauf eines Algorithmus stellt `A` genau die Elemente der Eingabe dar. Es wird keine Annahme darüber gemacht, ob in `A` die tatsächlichen Eingabedaten oder nur Referenzen auf diese Daten gespeichert sind. In vielen praktischen Anwendungen ist von der Übergabe in Form von Referenzen auszugehen, wobei sich die referenzierten Daten auf einem externen Medium ohne Schreibzugriff befinden. Die Zulässigkeit dieser Art der Übergabe impliziert für das gewählte Modell, dass in `A` keine Manipulation einzelner Eingabeelemente, sondern höchstens die Permutation solcher zulässig ist.

Gliederung Die genannten Anwendungsfelder für speichereffiziente Algorithmen verbindet, dass die zu verarbeitenden Daten häufig geometrischer Natur oder zumindest geometrisch interpretierbar sind. Dementsprechend lösen die in jüngerer Zeit veröffentlichten speichereffizienten Algorithmen häufig geometrische Probleme. In Abschnitt 4.1 werden zunächst algorithmische Bausteine, d. h. Algorithmen für grundlegende Problemstellungen wie z.B. Sortierung einer Menge oder effiziente Suche nach Elementen in einer Menge vorgestellt. Anschließend werden in Abschnitt 4.2 Resultate aus dem Gebiet der speichereffizienten geometrischen Algorithmen zusammengefasst.

In den Kapiteln 5 und 6 werden Algorithmen zu Problemstellungen präsentiert, die ebenfalls eine geometrische Interpretation besitzen und für die bislang keine speichereffizienten Lösungen bekannt waren. Zusätzlich wird die Relevanz dieser Problemstellungen vor dem Hintergrund der genannten praktischen Anwendungsfelder für speichereffiziente Algorithmen genauer erläutert. In diesen Algorithmen findet die Mehrzahl der im Folgenden vorgestellten Bausteine Verwendung.

4.1 Bausteine für speichereffiziente Algorithmen

4.1.1 Implizite Kodierung von Informationen

Um speichereffiziente Algorithmen zu entwickeln, kann in vielen Problemstellungen ausgenutzt werden, dass sich zusätzliche Informationen, etwa über den Status der Bearbeitung der Eingabedaten, implizit im Eingabefeld `A` speichern lassen.

Im Folgenden wird eine Technik zur impliziten Kodierung von Informationen vorgestellt und erläutert, wie und mit welcher Effizienz diese Informationen verarbeitet werden können.

Implizite Kodierung durch Permutation Während *in-place*-Algorithmen für das explizite Vorhalten von Informationen nur konstant viel zusätzlichen Speicher verwenden, lässt sich potentiell eine größere Informationsmenge implizit durch Permutation einzelner Eingabeelemente vorhalten, wie von Munro [Mun86] beschrieben: Ein Bit, d. h. eine einzelne binäre Informationseinheit, kann durch die beiden Permutationen zweier unterscheidbarer Elemente p und q ausgedrückt werden, auf denen eine Ordnung „$<$“ definiert (und bekannt) ist: Gilt $p < q$, dann kodiert die Permutation qp eine binäre Eins und die Permutation pq eine binäre Null. Zwei solche im Eingabefeld benachbarten und in dieser Weise zur Kodierung einer binären In-

formation verwendeten Eingabeelemente werden im Folgenden als „*Bit-Nachbarn*“ bezeichnet.

An Hand des folgenden Beispiels ist skizziert, wie sich durch diese Kodierungstechnik zwei Zahlen (im Beispiel die Zahlen 1 und 7) — *zusätzlich* zur Eingabe, aber ohne Verwendung zusätzlichen Speichers — repräsentieren lassen. Jede der beiden Zahlen ist durch jeweils drei Bits darstellbar und wird daher durch Permutation von jeweils 6 kodierenden Eingabeelementen dargestellt. Für ein Auslesen der Kodierung ist erforderlich, dass die der Kodierung zu Grunde liegende Ordnung bekannt ist.

...	8	11	13	15	19	16	...	25	24	29	26	36	32	...
		0		0		1			1		1		1	

Wesentlich für die Eindeutigkeit der kodierten Information ist, dass die verwendete Ordnung *streng* im Sinne der folgenden Definition ist:

Definition 4.1.1 Eine Menge M heißt im Folgenden *streng linear anordbar*, wenn eine transitive Relation $<$ existiert, so dass für je zwei verschiedene Elemente p und q aus M entweder $p < q$ oder $q < p$ gilt. Eine Multimenge M heißt *(schwach) linear anordbar*, wenn eine transitive Relation $\leq$ existiert, so dass für je zwei Elemente p und q aus M $p \leq q$ oder $q \leq p$ gilt.

Für die Menge an implizit durch Permutationen von Eingabeelementen speicherbaren Informationen ergibt sich:

Beobachtung 4.1.2 *Sei als Eingabe in* `A` *eine n-elementige streng anordbare Menge gegeben. Dann können in* `A` *bis zu und nicht mehr als $\lfloor n/2 \rfloor$ Bits implizit durch die beschriebene Technik kodiert werden, ohne dabei Informationen über Eingabeelemente zu verlieren.*

Bemerkung 4.1.3 Die beschriebene Kodierungstechnik kann auch auf Multimengen angewendet werden; in der Eingabe befindliche Duplikate reduzieren allerdings den Umfang der kodierbaren Informationen. Die Beobachtung 4.1.2 ist auch für eine Multimenge als Eingabe gültig, sofern n als die Mächtigkeit der duplikatbereinigten Eingabe verstanden wird.

In den *in-place*-Algorithmen, die in den Kapiteln 5 und 6 vorgestellt werden, sind die zu kodierenden Informationen zumeist Referenzen auf Elemente der Eingabe des Algorithmus. Solche Referenzen sind jeweils als binäre Zahl aus der Menge $1, \ldots, n$ darstellbar. Dementsprechend wird eine Referenz durch jeweils $\lceil 2 \cdot \log_2 n \rceil$ Eingabeelemente, d. h. durch $\lceil \log_2 n \rceil$ binäre Informationen, kodiert. Als Folgerung von Beobachtung 4.1.2 ergibt sich:

Korollar 4.1.4 *In einer n-elementigen streng anordbaren Eingabemenge können durch die beschriebene Kodierungstechnik bis zu $\lfloor n/2 \rfloor / \lceil \log_2 n \rceil$ Referenzen auf Eingabeelemente kodiert werden.*

Für die Korrektheit der Referenzierung von Eingabeelementen (ob durch explizit gespeicherte oder durch implizit kodierte Referenz) ist folgende Konvention einzuhalten: Soll auf ein *kodierendes* Eingabeelement $\mathtt{A}[i]$ verwiesen werden, so hat diese Referenz auf die *korrekte* Position des Eingabeelementes (bezüglich der zur Kodierung verwendeten Ordnung) zu verweisen, obwohl das zu referenzierende Eingabeelement eventuell nicht an der i-ten Position in $\mathtt{A}$, sondern an der des Bit-Nachbarn von $\mathtt{A}[i]$ zu finden ist. Die korrekte Position eines Elementes bezüglich seines Bit-Nachbarn ist als seine Position in der Anordnung der beiden Bit-Nachbarn, so dass diese eine binäre 0 kodieren, definiert. In den in der Folge vorgestellten *in-place*-Algorithmen wird diese Ordnung auf allen Bit-Nachbarn etabliert, bevor diese zur Kodierung von Bits verwendet werden. Diese Vorabordnung ist allerdings nicht zwingend notwendig: Eine Referenz auf ein Element in korrekter Position, dass mit seinem Bit-Nachbarn ein Bit kodiert oder für eine solche Kodierung vorgesehen ist, kann in konstanter Zeit durch Betrachten dieses Elementes und seines Bit-Nachbarn aufgelöst werden: Da zur Kodierung verwendete Bit-Nachbarn im Eingabefeld benachbart sind, genügt ein Vergleich von $\mathtt{A}[i]$ mit seinem Bit-Nachbarn, um deren korrekte Ordnung und damit das referenzierte Eingabeelement zu identifizieren.[7]

Im skizzierten Beispieleingabefeld würde das Auslesen einer Referenz auf die Position der Zelle, die aktuell den Wert 36 trägt, den im Bit-Nachbarn gespeicherten Wert 32 liefern. Entscheidend für Laufzeiteffizienz und Konsistenz ist, dass diese Identifizierung durch nur lokale und nur temporäre Rekonstruktion der Ordnung der Elemente 32 und 36 möglich ist.

Vertauschen kodierter Werte Zwei durch Permutationen kodierte Werte a und b können durch Permutationen ausschließlich von Bit-Nachbarn vertauscht werden [BV06b]: Anstatt den Block der den Wert a kodierenden Elemente mit dem Block der den Wert b kodierenden Elemente zu tauschen, werden zur Kodierung des i-ten Bits von a die zuvor das i-te Bit von b kodierenden Bit-Nachbarn „wiederverwendet“. Nachfolgend ist das bereits skizzierte Beispielfeld jeweils vor und nach der Vertauschung der beiden kodierten Werte 1 und 7 dargestellt:

…	8	11	13	15	19	16	…	25	24	26	29	36	32	…
		0		0		1			1		1		1	

…	11	8	15	13	19	16	…	24	25	29	26	36	32	…
		1		1		1			0		0		1	

Durch diese „lokale“ Realisierung des Vertauschens von kodierten Werten bleibt gesichert, dass eine Referenz auf ein a oder b kodierendes Element $\mathtt{A}[i]$ auch nach der Vertauschung auf $\mathtt{A}[i]$ verweist. Die Vertauschungsoperation für zwei Werte a und b, die durch je $\mathcal{O}(\log_2 n)$ Eingabeelemente kodierbar sind, kann in $\mathcal{O}(\log_2 n)$ Zeit unter Verwendung von konstantem Speicher durchgeführt werden.

[7] Im Folgenden sei der Bit-Nachbar eines Eingabeelementes durch folgende Konvention eindeutig festgelegt: Ein Bit-Nachbarpaar wird jeweils von einem Eingabeelement $\mathtt{A}[2j]$ mit geradem Index, gefolgt von einem Eingabeelement $\mathtt{A}[2j+1]$ mit ungeradem Index, gebildet.

Sortieren von kodierten Werten Durch die oben beschriebene Realisierung der Vertauschung ist eine Sortierung von r kodierten Werten effizient und konsistent möglich. Für das Sortieren von kodierten Werten kann ein *in-place*-Sortieralgorithmus mit einer optimalen Laufzeit von $\mathcal{O}(r \cdot \log_2 r)$ wie der *heapsort*-Algorithmus (vergleiche Kapitel 4.1.2) angepasst werden: Jeder der $\mathcal{O}(r \cdot \log_2 r)$ notwendigen binären Vergleiche erfordert das Dekodieren von mindestens einem, in der Regel von zwei Werten, wenn nur konstanter zusätzlicher Speicher verwendet werden kann. Die Laufzeit eines Dekodiervorgangs ist linear in der Anzahl der für die Kodierung eines Wertes verwendeten Eingabeelemente. Der Bedarf an zusätzlichem Speicher entspricht dem einer Vertauschoperation. Dies führt zu folgendem Resultat:

Korollar 4.1.5 (Korollar 1 in [BV06b]) *Es seien in einer Eingabe* $\mathtt{A}$ *durch Permutationen von jeweils* $\mathcal{O}(\log_2 n)$ *Eingabeelementen* r *Werte kodiert. Dann ist das Sortieren dieser* r *Werte* in-place *in* $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$ *Zeit möglich, ohne dabei die Korrektheit etwaiger Referenzen auf Eingabeelemente zu verletzen.*

Bemerkung 4.1.6 Letzteres Resultat ergibt sich auch dann, wenn die r kodierten Werte Referenzen sind und die relative Ordnung zweier kodierter Referenzen a und b sich aus der Ordnung der referenzierten Eingabeelemente ergibt.

4.1.2 *In-place*-Algorithmen für grundlegende Problemstellungen

Im Folgenden werden einige grundlegende algorithmische Aufgabenstellungen beschrieben, für die *in-place*-Algorithmen mit optimaler asymptotischer Laufzeit existieren und die als algorithmische Bausteine in den innerhalb dieser Arbeit vorgestellten geometrischen Algorithmen eingesetzt werden.

Verschmelzen Seien zwei benachbarte und zusammenhängende Eingabeteile $\mathtt{A}_1$ und $\mathtt{A}_2$ eines Eingabefeldes $\mathtt{A}$ jeweils bezüglich derselben Ordnung $<_{\mathtt{A}}$ sortiert gegeben. Das *Verschmelzen* von $\mathtt{A}_1$ und $\mathtt{A}_2$ bezeichnet das Überführen dieser Teile in einen komplett bezüglich $<_{\mathtt{A}}$ sortierten Eingabeteil $\mathtt{A}_1 \cup \mathtt{A}_2$.

Es sind zahlreiche *in-place*-Verschmelzungsalgorithmen mit linearer Laufzeit vorgestellt worden, darunter der als einfach implementierbar geltende Algorithmus von Mannila und Ukkonen [MU84] sowie Varianten dieses Algorithmus, die von Geffert *et al.* [GKP00] publiziert wurden. Diese Varianten besitzen dieselbe asymptotische Laufzeitkomplxität, benötigen aber eine geringere Anzahl an Umpositionierungen von Eingabeelementen.

Entschmelzen Sei eine bezüglich einer Ordnung $<_{\mathtt{A}}$ sortierte Eingabe $\mathtt{A}$ sowie ein auf die Elemente von $\mathtt{A}$ anwendbares $\{0,1\}$-wertiges Prädikat π gegeben. Das *Entschmelzen* von $\mathtt{A}$ bezüglich π bezeichnet das Partitionieren von $\mathtt{A}$ in zwei Eingabeteile $\mathtt{A}_1$ und $\mathtt{A}_2$ entsprechend der Werte der Elemente von $\mathtt{A}$ bezüglich des Prädikates π. Nach Anwendung des Entschmelzens haben $\mathtt{A}_1$ und $\mathtt{A}_2$ jeweils in $<_{\mathtt{A}}$-Ordnung vorzuliegen. Nach Anwendung eines *stabilen Entschmelzens* müssen $\mathtt{A}_1$ und $\mathtt{A}_2$ zusätzlich

stabil geordnet sein.[8]

Ein stabiler *in-place*-Entschmelzungsalgorithmus mit linearer Laufzeit wird von Salowe und Steiger [SS87a] beschrieben. Dieser Algorithmus setzt voraus, dass für jedes Element $x \in \mathtt{A}$ der Wert $\pi(x)$ unabhängig von anderen Elementen $y \in \mathtt{A}$ und vom Status der Abarbeitung des Algorithmus ist.

Bose *et al.* [BMM$^+$06] stellen einen Algorithmus zur *stabilen Teilmengenextraktion* vor. Dieser *in-place*-Algorithmus mit linearer Laufzeit separiert wie der Algorithmus von Salowe und Steiger zwei Eingabeteile entsprechend eines Prädikates π. Liegt eine Sortierung vor, erhält er die Sortierung jedoch nur auf einem (wählbaren) der beiden Eingabeteile. Dieser Algorithmus ist einfacher implementierbar als der von Salowe und Steiger und setzt nicht voraus, dass die Werte der Eingabeelemente unter π zu Beginn des Algorithmus feststehen; vielmehr können diese von Status und Ablauf des Algorithmus abhängen. Zusätzlich stellen Bose *et al.* einen *in-place*-Algorithmus vor, der den Effekt ihres stabilen Teilmengenextraktionsalgorithmus in linearer Zeit rückgängig macht; dabei braucht das für die vorangegangene Teilmengenextraktion verwendete Prädikat π nicht bekannt zu sein.

Selektion Sei eine Eingabe $\mathtt{A}$ sowie eine Ordnung $<_{\mathtt{A}}$, bezüglich der $\mathtt{A}$ linear anordbar ist, gegeben. Die *Elementselektion*, angewendet auf $\mathtt{A}$ mit Ordnung $<_{\mathtt{A}}$ und einem Parameter k mit $1 \leq k \leq |A|$, liefert das bezüglich $<_{\mathtt{A}}$ k-te Element aus $\mathtt{A}$.[9] Elementselektionsalgorithmen mit linearer und damit optimaler Laufzeit, die nur konstanten Zusatzspeicher verwenden, wurden von Carlsson und Sundström [CS95], Geffert und Kollár [GK01] sowie von Bose *et al.* [BMM$^+$06] vorgestellt. Ein Spezialfall des Elementselektionsproblems ist die Bestimmung des *Medians*, d. h. des „mittleren“ Elementes einer (total anordbaren) Menge.[10] [11]

Sortieren Die (bezüglich des vorgestellten *real RAM*-Modells bzw. bezüglich des zugeordneten Entscheidungsbaummodells) optimale Laufzeitkomplexität für Sortieralgorithmen ist $\mathcal{O}(n \cdot \log_2 n)$ [CLR01]. Es existieren zahlreiche *in-place*-Sortieralgorithmen mit dieser Laufzeitkomplexität. Der bekannteste ihrer Vertreter ist als *heapsort* bekannt, siehe Floyd [Flo64] sowie Williams [Wil64]. Zu dem in der Praxis häufig eingesetzten *quicksort*-Algorithmus von Hoare [Hoa62] ist eine *in-place*-Variante mit asymptotisch optimaler Laufzeit realisierbar.[12] Auch der klas-

[8] Die Bedingung der Stabilität bedeutet hier, dass die Reihenfolge zweier bezüglich $<_{\mathtt{A}}$ nicht unterscheidbarer Elemente in $\mathtt{A}$ gewahrt bleiben muss, sofern sie unter π auf denselben Wert abgebildet werden.

[9] Die Teilmengenselektion ist auch für Eingaben definiert, die bezüglich $<_{\mathtt{A}}$ nicht unterscheidbare Elemente enthält. Entsprechende Duplikate bezüglich $<_{\mathtt{A}}$ werden bei der Elementselektion des k-ten Elementes berücksichtigt und nicht aus der Ordnung ausgeschlossen. Im Falle der Existenz solcher Duplikate ist zu beachten, dass das bezgülich $<_{\mathtt{A}}$ k-te Element nicht eindeutig ist.

[10] Im Folgenden sei der Median $m_{<_{\mathtt{A}}}$ einer Multimenge der Größe n (bezüglich einer vorgegeben Ordnung „$<_{\mathtt{A}}$“) formal als das (bezüglich $<_{\mathtt{A}}$) $\lceil n/2 \rceil$-te Element definiert.

[11] Die Probleme der Elementselektion und der Medianselektion besitzen im *real-RAM-Modell* dieselbe scharfe obere und (triviale) untere Schranke für Laufzeitkomplexität (von $\Theta(n)$) sowie für die Komplexität des Bedarfs an zusätzlichem Speicher (von $\Theta(1)$).

[12] Der *quicksort*-Algorithmus wählt mehrfach ein Pivotelement, mit dem jeweils eine Gruppierung von Eingabeelementen vorgenommen wird. Eine hinsichtlich Laufzeit und Speicherbedarf asymptotisch optimale *quicksort*-Variante erhält man, wenn als Pivotelement jeweils der durch

sische *mergesort*-Algorithmus ist *in-place* und in asymptotisch optimaler Laufzeit zu realisieren (vergleiche Katajainen und Pasanen [KP99] sowie die Beschreibung der *in-place*-Realisierung des rekursiven *mergesort*-Gerüstes in Abschnitt 5.3.2); für die Teiloperation des Verschmelzens ist dazu eine der oben erwähnten *in-place*-Realisierungen mit linearer Laufzeit zu verwenden. Franceschini und Geffert [FG05] stellten jüngst den ersten *in-place*-Sortieralgorithmus mit asymptotisch optimaler Laufzeit und nur linearer, also asymptotisch optimaler, Anzahl an Umpositionierungen von Eingabeelementen vor. Auch dieser Algorithmus basiert auf dem klassischen *mergesort*-Gerüst.

Speichereffiziente Datenstrukturen Die binäre Suche auf einer geordneten Eingabe `A` ist ein effizientes Konzept, das Suchen *in-place* und in logarithmischer Zeit ermöglicht; die im Rahmen dieser Arbeit entwickelten und in den Kapiteln 5 und 6 vorgestellten Algorithmen verwenden diese implizite Datenstruktur, jedoch keine der nachfolgend erläuterten komplexeren Datenstrukturen.

Effiziente *dynamische* Suchstrukturen zu konzipieren, für die also zusätzlich die Operationen des Löschens und Einfügens effizient realisiert sind, ist eine wesentlich größere Herausforderung. Ist zudem ein möglichst geringer Aufwand an zusätzlichem Speicher Zielsetzung, so ist die von Franceschini und Grossi kürzlich vorgestellte Suchstruktur [FG03] (zumindest bezüglich asymptotischer Komplexität) konkurrenzlos: Diese verwendet nur konstant viele Wörter an zusätzlichem Speicher und ermöglicht das Einfügen, Löschen und Suchen in jeweils logarithmischer (und damit optimaler) Zeit.[13] Gleichzeitig ist diese Struktur auch im *cache-oblivious*-Modell[14] optimal, d. h. die Anzahl der Blocktransfers ist (bezüglich jeder denkbaren Speicherhierarchie und auf allen Ebenen dieser Hierarchie) bei jeder Operation in der optimaler asymptotischer Komplexität von $\mathcal{O}(\log_B n)$.[15]

Ist vor allem die effiziente Identifikation nur des maximalen (oder minimalen) Elementes (und nicht so sehr die Suche nach beliebigen Elementen) relevant, so kann ein *heap* (als Realisierung eines *Prioritätswarteschlange* genannten abstrakten Datentyps[16]) verwendet werden: Die *max-heap* (bzw. *min-heap*)-Datenstruktur von Williams [Wil64] liefert *in-place* das maximale (bzw. minimale) Element in konstanter Zeit. Das Aktualisieren der Datenstruktur in Form des Entfernens dieses Elementes oder des Einfügens eines beliebigen Elementes ist in logarithmischer Laufzeit möglich.

Chen und Chan [CM03] präsentierten eine Kombination eines *heap* mit Munros speichereffizientem Suchbaum [Mun86], wobei diese Kombination die Verwendung zweier (verschiedener) Ordnungen, d. h. für Suchbaum und *heap* einer jeweils eigenen

einen laufzeitoptimalen *in-place*-Elementselektionsalgorithmus bestimmte Median gewählt wird.

[13] Ein Überblick über die Entwicklung laufzeit- und speichereffizienter dynamischer eindimensionaler Suchstrukturen findet sich ebenfalls bei Franceschini und Grossi [FG03].

[14] Für Erläuterungen zum *cache-oblivious*-Modell siehe beispielsweise Demaine [Dem02] und dort zu findende Referenzen.

[15] B gibt in dieser Notation die (nicht notwendigerweise dem Algorithmus bekannte) Blockgröße an, d. h. die Anzahl der Elemente, die bei einem einzelnen Zugriff auf das nächst niedrigere Medium der Speicherhierarchie ausgelesen werden.

[16] Eine Erläuterung des Konzeptes der abstrakten Datentypen sowie von Prioritätswarteschlangen findet sich beispielsweise bei Cormen *et al.* [CLR01].

Ordnung, zulässt. Diese Struktur ist speziell für die speicher- und laufzeiteffiziente Realisierung von *plane-sweep*-Algorithmen (vergleiche beispielsweise [PS85, GH93]) konzipiert und verwendet $\Theta(\log_2^2 n)$ Wörter an zusätzlichem Speicher. Sie unterstützt Such- und Einfügeoperationen sowie das Löschen des bezüglich der im *heap* verwendeten Ordnung maximalen Elementes in jeweils $\Theta(\log_2^2 n)$ Zeit.

Rekursive Algorithmen Ein zentrales Problem bei der *in-place*-Realisierung von rekursiven Algorithmen ist, dass im Allgemeinen der Rekursionsstack explizit vorgehalten werden muss. Dieser hat die Funktion, die Rücksprungadressen für jeden rekursiven Aufruf, eventuell zusammen mit weiteren Statusinformationen, zu speichern [CLR01].

Insbesondere eine Verarbeitung nach dem *divide & conquer*-Prinzip (siehe beispielsweise [CLR01, Kapitel 2.3.1]) führt zu Rekursionsstacks von (in der Größe der Eingabe) logarithmischer Größe: Ein *divide & conquer*-Algorithmus partitioniert die Eingabemenge rekursiv, bis die so erhaltenen Teilmengen von handhabbarer (d. h. gewöhnlich konstanter) Größe sind. Anschließend werden diese kleineren Probleminstanzen bearbeitet und nachfolgend die Ergebnisse rekursiv miteinander verknüpft, um eine Lösung des Gesamtproblems zu erhalten. Die Rekursionstiefe eines solchen Algorithmus ist bei balancierter Aufteilung logarithmisch in der Anzahl der Aufteilungen, d. h. im gewöhnlichen Fall logarithmisch in der Größe der Eingabe.

Bose *et al.* [BMM+06] beschreiben, wie sich die Kosten von $\Omega(\log_2 n)$ an zusätzlichem Speicher für den Rekursionsstack vermeiden lassen, so dass nur konstant viel zusätzlicher Speicher notwendig wird. Diese Methode ist für jeden *divide & conquer*-Algorithmus anwendbar, der die Eingabe rekursiv in zwei (oder allgemeiner: konstant viele) zusammenhängende Teile gleicher Größe aufteilt. Die Verwendung der Methode von Bose *et al.* impliziert einen zusätzlichen Zeitaufwand von $\mathcal{O}(n)$.[17]

4.2 Speichereffiziente geometrische Algorithmen

Erst in den letzten Jahren sind über speichereffiziente geometrische Algorithmen mehrere Artikel veröffentlicht worden. Deren Ergebnisse fußen größtenteils auf Adaption bereits vorgestellter, aber wenig speichereffizienter Algorithmen. Für einige fundamentale Probleme konnte nachgewiesen werden, dass sie *in-place* und in asymptotisch optimaler Laufzeitkomplexität lösbar sind. Für weitere Probleme wurden Algorithmen vorgestellt, die in Bezug auf die Laufzeiteffizienz und/oder den Bedarf an zusätzlichem Speicher nur um einen polylogarithmischen Faktor von den optimalen Komplexitäten abweichen.

Liniensegmentschnitte Chen und Chan [CM03] untersuchten das Liniensegmentschnittproblem und präsentierten eine Variante von Bentley und Ottmanns plane-sweep-Algorithmus [BO79], um alle k Schnittpunkte von n übergebenen Liniensegmenten auszugeben. Die Variante von Chen und Chan weist eine Laufzeit von

[17] Die Verwendung der Methode von Bose *et al.* wird allgemein swoei für die konkrete Aufgabenstellung der Bestimmung der Maxima einer Punktmenge genauer in Kapitel 6.2.2 beschrieben.

$\mathcal{O}((n+k)\cdot\log_2^2 n)$ auf und verwendet $\mathcal{O}(\log_2^2 n)$ Wörter an zusätzlichem Speicher.[18] In einem erweiterten Berechnungsmodell, in dem zum einen Divisionen in konstanter Zeit berechenbar sind und zum anderen die unwiderrufliche Zerstörung (d.h. Löschung oder Verfälschung) von Eingabeelementen gestattet ist, kann eine Adaption des Algorithmus als *in-place*-Verfahren und mit einer Laufzeitkomplexität von $\mathcal{O}((n+k)\cdot\log_2 n)$ analysiert werden.

Eine Verbesserung dieser Resultate stellt der später von Vahrenhold [Vah05] präsentierte *in-place*-Algorithmus für das Linienschnittproblem mit einer Laufzeit von nur $\mathcal{O}(n\cdot\log_2^2 n + k)$ dar, der das Verfahren von Balaban [Bal95] modifiziert. Bereits zuvor hatten Bose *et al.* einen *in-place*-Algorithmus vorgestellt, der das Segmentschnittproblem für den Spezialfall orthogonaler (d. h. o.B.d.A. nur horizontaler und vertikaler) Liniensegmente in optimaler Komplexität von $\mathcal{O}(n\cdot\log_2 n + k)$ löst.

Konvexe Hüllen Für das ebenso klassische Problem der Berechnung der konvexen Hülle einer n-elementigen Punktmenge in der Ebene stellten Brönnimann *et al.* [BIK+04] verschiedene speichereffiziente Algorithmen vor. Der effizienteste dieser Algorithmen ist eine *in-place*-Variante von Chans Algorithmus [Cha96], die eine asymptotisch optimale ausgabesensitive Laufzeit von $\mathcal{O}(n\cdot\log_2 h)$ aufweist, wobei h die Anzahl der Punkte auf der konvexen Hülle bezeichnet. Brönnimann *et al.* verwenden des Weiteren in jeder Iteration ihres von Chan adaptierten Algorithmus eine *in-place*-Variante von Grahams Algorithmus [Gra72] zur Berechnung der konvexen Hülle von Teilen der Eingabe. Zudem präsentierten sie — ebenfalls als Adaption des Algorithmus von Graham — ein Verfahren, das *in-place* und in linearer Zeit die konvexe Hülle einer Punktmenge, die in einer lexikographischen Sortierung vorliegt, berechnet. Des Weiteren geben die Autoren an, dass sich die Adaption von Chans Algorithmus in abgewandelter Form auch für die *in-place*-Bestimmung aller Maxima einer planaren Punktmenge in $\mathcal{O}(n\cdot\log_2 n)$ Zeit verwenden lässt.[19]

Brönnimann und Chan [BC04] stellten zudem einen *in-place*-Algorithmus mit linearer Laufzeit zur Berechnung der konvexen Hülle einer einfachen (offenen oder geschlossenen) Polylinie vor, wobei letztere durch eine Liste ihrer Punkte (die entsprechend ihren Nachbarschaftsbeziehungen angeordnet sind) übergeben wird. Hierfür adaptieren die Autoren den Algorithmus von Lee [Lee83], welcher wiederum auf dem Ansatz von Graham [Gra72] basiert.

Für die Berechnung konvexer Hüllen in höherdimensionalen Räumen führen Brönnimann *et al.* [BIK+04] aus, dass ein Algorithmus, der unabhängig voneinander von Chan [Cha96], Clarkson [Cla94] und Ottmann *et al.* [OSS01] vorgestellt wurde, bereits *in-place* arbeitet, sofern zur Lösung der in der Berechnung auftretenden

[18] Das im vorigen Abschnitt vorgestellte Berechnungsmodell für speichereffiziente Algorithmen ist für die genannten Liniensegmentschnittalgorithmen bzw. für deren Speicherbedarfsanalyse in folgender Weise zu erweitern: Neben dem begrenzten Arbeitsspeicher für Berechnungen kann die Ausgabe in Form eines Datenstroms in einen *write-only*-Speicher geschrieben werden, auf dessen Inhalt nicht mehr durch die zu analysierenden Algorithmen zugegriffen werden kann und dessen Größe nicht näher betrachtet wird. Ohne diese Modifikation des Berechnungsmodells ist es offensichtlich nicht möglich, Segmentschnittalgorithmen mit — auch im schlimmsten Falle — subquadratischem Speicherbedarf zu entwickeln, da die Anzahl der Schnittpunkte quadratisch in der Anzahl der betrachteten Liniensegmente sein kann.

[19] Diese Problemstellung wird in Kapitel 6 genauer betrachtet.

linearen Optimierungsprobleme ein speichereffizienter Algorithmus verwendet wird. Die resultierende Gesamtlaufzeit für die Bestimmung (und Ausgabe) der konvexen Hülle ist in $\mathcal{O}(d! \cdot n \cdot h)$. Brönnimann *et al.* [BCC04] präsentierten später einen randomisierten *in-place*-Algorithmus, der zu einer d-dimensionalen Eingabe die Punkte der konvexen Hülle in einer erwarteten Laufzeit von $\mathcal{O}(n^{2-1/\lfloor d/2 \rfloor + \epsilon})$ für ein beliebig kleines positives ϵ bestimmt. Hierzu verwenden die Autoren unter anderem ein von ihnen entwickeltes *in-place*-Lösungsverfahren für spezielle lineare Optimierungsaufgaben.

Im selben Artikel wird auch ein ausgabesensitiver Algorithmus für die Bestimmung (und Ausgabe) der Komponenten der unteren bzw. oberen konvexen Hülle einer dreidimensionalen Eingabe in erwartet $\mathcal{O}(n \cdot \log_2 h)$ Zeit beschrieben, der Chans Verfahren [Cha96] adaptiert. Für die Berechnung der gesamten konvexen Hülle im dreidimensionalen Raum stellten Brönnimann *et al.* [BCC04] einen Algorithmus mit einer Laufzeit von $\mathcal{O}(n \cdot \log_2^3 n)$ und einem Bedarf an zusätzlichen Speicherwörtern von $\mathcal{O}(\log_2 n)$ vor.[20]

Voronoi-Diagramme Die Bestimmung des Voronoi-Diagramms[21] zu einer planaren Punktmenge kann durch Berechnung der konvexen Hülle der auf einen dreidimensionalen Paraboloid gelifteten Eingabepunkte erfolgen, vergleiche de Berg *et al.* [dBvKOS97]. Der letztgenannte Konvexe-Hülle-Algorithmus von Brönnniman *et al.* liefert in der oben angegebenen Laufzeit und unter Verwendung von $\mathcal{O}(\log_2 n)$ zusätzlichen Speicherwörtern das Voronoi-Diagramm einer zweidimensionalen Eingabe.

Nächste-Nachbarn-Probleme Ein Paar mit geringstem Abstand innerhalb einer planaren Punktmenge lässt sich *in-place* und in optimaler Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ mit einem Algorithmus von Bose *et al.* [BMM$^+$06] berechnen, welcher das divide & conquer-Verfahren von Bentley und Shamos [BS76] adaptiert. Bose *et al.* stellten zudem einen randomisierten Algorithmus mit einer erwarteten Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ für die Berechnung des bichromatischen Paares mit geringstem Abstand vor, der logarithmisch viele zusätzliche Speicherwörter verwendet und der ebenfalls das algorithmische Gerüst von Bentley und Shamos modifiziert.[22] Brönnimann *et al.* präsentierten für dieses Problem einen weniger effizienten plane-sweep-Algorithmus mit $\mathcal{O}(n \cdot \log_2 n)$ Laufzeit und einem Speicherbedarf von $\mathcal{O}(\log_2^2 n)$ zusätzlichen Wörtern. Für diesen stellten sie jedoch eine Erweiterung vor, die — in denselben Komplexitäten — zu jedem Punkt einer planaren Eingabemenge einen nächsten (je nach Aufgabenstellung monochromatischen oder bichromatischen) Nachbarn berechnet (und in einen Ausgabestrom schreibt).

[20] Dieser Algorithmus separiert im übergebenen Feld `A` die Punkte der konvexen Hülle von den übrigen Eingabepunkten. Zusätzlich (und in derselben Laufzeitkomplexität) können die Kanten und Facetten der konvexen Hülle in einen Ausgabestrom geschrieben werden, sofern ein erweitertes Berechnungsmodell wie das für Liniensegmentschnittalgorithmen angegebene (für die Laufzeitanalyse) zu Grunde gelegt wird.

[21] Einen Überblick verschiedener Definitionen von Voronoi-Diagrammen und effizienter Algorithmen für ihre Berechnung geben Aurenhammer und Klein [AK00].

[22] Beim Problem der Bestimmung des bichromatischen Paares mit geringstem Abstand werden zwei (verschiedenfarbige) Eingaben von Punkten übergeben; es sind nur die Abstände von Paaren von Punkten zu betrachten, welche aus unterschiedlichen Mengen stammen.

Bereichsabfragen Brönnimann *et al.* [BCC04] untersuchten verschiedene Typen von Partitionsbäumen, die sich als Datenstrukturen für Bereichsabfragen verwenden lassen. Sie führten aus, dass ein Partitionsbaum implizit, d.h. *in-place*, vorgehalten werden kann. Die Erstellung dieser impliziten Datenstruktur nimmt für eine n-elementige Punktmenge $\mathcal{O}(n \log_2 n)$ Zeit in Anspruch. Ist die Eingabe einmal entsprechend arrangiert, lassen sich die k Eingabepunkte, die sich in einem beliebigen Anfragesimplex befinden, *in-place* in $\mathcal{O}(n^{1-1/d+\epsilon} + k)$ Zeit bestimmen. Auch Halbraumfragen lassen sich effizient in $\mathcal{O}(n^{1-1/\lfloor d/2 \rfloor + \epsilon})$ Zeit beantworten.

Diese implizite Datenstruktur — bzw. verschiedene Varianten dieser — können nach Brönnimann *et al.* modifiziert werden, um das Einfügen weiterer Punkte zu unterstützen. Diese Funktionalität bzw. die hierfür notwendigen Modifikationen schlagen sich in einem weiteren logarithmischen Faktor in der Anfragezeit nieder.

Ausdehnungsprobleme Chen und Chan [CM05] betrachteten als erste das Kleesche Maßproblem mit der Zielsetzung, es speichereffizient zu lösen. Die Eingabe besteht bei diesem Problem aus n achsenparallelen Rechtecken, für deren Vereinigung der Flächeninhalt zu bestimmen ist. Die Autoren stellten einen Algorithmus mit einer Laufzeit von $\mathcal{O}(n^{3/2} \cdot \log_2 n)$ vor, der $\mathcal{O}(\sqrt{n})$ zusätzliche Speicherwörter verwendet. Laufzeit und Speicherbedarf reduzieren sich auf $\mathcal{O}(n \cdot \log_2^2 n)$ bzw. auf $\mathcal{O}(\log_2^2 n)$, wenn die unwiderrufliche Zerstörung der Eingabe zugelassen werden kann. Zusätzlich präsentierten Chen und Chan ein Verfahren, das — in $\mathcal{O}(n \cdot \log_2^3 n)$ Zeit und mit $\mathcal{O}(\log_2^2 n)$ zusätzlichen Speicherwörtern — zu einer Eingabe von n Punkten dasjenige Einheitsquadrat (bzw. seine Lage) bestimmt, das die maximale Anzahl an Eingabepunkten überdeckt.

Vahrenhold [Vah06] beschrieb einen mit einer Laufzeit von $\mathcal{O}(n^{3/2} \cdot \log_2 n)$ effizienteren Algorithmus für das Kleesche Maßproblem, der zudem eine nur konstante Anzahl an zusätzlichen Speicherwörtern verwendet. Weiterhin präsentierte Vahrenhold einen *in-place*-Algorithmus für die eindimensionale Variante dieses Problems, dessen Laufzeitkomplexität mit $\mathcal{O}(n \cdot \log_2 n)$ optimal ist.

Kapitel 5

Speichereffiziente Berechnung von Geradenschätzern

In diesem Kapitel wird die algorithmische Herausforderung, speichereffizient Daten zu aggregieren, zunächst allgemein beschrieben und die Relevanz speichereffizienter Aggregation in praktischen Szenarien erläutert. Die häufig verwendete geometrische Aggregationsoperation des Bestimmens einer Ausgleichsgeraden zu einer Punktmenge durch einen Geradenschätzer wird näher motiviert und nachfolgend werden im Rahmen dieser Arbeit entwickelte *in-place*-Algorithmen zur Berechnung von robusten Geradenschätzern vorgestellt [BV06b].

5.1 Speichereffiziente Aggregation und Einordnung der Geradenschätzer-Aufgabe

Die Aggregation von Daten ist seit jeher eine zentrale Aufgabenstellung in vielen Anwendungssystemen: Jedes praxistaugliche Datenbanksystem bietet zumindest die Aggregationsoperationen des Zählens von Einträgen (jeweils mit und ohne Vielfachheiten) sowie der Bildung von Summe, Maximum, Minimum und des Durchschnitts (jeweils für eindimensionale numerische Wertebereiche) an.[1] Diese Operationen sind algorithmisch einfach zu realisieren — auch dann, wenn nur konstant viel zusätzlicher Speicher zur Verfügung steht: Ihre Ergebnisse sind *in-place* und in (in der Größe der zu aggregierenden Eingabe) linearer und damit optimaler Zeit berechenbar. Letzteres gilt sogar für die komplexere Berechnung des Medians einer Multimenge (bzw. für die Elementselektion, vergleiche Abschnitt 4.1.2).

Wie bereits in Kapitel 4 skizziert, sind speichereffiziente Aggregationsalgorithmen insbesondere in den Anwendungsfeldern der lokationsbasierten Dienste und vor allem der Sensornetzwerke von Nutzen: In Sensornetzwerken ist die Aggregation von gewonnenen Daten vor deren Propagierung ratsam, um hohe Sende- und damit Energiekosten zu vermeiden (siehe beispielsweise [LHY+04, DNW05]); Aggregationen sollten daher lokal, d. h. bereits von den Sensor-Computern ausgeführt werden, wobei deren begrenzte Speicherressourcen zu berücksichtigen sind [SS04]. Anschlie-

[1] Einen Überblick über Technik, Funktionsumfang und Historie von Datenbanksystemen bieten unter anderem die Lehrbücher von Elmasri und Navathe [EN03] und Silberschatz *et al.* [SKS05].

ßend werden nur die Resultate der Aggregationen energiesparend zur zentralen oder dezentralen Protokollierung oder Weiterverarbeitung gesendet. Eines der Anwendungsfelder dieser Strategie ist die Sammlung und Propagierung von Bewegungsdaten. Entsprechende Anwendungungsszenarien lassen sich danach unterteilen, ob Bewegungen durch stationäre Sensoren oder durch an spezifischen sich bewegenden Objekten (oder Subjekten) angebrachte Sensoren überwacht werden. In Szenarien ersteren Typs ist, bedingt durch die Vielzahl an gewonnenen und zu integrierenden Informationen, die Aggregation wesentlicher Bestandteil der Datenverarbeitung.[2] In Szenarien letzteren Typs ist die gemessene Bewegung die der jeweiligen Sensoren [JOW+02, KV03, MCP+02], und ein wesentlicher Anwendungsbereich ist die Beobachtung von Tieren (bzw. Tierherden) und deren Verhalten [MCP+02, JOW+02]. Auch für diese Szenarien wird die Aggregation durch Sensor-Computer aus den bereits genannten Gründen praktiziert [SOP+04].

Auch im Bereich lokationsbasierter Dienste werden (über für diese Dienste taugliche Geräte wie Mobiltelefone oder PDAs) Informationen kommuniziert, deren Sendung kostspielig ist.[3] Zudem erfordern diese Dienste zuweilen, dass größere Datenmengen auf ein mobiles Endgerät eines konkreten Nutzers übertragen werden, diese aber anschließend nach persönlichen Präferenzen selektiert oder visuell aufbereitet werden können. Dies erfordert wiederum Datenaggregationen und -selektionen auf dem — eventuell stark speicherbeschränkten — mobilen Endgerät.[4] Als Beispiele seien für einen konkreten Nutzer die Ermittlung der nächsten erreichbaren Nutzer (oder Dienstleister, wie etwa Taxifahrer) sowie die prioritätsgesteuerte Selektion von Geo- oder Metainformationen zu Kartenausschnitten genannt, für weitere Anwendungsbeispiele siehe [HTKR05, Kapitel 5.5 und Kapitel 7.4]).

[2] Aslam *et al.* [ABC+03] beschreiben die Bewegungsmessung und -analyse von Objekten durch Sensoren, die nur ein Nähern oder Entfernen von Objekten registrieren, Anzahl, Richtung und Entfernung von sich bewegenden Objekten aber nicht eigenständig beurteilen können. Für die Analyse dieser Sensormessungen werden Aggregationsoperationen benutzt und auch das Komprimieren von Sensordaten vor deren Sendung wird vorgeschlagen. Duckham *et al.* [DNW05] beschreiben die Überwachung der Bewegungen von räumlich ausgedehnten Phänomenen wie Wettergebieten. Für die Beantwortung von Anfragen werden Informationen regionsbasiert und nur durch miteinander kommunizierende Sensor-Computer aggregiert. Cerpa *et al.* [CEH+01] fassen verschiedene Designziele für die Überwachung von Biotopen durch Sensornetzwerke zusammen und stellen insbesondere ein Modell zur Beobachtung von Bewegungen durch stationäre Sensoren vor. Die Notwendigkeit von — auf Sensor-Computern mit beschränkten Ressourcen durchführbarer — Aggregation wird herausgestellt.

[3] Im Anwendungsfeld lokationsbasierter Dienste beziehen sich die Kosten — wie für Sensornetzwerke — auf den Energieverbrauch, aber auch auf Kriterien wie die im verwendeten Netz zur Verfügung stehende Bandbreite sowie die monetären Kosten für den Nutzer, vergleiche Höpfner *et al.* [HTKR05].

[4] Die Aggregation und Selektion sollte zur Verringerung der Sendekosten bei mobilen Diensten idealerweise auf Serverseite durchgeführt werden. Viele mobile Dienste unterstützen diese Funktionalitäten allerdings nicht — auch aus nicht technisch bedingten Gründen: So senden viele Dienste beispielsweise Werbung, die allerdings durch auf dem Endgerät installierte Software herauszufiltern ist. Zudem kann — etwa mit der Bewegung des Nutzers oder der Änderung seiner Interessen — die Wiederverwendung und Neubewertung der an ein mobiles Gerät gesendeten Daten ratsam bzw. notwendig sein, etwa bei temporärer Nichtverfügbarkeit des Dienstes (vergleiche jeweils Höpfner *et al.* [HTKR05, Kapitel 5 und 7]).

Das Geradenschätzer-Problem Die genannten Szenarien und referenzierten Arbeiten legen nahe, dass sowohl für Sensornetzwerke als auch für mobile Dienste insbesondere die speichereffiziente Aggregation geometrischer oder geometrisch interpretierbarer (etwa zeitvarianter) Daten relevant ist. Geradenschätzer (*engl.: line estimators*) dienen der Aggregation räumlicher, räumlich-temporaler oder generell mehrdimensionaler Daten. Sie berechnen zu einer Menge $\mathcal{P}$ von Punkten $p_1, \ldots, p_n$ eine Ausgleichsgerade, d. h. sie schätzen die Steigung (oder allgemeiner: die Gestalt) einer Geraden, auf der die Punkte von $\mathcal{P}$ — von (etwa durch Messfehler verursachten) Ungenauigkeiten abstrahierend — vermutet werden. Es sei als Geradenschätzer fortan ein konkretes Verfahren bezeichnet, das festgelegt, auf welche Weise die resultierende Gerade aus der Eingabe bestimmt wird.

Geradenschätzer finden in vielen Bereichen der statistischen Analyse — in Anwendungsfeldern der Natur-, Wirtschafts- und Sozialwissenschaften — Verwendung [Val01, Bob01]. Die Aufgabe des Geradenschätzens für eine Menge $\mathcal{P}$ mehrdimensionaler Daten $\{p_i = (x_{i,1}, \ldots, x_{i,d})\}_{i=1,\ldots,n}$ wird dabei zumeist als lineare Regressionsanalyse bezeichnet (siehe für eine Einführung Rousseeuw und Leroy [RL87]). In dieser wird versucht, durch Wahl der Gestalt einer *Ausgleichsgeraden* (durch Wahl von Parametern $\Theta_1, \ldots, \Theta_d$) die Fehlerwerte $e_1, \ldots, e_n$ in folgendem Gleichungssystem geeignet zu minimieren:

$$y_i = \Theta_0 + x_{i,1} \cdot \Theta_1 + \ldots + x_{i,d} \cdot \Theta_d \; + \; e_i \quad (i = 1, \ldots, n) \tag{5.1}$$

Die Parameter Θ_0 bis Θ_d bestimmen eine Gerade im d-dimensionalen Raum eindeutig und sie ergeben sich (allerdings nicht zwangsweise eindeutig) aus dem Maß, bezüglich dessen die Gesamtheit der Fehler $e_1, \ldots e_n$ im Gleichungssystem (5.1) minimiert werden soll. Die Werte y_i werden als *Antwortvariablen* bezeichnet. Der Fokus dieses Kapitels liegt auf der einfachen linearen Regressionsanalyse, einem Spezialfall der obigen Formulierung des Problems für $d = 2$:

$$y_i = \Theta_0 \; + \; x_{i,1} \cdot \Theta_1 \; + \; e_i \quad (i = 1, \ldots n) \tag{5.2}$$

Der ermittelte Wert von Θ_1 entspricht im Gleichungssystem (5.2) der geschätzten Steigung einer Geraden in der Ebene. Die zusätzliche Festlegung von Θ_2 legt den y-Achsenabschnitt der Geraden fest. Im Idealfall liegen die vorgegebenen Punkte $(x_1, y_1), \ldots, (x_n, y_n)$ auf dieser Geraden (so dass $e_i = 0$ für $i = 1, \ldots, n$ gilt).[5]

Geradenschätzer besitzen auch rein geometrische Anwendungen: Sie werden in der Computervision und -grafik zur Transformation von pixelbasierten Bildern in eine vektorbasierte Darstellung sowie in der automatisierten Bildanalyse zur Erkennung insbesondere von Horizonten oder Raummerkmalen wie Türen oder Fenstern eingesetzt [KN89, EHM04]. Im Kontext von Sensornetzwerken kann ein Geradenschätzer insbesondere zur Simplifizierung von (als linear vermuteten) beobachteten Bewegungen oder von anderen sich in der Zeit entwickelnden Größen verwendet werden. Eine solche Komprimierung zeitlicher Verläufe wird von Lazaridis

[5] Die *multiple* lineare Regressionsanalyse behandelt den allgemeinen, in Gleichung (5.1) dargestellten Fall. *Multivariat* werden Regressionsanalysen genannt, in denen zu Eingabepunkten p_i statt jeweils einer Antwortvariablen y_i wie in obigem Gleichungssystem mehrere Antwortvariablen $y_{i,1}, \ldots, y_{i,r}$ in der Fehlerminimierung zu berücksichtigen sind (siehe [RL87]).

und Mehrotra [LM03] und Lazaridis *et al.* [LHY+04] als eine für den Einsatz in Sensornetzwerken zentrale Aggregationsoperation angesehen. Zusätzlich stellen die genannten Arbeiten die Relevanz heraus, die Qualität der durch Sensor-Computer gewonnenen Daten — trotz einer notwendigen Komprimierung — (gemäß einem gewissen Standard) zu bewahren. Der folgende Absatz motiviert unter anderem, warum gerade die robusten Geradenschätzer, zu deren Berechnung in dieser Arbeit speichereffiziente Algorithmen vorgestellt werden, einen Beitrag zu dieser Qualitätssicherung leisten können.

Zur Robustheit eines Geradenschätzers Ein wesentliches Kriterium für die Qualität eines Geradenschätzers ist seine Robustheit. Eine geringe Robustheit bedeutet im Kontext von Geradenschätzern, dass nur wenige "Ausreißer" unter den zu aggregierenden Punktdaten genügen, um das Ergebnis des Schätzers signifikant zu verfälschen. Das allgemein gebräuchliche Maß für die Robustheit eines Schätzers ist sein *breakdown value*. Dieser Wert gibt eine untere Schranke für den prozentualen Anteil an Ausreißern unter den zu aggregierenden Daten an, der genügt, um das Ergebnis des Schätzers beliebig zu verfälschen.[6]

Ein einfach zu berechnender, aber nicht robuster Geradenschätzer ist der (lineare) *least squares estimator*. Dieser berechnet (als Lösung eines linearen Gleichungssystems) zu Punkten $p_1, \ldots, p_n$ eine Gerade, für die die Summe über die quadratischen Entfernungen zu den Eingabepunkten minimiert wird. Da ein einziger Ausreißer die so geschätzte Gerade beliebig verfälschen kann, besitzt dieser Schätzer ein *breakdown value* von 0 %. Der gegenüber solchen einfacheren Schätzern größere Berechnungsaufwand für robuste Schätzer wird in vielen Anwendungsszenarien akzeptiert (siehe etwa Matoušek *et al.* [MMN98] und dort enthaltene Referenzen).

Zur algorithmischen Komplexität des Geradenschätzens Auch für geometrische Daten sind viele Standard-Aggregationsoperatoren (im *real RAM*-Modell) in linearer Zeit *in-place* berechenbar, etwa der Schwerpunkt einer Menge von Punkten beliebiger Dimension.

Aus Sicht der Erforschung und Entwicklung speichereffizienter Algorithmen ist unter den Aufgabenstellungen zur Datenaggregation die Berechnung robuster Schätzer aus folgendem Grund von besonderem Interesse: Die Robustheit eines Schätzers wird durch Verarbeitung *globaler* statistischer Informationen über alle Eingabedaten (sowie über ihre relative Lage) gesichert; die Berechnung, Speicherung und Verarbeitung dieser statistischen Informationen in asymptotisch geringer Laufzeit bei gleichzeitiger Verwendung von nur konstantem zusätzlichem Speicher ist daher ein herausforderndes und — für alle in dieser Arbeit betrachteten robusten Geradenschätzer

[6] Formal kann der Begriff des *breakdown value* wie folgt definiert werden (vergleiche [RL87, Kapitel 2]): Eine Eingabe $\mathcal{P}$ bestehe aus n Punkten. Bezeichne $\mathcal{P}'_m$ eine Menge, die einer Eingabe $\mathcal{P}$ entspricht, in der m ihrer Punkte durch beliebig zu wählende Punkte ersetzt sind. Definiere für einen Schätzer T den *breakdown value* $b_{||\cdot||}(m; \mathcal{P}, T) = \sup\{||T(\mathcal{P}'_m) - T(\mathcal{P})|| \ : \mathcal{P}'_m\}$, wobei $||\cdot||$ eine Metrik ist. Der *breakdown value* ε^* des Schätzers T bezüglich der n-elementigen Eingabe $\mathcal{P}$ ist durch $\varepsilon^*(T, \mathcal{P}) = \min\{m/n \colon b_{||\cdot||}(m; T, \mathcal{P}) = \infty\}$ definiert, wobei die für die Definition von $b_{||\cdot||}$ verwendete Metrik $_{||\cdot||}$ die euklidische Metrik ist. Der *breakdown value* eines Schätzers T ist dann als der Grenzwert $\lim_{n\to\infty} \{\varepsilon^*(T, \mathcal{P}) \ : \mathcal{P} \subset \mathbb{R}^2, |\mathcal{P}| = n\}$ definiert.

— nicht triviales algorithmisches Problem. Rousseeuw und Leroy formulieren, dass „es tieferliegende Gründe dafür zu geben scheint, dass möglichst robuste Schätzer für die Regressionsanalyse sich nicht billig berechnen lassen“ [RL87, Seite 29].[7]

Insbesondere ist für die im Folgenden beschriebenen Varianten der Aufgabenstellung, eine Gerade robust zu schätzen, eine superlineare untere Schranke innerhalb des *real RAM*-Modells nachweisbar. Die dargestellten Varianten nehmen eine zentrale Stellung in Überblicken zur Regressionsanalyse wie dem von Rousseeuw und Leroy [RL87] ein.

Theil-Sen-Schätzer Theil [The50] entwickelte einen Schätzer, für den Sen [Sen68] folgende robustere Variante definierte: Der Theil-Sen-Schätzer bestimmt für eine n-elementige Menge zweidimensionaler Punkte die mediane Steigung unter den Steigungen aller $\binom{n}{2}$ Geraden, die jeweils durch zwei Punkte der Eingabe definiert werden.[8] Die Definition dieses Schätzers lässt sich auf Eingaben einer beliebigen Dimension d erweitern, in denen nun die Steigungen der $\binom{n}{d}$ Hyperebenen zu betrachten sind. Der *breakdown value* des Theil-Sen-Schätzers beträgt $1 - (1/2)^{1/d}$, für zweidimensionale Eingaben also etwa 29.3.% [MN93].

Repeated Median(RM)-Schätzer Der von Siegel entwickelte *repeated median*-Geradenschätzer [Sie82] ist hinsichtlich seiner Robustheit eine Optimierung des Theil-Sen-Schätzers. Er bestimmt eine Gerade zu einer Menge von Punkten in der Ebene wie folgt: Für jeden Punkt $p_i = (x_i, y_i)$ der Eingabe $\mathcal{P}$ bezeichne m_i die mediane Steigung unter den Steigungen s_j der $n - 1$ Geraden, die durch p_i und einen weiteren Punkt aus $\mathcal{P} \backslash \{p_i\}$ festgelegt werden. Der von Matoušek *et al.* so genannte *repeated median slope* [MMN98] s ist als der Median in der Multimenge der n so bestimmten medianen Steigungen m_i definiert.[9] Der *breakdown value* des *repeated median*-Geradenschätzers ist mit 50 % optimal [RL87, Kapitel 2.7].[10]

Least Squares of Median(LMS)-Schätzer Rousseeuw [Rou84] entwickelte den LMS-Geradenchätzer als robustere Alternative zur *least squares*-Methode. Der LMS-Schätzer bestimmt eine Gerade, für die der Median (statt der Summe wie bei der *least squares*-Methode) der quadratischen Abweichungen zu den Eingabepunkten minimiert wird. Sein *breakdown value* beträgt 50 % [RL87, Kapitel 3.4].

---

[7] Der Aussage von Rousseeuw und Leroy im orignalen Wortlaut: „It appears that there are deep reasons why high-breakdown regression cannot be computed cheaply. “

[8] Für die Wahl des y-Abschnittes einer zu schätzenden Geraden, d. h. des zweiten Parameters, der diese Gerade eindeutig bestimmt, existieren verschieden Vorschläge, die sich mit dem Theil-Sen-Schätzer kombinieren lassen [RL87].

[9] Die berechneten medianen Steigungen können zusätzlich zur Schätzung der Lage der Geraden herangezogen werden: Der so benannte *reapeated median intercept* [MMN98] c ist als der Median der Multimenge $(y_i - s_i \cdot x_i)_{i=1,\ldots,n}$ definiert und kann daher — als zweiter Parameter der zu schätzenden Geraden — durch linearen zusätzlichen Zeitaufwand berechnet werden. Die resultierende und *reapeated median line* [MMN98] benannte Gerade g ist durch $g(x) := s \cdot x + c$ gegeben.

[10] Auch die Definition des *repeated median*-Geradenschätzers lässt sich für höherdimensionale Eingaben verallgemeinern; auch für solche Eingaben beträgt sein *breakdown value* 50 %, vergleiche Mount und Netanyahu [MN93].

Zur Berechnung des LMS-Schätzers für zweidimensionale Eingaben präsentierten Souvaine und Steele einen Algorithmus [SS87b] mit einer Laufzeit von $\mathcal{O}(n^2)$ und Steele und Steiger [SS86] stellten einen effizienten randomisierten Algorithmus vor; letzterer besitzt eine erwartete Laufzeit von $\mathcal{O}(n \cdot \log_2^2 n)$ — allerdings nur gemittelt über zweidimensionale Eingaben, die einer geeigneten Zufallsverteilung entsprechen. Für Eingaben einer beliebigen Dimension d präsentierten Erickson *et al.* [EHM04] einen Algorithmus mit einer Laufzeit von $\mathcal{O}(n^d \cdot \log_2 n)$. Zudem wiesen sie für die Problemstellung eine untere Schranke von $\Omega(n^d)$ im *real RAM*-Modell nach.

Abbildung 5.1 skizziert die Resultate der verschiedenen Geradenschätzer für ein Beispiel. [11] Im linken Teil ist die Bestimmung durch den Theil-Sen-Schätzer dargestellt: Die $\binom{n}{2}$ durch die Eingabe induzierten Geraden werden nach ihrer Steigung aufgezählt; die Gerade mit der zu bestimmenden medianen Steigung ist rot markiert. Im mittleren Teil der Abbildung ist zu jedem Punkt der Eingabe die Bestimmung der (unterbrochen gezeichneten) medianen Steigungen der Geraden, die durch diesen und einen weiteren Eingabepunkt laufen, dargestellt. Für den *repeated median*, also den Median dieser medianen Steigungen, ist die zugehörige Gerade in roter Farbe dargestellt.[12] Im rechten Teil der Abbildung ist das Ergebnis der Anwendung des LMS-Schätzers dargestellt: Die rot markierte Gerade minimiert den Median der quadratischen Entfernungen zu den vier Eingabepunkten. Die entsprechende mediane quadratische Entfernung ist als rot gefärbtes Lot angedeutet.

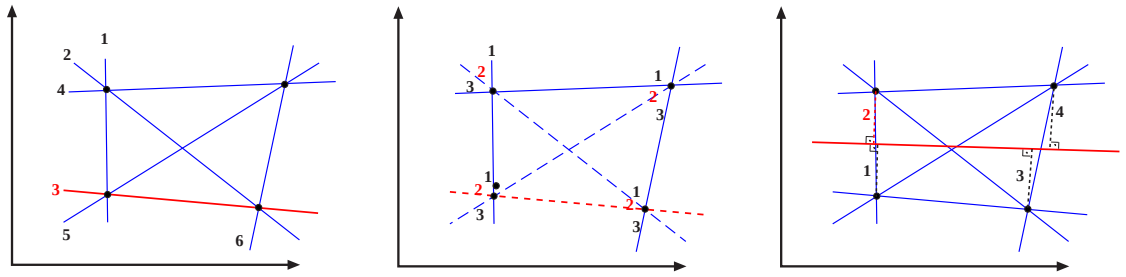


Abbildung 5.1: Skizzen zur Schätzung einer Steigung durch den Theil-Sen- und den *repeated median*-Schätzer sowie zur Schätzung einer Geraden durch den LMS-Schätzer.

In diesem Kapitel werden randomisierte *in-place*-Algorithmen mit erwartet optimaler Laufzeit zur Berechnung des Theil-Sen- bzw. des RM-Schätzers jeweils für zweidimensionale Eingaben vorgestellt.

[11] In den folgenden Ausführungen wird als Ordnung, bezüglich der die mediane (bzw. allgemeiner: die k-te) Steigung zu bestimmen ist, wie in der Literatur üblich, die Ordnung nach dem gegen den Uhrzeigersinn gemessenen Winkel mit der x-Achse verwendet, vergleiche [MN91]. Diese Festlegung ist konsistent mit der Formulierung der Gleichung 5.2.

[12] Zu beachten ist, dass die feiner gestrichelt dargestellte Gerade die mediane Steigung in *zwei* Punkten der Eingabe besitzt. Da der wiederholte Median als der Median über die *Multimenge* der medianen Geraden gebildet wird, muss diese Gerade bei der Bestimmung des Medians der medianen Steigungen zweifach berücksichtigt werden.

5.2 Suche durch randomisierte Interpolation

Der nachfolgend vorgestellte randomisierte *in-place*-Algorithmus zur Berechnung des Theil-Sen-Schätzers bzw. zur Bestimmung einer Geraden medianer Steigung nutzt eine Suchtechnik, die unabhängig voneinander sowohl von Matoušek [Mat91b] als auch von Dillencourt *et al.* [DMN92] vorgestellt wurde. Diese Suchtechnik benennt Matoušek als „randomisierte Suche durch Interpolation“ (*engl.: randomized interpolation search*). Matoušek skizziert die Anwendung dieser Technik auch für weitere Problemstellungen, für die er durch praktische und theoretische Evaluation zeigt, dass die randomisierte Suche durch Interpolation (in den betrachteten Fällen) Effizienzvorteile gegenüber einfacheren randomisierten Suchkonzepten bietet. Zudem ist die randomisierte Suche durch Interpolation — anscheinend im Gegensatz zu vergleichbaren deterministischen Suchverfahren — für viele Problemstellungen auch *in-place* realisierbar.

Diese randomisierte Suchtechnik kann daher als Basis für *in-place*-Lösungen von Suchproblemen verwendet werden, für die bislang keine laufzeit- und zugleich speichereffizienten Algorithmen publiziert wurden. Belegt wird dies in dieser Arbeit durch Vorstellung solcher Algorithmen für die Berechnung des Theil-Sen- und des *repeated median*-Geradenschätzers. Diese Technik wird zunächst losgelöst von der *slope selection*-Problemstellung beschrieben und die Realisierbarkeit einer allgemeinen speichereffizienten Variante überprüft.

5.2.1 Das Konzept der randomisierten Suche durch Interpolation

Betrachtet wird die folgende allgemein gefasste Aufgabenstellung: Es ist ein Element $\mathcal{K}$ in einer großen Kandidatenmenge an Hand seines Ranges k bezüglich einer auf der Kandidatenmenge definierten Ordnung $<_{\mathcal{K}}$ zu identifizieren. Für die Beantwortung von Aufgabenstellungen dieser Art existieren Techniken, die eine effizientere Alternative zur expliziten Betrachtung sämtlicher Kandidaten darstellen: Dies sind zum einen die von Meggido entwickelte deterministische Technik der parametrischen Suche [Meg83], zum anderen einfachere randomisierte Alternativen, insbesondere die von Matoušek [Mat91b] beschriebene Suche durch randomisierte Interpolation. Die dieser Suchtechnik zu Grunde liegende Strategie besteht darin, eine Suchdomäne $\mathcal{S}$, die anfänglich alle Kandidaten umfasst, iterativ durch Verwendung von Interpolation (und deren Überprüfung) zu verkleinern: In jeder Iteration wird durch Betrachtung einer Menge $\mathcal{R}$ von zufällig ausgewählten Kandidaten aus $\mathcal{S}$ die Position des zu identifizierenden Elementes $\mathcal{K}$ interpoliert. Zusätzlich wird eine verkleinerte Suchdomäne $\mathcal{S}'$, die mit hoher Wahrscheinlichkeit $\mathcal{K}$ enthält und um die vermutete Position von $\mathcal{K}$ gewissermaßen „zentriert“ ist, bestimmt. Da die Position von $\mathcal{K}$ nur vermutet wird, bedarf es einer Überprüfung, ob $\mathcal{K}$ sich tatsächlich in $\mathcal{S}'$ befindet. Ist dies der Fall, so wird mit $\mathcal{S}'$ weiter iteriert, andernfalls erneut mit $\mathcal{S}$. Die Iteration endet spätestens, wenn $\mathcal{S}$ nur noch einen Kandidaten für das zu identifizierende Element enthält. Die Iteration kann alternativ abgebrochen werden, sobald $\mathcal{S}$ bzw. die verbliebene Anzahl $|\mathcal{S}|$ an Kandidaten soweit verkleinert ist, dass die Suche nach $\mathcal{K}$ effizient in deterministischer Weise (d. h. ohne weitere Verwendung von Interpolationen) beendet

werden kann. Das Gerüst für eine Suche durch randomisierte Interpolation ist — abstrahiert von einer konkreten Problemstellung — in Algorithmus 5.1 skizziert.

Algorithmus 5.1 Algorithmisches Gerüst zur Suche durch randomisierte Interpolation[Mat91b].

```
1: Initialisiere die Suchdomäne S.          /* Erste „Schätzung“ enthält alle
   Kandidaten. */
2: repeat
3:    Generiere eine zufällige Auswahl R von Kandidaten aus S.
4:    Konstruiere eine verkleinerte Suchdomäne S' durch „Interpolieren“ der Lage
      von K an Hand der zufälligen Auswahl R.
5:    if S' enthält K then
6:       Iteriere mit neuer Suchdomäne S'.
7:    else
8:       Iteriere mit alter Suchdomäne S.
9:    end if
10: until (Suchdomäne hinreichend verkleinert)
11: Beende die Suche nach K in deterministischer Weise.
```

5.2.2 Laufzeiteffizienz der Suche durch randomisierte Interpolation

Die Effizienz der randomisierten Suche durch Interpolation ist unter anderem abhängig von der Effizienz der Überprüfung, ob die verkleinerte Suchdomäne das zu suchende Element enthält (Zeile 5 in Algorithmus 5.1). Für die Laufzeit des in Abschnitt 5.3 vorgestellten *slope selection*-Algorithmus ist essentiell, dass eine solche Überprüfung keine explizite Aufzählung aller Kandidaten aus $\mathcal{S}'$ erfordert.

Generell hängt die Effizienz dieser Suchtechnik davon ab, dass die Suchdomäne $\mathcal{S}$ in möglichst wenigen Iterationen hinreichend verkleinert und damit die Anzahl $|\mathcal{S}|$ der Kandidaten hinreichend verringert werden kann. Für viele Problemstellungen kann dies gewährleistet werden, so dass die randomisierte Suche durch Interpolation der *randomisierten binären Suche* [MR95], auch randomisierte Halbierung genannt, nachweislich überlegen ist: Diese letztere, simplere und häufig verwendete Technik wählt in iterativen Suchschritten jeweils ein *einzelnes* Element R aus den Kandidaten für $\mathcal{K}$ zufällig aus. Entsprechend der linearen Ordnung auf den Kandidaten separiert dieses Element die Kandidatenmenge in zwei Teilmengen. An deren Mächtigkeit, die in linearer Zeit zu ermitteln ist, ist abzulesen, in welcher der beiden Mengen das zu suchende k-te Element $\mathcal{K}$ zu finden ist. Da das separierende Element R jeweils zufällig gewählt wird, reduziert sich die Menge der zu betrachtenden Kandidaten in einem solchen Suchschritt im erwarteten Fall um die Hälfte. Demzufolge bedarf es im erwarteten Fall einer in der Größe der Eingabe logarithmischen Anzahl von Suchschritten, um die Suche erfolgreich abzuschließen [MR95].

Die randomisierte Suche durch Interpolation bedarf hingegen für geeignete Problemstellungen nur einer erwartet *konstanten* Anzahl an Iterationen, um die Suche zu beenden. Für das im Folgenden zunächst betrachtete Problem der Suche in

einer Menge von durch Zahlen darstellbare Kandidaten ist dies nachweisbar und tatsächlich werden sich die später zu betrachtenden komplexeren Suchprobleme auf dieses einführende Problem reduzieren lassen. Der Nachweis der erwartet konstanten Anzahl an Suchschritten fußt auf folgendem Lemma:[13]

Lemma 5.2.1 (Lemma 2.1 in [MMN98]) *Sei eine Menge $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ von Zahlen sowie ein Index k mit $1 \leq k \leq n$ und eine ganze Zahl $r > 0$ gegeben. Dann ist in $\mathcal{O}(r)$ Zeit ein Intervall $\mathcal{S}' = [b', e']$ berechenbar, so dass mit einer Wahrscheinlichkeit von $1 - 1/\Omega(\sqrt{r})$ die k-kleinste Zahl aus $\mathcal{S}$ in $[b', e']$ liegt und die Anzahl der in $[b', e']$ liegenden Zahlen aus $\mathcal{S}$ durch $n/\Omega(\sqrt{r})$ beschränkt ist.*

Der Nachweis dieses Lemmas ist insofern konstruktiv, als dass er zu einer konkreten Vorschrift korrespondiert, wie die Intervallgrenzen b' und e' zu bestimmen sind. Diese Vorschrift besteht aus der Selektion zweier Zahlen b und e aus einer zufällig (mit Wiederholungen) gewählten Stichprobe $\mathcal{R}$ von $\mathcal{S}$ nach ihrem Rang k_b und k_e bezüglich der Zahlordnung $<$ der Zahlen aus $\mathcal{R}$. Die geeignete Wahl der Ränge k_b und k_e sichert, dass mit der angegebenen Wahrscheinlichkeit die gesuchte k-te Zahl der Eingabe in $[b', e']$ liegt, aber nur $n/\Omega(\sqrt{r})$ weitere Zahlen.

Diese Vorschrift kann iterativ in einem jeden Suchschritt einer randomisierten Suche durch Interpolation verwendet werden, um den jeweils nächsten Suchstreifen $\mathcal{S}'$ zu mutmaßen. Die entsprechende Konkretisierung der randomisierten Suche für eine aus Zahlen bestehende Kandidatenmenge ist nachfolgend in Algorithmus 5.2 skizziert; in den Zeilen 4 bis 6 wird besagte Vorschrift zur Wahl einer neuen Suchdomäne, bzw. der Intevallgrenzen b' und e', angewendet.

Durch die iterative Anwendung von Lemma 5.2.1 ergibt sich, dass für den Algorithmus NumberSelection die erwartete Anzahl an Suchschritten konstant ist, sofern die zufälligen Auswahlen $\mathcal{R}$ in den einzelnen Suchschritten hinreichend groß gewählt sind:

Lemma 5.2.2 (sinngemäß nach [MMN98]) *Bezeichne $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ eine Menge von Zahlen sowie k mit $(1 \leq k \leq n)$ und $r > 0$ ganze Zahlen. Existieren ein $0 < \beta < 1$, so dass $r \geq c \cdot |\mathcal{S}|^\beta$ gilt, dann ist die k-kleinste Zahl aus $\mathcal{S}$ durch $\mathcal{O}(1)$ erwartet viele Iterationen des zu Lemma 5.2.1 korrespondierenden Algorithmus, mit einer Auswahlgröße r, zu bestimmen.*

Beweis:

Nach Lemma 5.2.1 existieren ein r_0 und Konstanten $c' > 0$ und $c'' > 0$, so dass für $r \geq r_0$ eine Iteration des zu Lemma 5.2.1 korrespondierenden Algorithmus mit mindestens einer Wahrscheinlichkeit von $1 - (c'/\sqrt{r})$ „erfolgreich“ ist, dass also die gesuchte k-te Zahl, aber nicht mehr als $c'' \cdot n/\sqrt{r}$ der Zahlen aus $\mathcal{S}$ im Intervall $\mathcal{S}'$ liegen. Die Anzahl der „erfolgreichen“ Anwendungen dieses Algorithmus, die notwendig sind, um das sukzessiv verkleinerte Intervall $\mathcal{S}$ auf die zu suchende k-te Zahl zu reduzieren, ist somit nach oben beschränkt durch $\log_{\sqrt{r}/c''} n = \log_2 n \;/\; \log_2(\sqrt{r}/c'')$.

[13] Matoušek *et al.* publizierten Lemma 5.2.1 zusammen mit einer Skizze seines Nachweises und verweisen für einen kompletten Beweis auf die Beweisführung zu den Lemmata 3.1 und 3.2 in Dillencourt *et al.* [DMN92]. Diese Lemmata sind speziell für das *slope selection*-Problem formuliert, aber auch für den in Lemma 5.2.1 genannte Problemstellung anwendbar.

Algorithmus 5.2 Algorithmus NumberSelection($\mathtt{A}[0,\ldots,n-1], k, r$) bestimmt die bzgl. der Ordnung $<$ auf $\mathbb{R}$ k-te Zahl unter den Zahlen in $\mathtt{A}$.

1: Setze $b := -\infty$ und $e := \infty$. /* Erste „Schätzung“ für ein die gesuchte Zahl enthaltendes Suchintervall. */
2: **repeat**
3: Erzeuge eine Multimenge $\mathcal{R}$ von r zufällig aus $\mathtt{A}$ (mit Zurücklegen) gewählten Zahlen.
4: Setze $\kappa := (r/|\mathcal{S}|) \cdot (k - |\{s \in \mathtt{A}\colon s < b\}|)$, $\kappa_{b'} := max(1, \lfloor \kappa - 3\sqrt{r}/2 \rfloor)$, und $\kappa_{e'} := min(r, \lfloor \kappa + 3\sqrt{r}/2 \rfloor)$.
5: Bestimme Zahlen mit Rängen $\kappa_{b'}$ und $\kappa_{e'}$ (bzgl. der $<$-Ordnung) in $\mathcal{R}$.
6: Setze b' bzw. e' als die Zahlen mit Rang $\kappa_{b'}$ bzw. $\kappa_{e'}$ in $\mathcal{R}$.
7: **if** $|\{s \in \mathtt{A}\colon s < b'\}| < k \leq |\{s \in \mathtt{A}\colon s < e'\}|$ **then**
8: Setze $b := b'$ und $e := e'$. /* Die k-te Zahl befindet sich in $\langle b', e' \rangle$; verwende fortan $[b', e']$ als Suchintervall. */
9: **end if**
10: **until** $|\mathcal{S}| \leq r$ /* Kandidatenmenge hinreichend reduziert. */
11: Bestimme κ wie in Zeile 4 mit $\mathcal{R}$ als der Menge *aller* Zahlen der Eingabe in $\langle b, e \rangle$. Gib die Zahl in $\mathcal{R}$ mit Index κ in $\langle b', e' \rangle$ als die gesuchte, bzgl. $<_x$ k-te Zahl zurück.

Für eine Stichprobengröße r mit $r \geq r_0$ für ein $\beta > 0$ lässt sich dieser Term für beliebige Problemgrößen $n \geq r_0^{1/\beta}$ gegen $(\beta/2) \cdot \log_2 n \,/\, (\log_2(c''^{-2})/\beta + \log_2 n)$ abschätzen. Letzterer Ausdruck konvergiert für große Eingaben, d. h. für $n \to \infty$, gegen $\beta/2$. Für Problemgrößen $n < r_0^{1/\beta}$ ist nichts zu zeigen, da das Lemma nur eine asymptotische Aussage für $n \to \infty$ über die erwartete Anzahl der Suchschritte trifft. □

Bemerkung 5.2.3 Es sei erwähnt, dass bei der randomisierten Suche durch Interpolation die erwartete Anzahl an notwendigen Suchschritten für viele Problemstellungen und insbesondere für praktische Szenarien nicht nur konstant, sondern äußert gering ist — auch für relativ kleine Auswahlgrößen $r \approx \sqrt{n}$. Dies gilt für die in Lemma 5.2.2 dargestellte Problemstellung der Identifizierung der k-ten Zahl, aber auch für Problemstellungen des Geradenschätzens: Für die Bestimmung des Theil-Sen- oder des *repeated median*-Schätzers benötigen die bereits publizierten randomisiert durch Interpolation suchenden Algorithmen für gleichverteilte (oder günstiger: annähernd auf einer Geraden befindliche) Daten zumeist nicht mehr als vier Suchschritte bei einer Auswahlgröße $r \approx \sqrt{m}$ mit m als der Anzahl der initial zu betrachtenden Kandidaten (vergleiche [DMN92, Kapitel 5] bzw. [MMN98, Kapitel 4]).

5.2.3 Speichereffiziente randomisierte Suche durch Interpolation

Die randomisierte Suche durch Interpolation ist — unter anderem für die Suche in einer Menge von Zahlen — *in-place* zu realisieren, was im Folgenden nachgewiesen

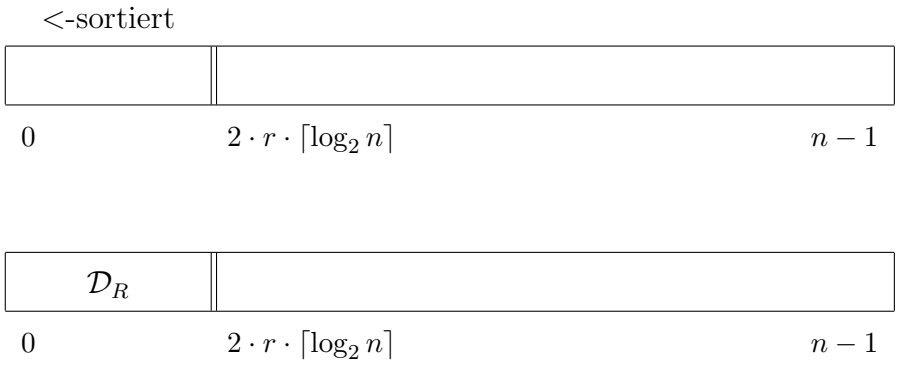
wird.

Lemma 5.2.4 *Steht nur konstant viel zusätzlicher Speicher zur Verfügung, so gelten die Aussagen von Lemma 5.2.1 und 5.2.2 mit folgenden Einschränkungen: Die zulässige Größe r von $\mathcal{R}$ ist durch $\lfloor n/2 \rfloor / \lceil \log_2 n \rceil$ beschränkt und die in Lemma 5.2.1 angegebene Laufzeit erhöht sich auf $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$.*

Beweis: Die zufällige Auswahl (mit Wiederholungen) und Speicherung von r Zahlen aus einer Menge von n Zahlen ist *in-place* in $\mathcal{O}(r \cdot \log_2 n)$ Zeit möglich: Nach Korollar 4.1.4 sind in der Eingabe bis zu $\lfloor n/2 \rfloor / \lceil \log_2 n \rceil$ viele auf Eingabeelemente verweisende Referenzen kodierbar. Dementsprechend kann eine zufällige Auswahl $\mathcal{R}$ von bis zu $\lfloor (n/2) \rfloor / \lceil \log_2 n \rceil$ vielen Zahlen aus der Eingabe implizit durch kodierte Referenzen vorgehalten werden. Die Kriterien für die Zufälligkeit der Auswahl können auch von *in-place* realisierbaren Zufallszahlgeneratoren erfüllt werden (siehe für eine Beschreibung geeigneter Generatoren beispielsweise Blum *et al.* [BBS83]).[14] Die Laufzeit für die Auswahl von r Rängen aus der Menge $[1, \ldots, n]$ durch einen solchen Generator ist in $\mathcal{O}(r \cdot \log_2 n)$.

Jeder ausgewählte Rang kann durch Permutation von jeweils $\lceil 2 \cdot \log_2 n \rceil$ Eingabeelementen kodiert werden. Dies impliziert für die Größe r der kodierbaren Auswahl eine obere Schranke von $\lfloor (n/2) \rfloor / \lceil \log_2 n \rceil$. Vor dem Kodieren werden die $2 \cdot r \cdot \lceil \log_2 n \rceil$ kodierenden Eingabeelemente zunächst (bzgl. der Zahlordnung $<$) sortiert. Diese Sortierung kann durch heapsort *in-place* und in $\mathcal{O}(r \cdot \log_2 n \cdot \log_2 r)$ Zeit erfolgen, vergleiche Abschnitt 4.1.2. Nachfolgend ist die Struktur des Eingabefeldes `A` nach der Sortierung der ersten $2 \cdot r \cdot \lceil \log_2 n \rceil$ Elemente und der anschließenden Kodierung der r Ränge (in einer Datenstruktur $\mathcal{D}_R$) durch diese Elemente skizziert.

Das zufällige Auswählen einzelner Ränge erfolgt unabhängig voneinander; die Datenstruktur wird daher wie eine fortlaufende Liste sukzessive gefüllt. Die Kodierung verändert dabei die vorher etablierte Sortierung auf den Eingabeelementen nur lokal, d. h. nur Bitnachbarn tauschen ihre Position.



Ein ausgewählter und kodierter Rang i wird als Referenz auf das Eingabeelement `A`$[i]$ — in der Reihenfolge der Elemente von `A` *nach* der Sortierung der ersten $2 \cdot r \cdot \lceil \log_2 n \rceil$ Eingabeelemente — interpretiert.

[14] In den Publikationen von Matoušek [Mat91b], Matoušek *et al.* [MMN98] und Dillencourt *et al.* [DMN92] zur randomisierten Suche durch Interpolation und ihren Anwendungen werden keine Aussagen über die Kriterien für die Zufälligkeit der Stichproben $\mathcal{R}$ getroffen. Ein Überblick über Strategien und Kriterien für zufällige Stichproben findet sich bei Knuth [Knu98b, Kapitel 3].

Das Selektieren der zwei gesuchten Elemente aus $\mathcal{R}$ nach ihrem Rang (bezüglich der Zahlordnung $<$) erfolgt durch Betrachten der Kodierung von $\mathcal{R}$ in $\mathcal{D}_R$: Die in $\mathcal{D}_R$ kodierten Ränge können zunächst — wie in Abschnitt 4.1.1 dargestellt — *in-place* und in $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$ Zeit sortiert werden. Dabei wird weder die Kodierung selbst noch die (nur lokal permutierte) Sortierung der Eingabe der ersten $2 \cdot r \cdot \lceil \log_2 n \rceil$ Eingabeelemente in `A` irreversibel korrumpiert; insbesondere ist eine in $\mathcal{D}_R$ kodierte Referenz auf eines der ersten $2 \cdot r \cdot \lceil \log_2 n \rceil$ Eingabeelemente $\mathtt{A}[i]$ — wegen der vorab etablierten Sortierung auf diesem Eingabeteil— nach wie vor eindeutig und in konstanter Zeit auszulesen. Die zwei notwendigen Selektionen können nun durch direkten Zugriff (auf das $\kappa_{b'}$-te und das $\kappa_{e'}$-te Element) der Sortierung erfolgen; das Dekodieren der beiden selektierten Zahlen kostet erneut $\mathcal{O}(\log_2 n)$ Zeit.

Die oben beschriebene und gegenüber den Lemmata 5.2.1 und 5.2.2 konkretisierte Form der Speicherung von $\mathcal{R}$ (durch die Struktur $\mathcal{D}_R$) hat keinerlei Einfluss auf die den Lemmata 5.2.1 und 5.2.2 zu Grunde liegende probabilistische Analyse. Folglich bleiben die weiteren Aussagen der Lemmata auch bei Verwendung von nur konstant viel zusätzlichem Speicher bestehen. □

Dieses nur einführende (jedoch verallgemeinerbare) Beispiel liefert einen *in-place*-Elementselektionsalgorithmus mit erwartet linearer Laufzeit, sofern die Stichprobengrößen r in jedem Suchschritt aus $\mathcal{O}(n/\log_2^2 n)$ und gleichzeitig kleiner oder gleich $\lfloor n/2 \rfloor / \lceil \log_2 n \rceil$ gewählt werden:

Korollar 5.2.5 *Die Elementselektion des k-ten Elementes einer streng linear anordbaren Menge von Zahlen ist* in-place *und in erwarteter Zeit von $\mathcal{O}(n)$ per randomisierter Suche durch Interpolation möglich.*

5.3 Randomisierte Selektion einer Geraden medianer Steigung

Das Schätzen der Steigung einer Geraden durch den Theil-Sen-Schätzer ist auch im Gebiet der algorithmischen Geometrie ein gut bekanntes und viel diskutiertes Problem, welches meist als *slope selection*-Problem bezeichnet wird. Im Folgenden bestehe eine Probleminstanz aus einer in einem Eingabefeld `A` übergebenen Menge $\mathcal{P}$ von Punkten in der Ebene. Statt der ursprünglich angegebenen Formulierung des Problems der Bestimmung einer Geraden mit *medianer* Steigung (unter den von $\mathcal{P}$ induzierten Geraden) wird die allgemeinere *slope selection*-Problemstellung der Bestimmung der bezüglich ihrer Steigung k-ten Geraden (unter den von $\mathcal{P}$ induzierten Geraden) betrachtet.

Bislang publizierte Resultate Für das *slope selection*-Problem ist eine untere Schranke von $\Omega(n \cdot \log_2 n)$ in folgender Weise nachweisbar [CSSS89]: Das Elementeindeutigkeitsproblem, das als die Entscheidung definiert ist, ob eine übergebene Multimenge $\mathcal{H}$ von Zahlen duplikatfrei ist, besitzt eine untere Schranke von $\Omega(n \cdot \log_2 n)$ [PS85]. Eine Instanz dieses Problems ist in linearer Zeit in eine Instanz des *slope selection*-Problems transformierbar, indem eine jede Zahl h_i der Eingabe

$\mathcal{H}$ auf den Punkt (h_i, i) abgebildet wird. Nun ist die betragsmäßig größte Steigung unter den Steigungen aller Geraden durch je zwei Punkte (h_i, i) genau dann endlich, wenn die Eingabe $\mathcal{H}$ duplikatfrei ist. Ein *slope selection*-Algorithmus mit einer Laufzeit außerhalb von $\Omega(n \cdot \log_2 n)$ würde somit einen Widerspruch zur unteren Schranke für das Elementeindeutigkeitsproblem darstellen.

Es wurden für das *slope selection*-Problem mehrere deterministische Algorithmen mit der optimalen Laufzeit von $\Theta(n \cdot \log_2 n)$ entwickelt: Zunächst wurde von Cole *et al.* [CSSS89] ein Algorithmus vorgestellt, der unter anderem Sortiernetzwerke und die Technik der parametrischen Suche verwendet. Weitere Algorithmen von Katz und Sharir [KS93] sowie von Brönnimann und Chazelle [BC98] stützen sich auf alternative, aber ebenfalls nur aufwendig zu implementierende Techniken: Im ersteren Fall sind dies Expandergraphen, im letzteren *ϵ-nets*, deren Verwendung laut den jeweiligen Autoren gegenüber der Verwendung von Sortiernetzwerken und der parametrischen Suche zu tendenziell praktikableren Algorithmen führt.

Des Weiteren wurden randomisierte Algorithmen mit einer optimalen erwarteten Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ vorgestellt, die gegenüber bekannten deterministischen Algorithmen einfacher zu implementieren sind [DMN92, SS93]. Zwei Varianten desselben Algorithmus sind unabhängig voneinander von Matoušek [Mat91b] und von Dillencourt *et al.* [DMN92] vorgestellt worden. Noch effizienter als diese Varianten, zumindest für kleine Probleminstanzen, arbeitet nach Aussage von Shafer und Steiger der von ihnen vorgestellte Algorithmus [SS93].

Die duale Formulierung des *slope selection*-Problems Für die Auswahl der bezüglich ihrer Steigung k-ten Geraden wird die folgende duale Form dieser Problemstellung betrachtet, für die die Eingabe des Problems durch eine *Dualitätsabbildung* f in einen dualen Raum abgebildet wird. Diese Dualitätsabbildung transformiert bijektiv Geraden in Punkte und Punkte in Geraden. Zudem kann die Abbildung f so definiert werden, dass Lagebeziehungen zwischen Geraden und Punkten bewahrt bleiben. Zur Lösung des vorliegenden Problems sei die Dualitätsabbildung wie folgt gewählt: Einem Punkt (x, y) wird die Gerade $\{(\xi, \upsilon) \mid \upsilon = x \cdot \xi - y\}$ zugeordnet. Somit entsprechen die durch je zwei Punkte aus $\mathcal{P}$ gebildeten Geraden aus dem ursprünglichen Raum den $\binom{n}{2}$ Schnittpunkten im dualen Raum, die jeweils durch ein Paar aus $\mathcal{D}(\mathcal{P})$, der Menge der Duale der Eingabepunkte, bestimmt sind. Die Bestimmung der bezüglich ihrer Steigung k-ten Geraden lässt sich nun im dualen Raum formulieren: Zu bestimmen ist der bezüglich $<_x$, d. h. bezüglich seiner x-Koordinate, k-te Schnittpunkt unter den Schnittpunkten aller Geraden in $\mathcal{D}(\mathcal{P})$.[15]

Im linken Teil der Abbildung 5.2 sind eine Beispieleingabe $\mathcal{P}$ sowie alle durch je zwei Punkte dieser Eingabe definierten Geraden dargestellt; im rechten Teil sind die Duale der Eingabepunkte und dieser Geraden abgebildet. Die Lösung des *slope selection*-Problems für die dargestellte Eingabe ist in beiden Abbildungsteilen rot markiert.

Bemerkung 5.3.1 In den folgenden Beschreibungen wird vereinfachend davon ausgegangen, dass als Eingabe im Eingabefeld `A` die Menge der zu $\mathcal{P}$ dualen Geraden

[15] Das Dual einer Geraden durch zwei vertikal übereinander liegende Punkte (x, y_1) und (x, y_2) der originären Eingabe sei als Schnittpunkt bei $(+\infty, +\infty)$ festgelegt.

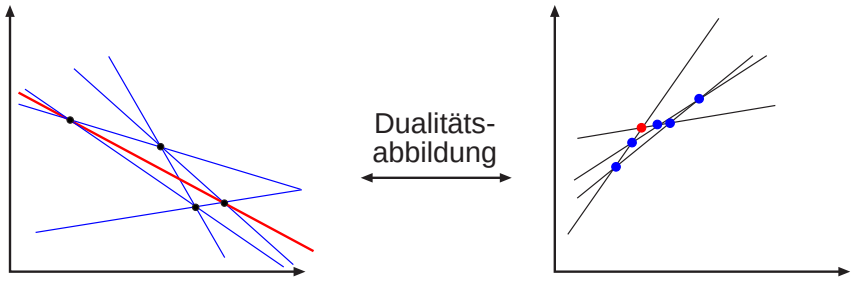


Abbildung 5.2: Probleminstanz und Lösung zur originären und dualen *slope selection*-Problemstellung

vorliegt. Dabei ist zu beachten, dass der im Folgenden beschriebene *slope selection*-Algorithmus keine tatsächliche Transformation der Eingabe erfordert: Das Dual eines Eingabepunktes ergibt sich eindeutig aus der in `A` vorliegenden Punktdarstellung und lässt sich aus dieser bei Bedarf in konstanter Zeit berechnen. Die Kosten für Berechnungen von Dualen müssen folglich in den folgenden asymptotischen Laufzeitanalysen nicht explizit betrachtet werden, da sie die Kosten für Zugriffe auf Elemente aus `A` nicht dominieren.

5.3.1 Suche durch randomisierte Interpolation nach einer Geraden mit medianer Steigung

Das Arrangement aller Geraden der (dualen) Eingabemenge $\mathcal{D}(\mathcal{P})$ einer *slope selection*-Probleminstanz enthält $\Theta(n^2)$ Schnittpunkte. Für den zu bestimmenden Schnittpunkt mit k-ter x-Koordinate existieren also quadratisch viele Kandidaten. Daher wäre eine explizite Berechnung *aller* dieser Schnittpunkte nicht in der angestrebten asymptotischen Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ möglich. Des Weiteren bedürfte es zur Speicherung dieser Schnittpunkte einer superlinearen Anzahl von Speicherplätzen. Im Folgenden wird beschrieben, wie sich durch Verwendung der randomisierten Suche durch Interpolation die explizite Berechnung aller Schnittpunkte vermeiden und ein bezüglich Laufzeit und Speicherplatzbedarf asymptotisch optimaler Algorithmus für das *slope selection*-Problem realisieren lässt.

Eine Skizze von Matoušeks Algorithmus

Der unabhängig voneinander von Matoušek [Mat91b] und Dillencourt *et al.* [DMN92] vorgestellte *slope selection*-Algorithmus verwendet die randomisierte Suche durch Interpolation und arbeitet folglich in Suchschritten, in denen die Kandidatenmenge für den gesuchten Schnittpunkt bzw. die Suchdomäne $\mathcal{S}$ jeweils verkleinert werden soll. Die Suchdomäne ist für das (duale) *slope selection*-Problem ein y-paralleler, d. h. durch ein x-Intervall definierter, Suchstreifen $\mathcal{S} = \langle b, e\rangle := [b, e) \times \mathbb{R} \subset \mathbb{R}^2$, der initial alle Schnittpunkte der Geraden der Eingabe enthält[16] und welcher in den nachfolgenden Suchschritten jeweils verkleinert wird. Dabei wird die Invariante aufrecht

[16] Dies ist nur für Eingaben ohne parallele Geraden korrekt. Wie mit Eingaben mit parallelen Geraden zu verfahren ist, wird in Bemerkung 5.3.10 erläutert.

erhalten, dass der gesuchte (bzgl. $<_x$) k-te Schnittpunkt immer im aktuellen Suchstreifen enthalten ist. In jedem Suchschritt wird aus dem aktuellen Suchstreifen eine Menge $\mathcal{R}$ von Schnittpunkten zufällig ausgewählt. Aus $\mathcal{R}$ wird interpolierend eine Schätzung aus $\mathcal{I}(b, e) := [b, e)$ für die x-Koordinate des gesuchten Schnittpunktes bestimmt. Ebenfalls durch Interpolation wird ein um diese Schätzung „zentrierter“ (Unsicherheits-) Streifen $\mathcal{S}' = \langle b', e' \rangle$ bestimmt, der mit einer hohen Wahrscheinlichkeit den gesuchten Schnittpunkt enthält. Ein anschließender Test überprüft, ob der zu bestimmende Schnittpunkt tatsächlich im Streifen $\langle b', e' \rangle$ liegt, so dass letzterer als neue Suchdomäne verwendet werden kann. Für diese Überprüfung ist für jeden der quadratisch vielen Schnittpunkte nur seine Zugehörigkeit zum Streifen $\langle b', e' \rangle$ bzw. zur Halbebene links bzw. rechts dieses Streifens relevant; daher ist dieser Test in $\mathcal{O}(n \cdot \log_2 n)$ Zeit durch eine in Abschnitt 5.3.2 erläuterte Adaption des klassischen Inversionen zählenden Algorithmus[17] durchführbar.

Das geschilderte, bereits von Matoušek [Mat91b] und von Dillencourt *et al.* [DMN92] beschriebene algorithmische Gerüst zur randomisierten Suche für die *slope selection*-Problemstellung ist als Algorithmus 5.3 skizziert. Mit $\mathcal{I}(b, e)$ sei hierfür die Anzahl der Schnittpunkte, deren x-Koordinate in $[b, e)$ fällt, bezeichnet. Für die

---

Algorithmus 5.3 Algorithmus SlopeSelection($\mathtt{A}[0, \ldots, n-1], k, r$) bestimmt den bzgl. der $<_x$-Sortierung k-ten Schnittpunkt unter den Schnittpunkten der Geraden (bzw. der Duale der Punkte) in $\mathtt{A}$ [Mat91b].

1: $b := -\infty$; $e := \infty$. /* Erste „Schätzung“ für einen den gesuchten Schnittpunkt enthaltenden Suchstreifen. */
2: **repeat**
3: $N := |\mathcal{I}(b, e)|$. /* Anzahl der Schnittpunkte im Streifen $\langle b, e \rangle$. */
4: Erzeuge eine Multimenge von r zufällig aus $\mathcal{I}(b, e)$ (mit Zurücklegen) gewählten Schnittpunkten.
5: $\kappa := (r/N)(k - |\mathcal{I}(-\infty, b)|)$; $\kappa_{b'} := max(1, \lfloor \kappa - 3\sqrt{r}/2 \rfloor)$; $\kappa_{e'} := min(r, \lfloor \kappa + 3\sqrt{r}/2 \rfloor)$.
6: Bestimme Schnittpunkte mit Rängen $\kappa_{b'}$ und $\kappa_{e'}$ (bzgl. $<_x$-Ordnung) in $\mathcal{R}$.
7: Setze b' und e' als x-Koordinaten der Schnittpunkte mit Rang $\kappa_{b'}$, bzw. $\kappa_{e'}$ in $\mathcal{R}$.
8: **if** $|\mathcal{I}(-\infty, b')| < k \leq |\mathcal{I}(-\infty, e')|$ **then**
9: $b := b'$; $e := e'$. /* Der k-te Schnittpunkt befindet sich in $\langle b', e' \rangle$; verwende fortan $\langle b', e' \rangle$ als Suchstreifen. */
10: **end if**
11: **until** $N \leq r$ /* Kandidatenmenge hinreichend reduziert. */
12: Bestimme κ wie in Zeile 5 mit $\mathcal{R}$ als der Menge *aller* Schnittpunkte in $\langle b, e \rangle$. Gib den Schnittpunkt in $\mathcal{R}$ mit Index κ in $\langle b', e' \rangle$ als den gesuchten, bzgl. $<_x$ k-ten Schnittpunkt zurück.

Mutmaßung eines verkleinerten Suchstreifens wird die zu Lemma 5.2.1 korrespondierende Vorschrift verwendet (Zeile 5 in Algorithmus 5.3); die Aussagen der Lemmata 5.2.1, 5.2.2 und 5.2.4 übertragen sich auf den Fall der Bestimmung des k-ten der von

[17] Eine Beschreibung des ursprünglichen Inversionen zählenden Algorithmus findet sich beispielsweise bei Kleinberg und Tardos [KT05].

den Geraden der Eingabe induzierten Schnittpunkte: Die initiale Kandidatenmenge in der vorliegenden Problemstellung ist die (Multi-)menge der x-Koordinanten der $\binom{n}{2}$ Schnittpunkte; ist diese Multimenge duplikatfrei, so bilden die Kandidaten eine strenge lineare Ordnung und können jeweils durch eine Zahl (aus $\{1, \ldots, n^2\}$) repräsentiert werden. Die strenge Anordbarkeit der Kandidatenmenge bezüglich $<_x$, dass also die originäre Eingabe keine drei kollinearen originären Punkte enthält, ist jedoch keine zwingende Voraussetzung, um das Resultat von Lemma 5.2.2 zu erhalten. Ein entsprechender Nachweis ist von Dillencourt *et al.* erbracht worden (siehe Lemma 3.1 und 3.2 bei Dillencout *et al.* [DMN92]).[18] Folglich lassen sich auch für solche Eingaben die Suchstreifen $\mathcal{S}'$ iterativ so erzeugen (Zeile 4 bis 7 in Algorithmus 5.3), dass für die Beendigung der Suche nur eine erwartet konstante Anzahl an Suchschritten notwendig wird. Da die Laufzeit eines Suchschrittes von der Abarbeitung des Inversionen zählenden Algorithmus, dominiert wird, ist die erwartete Gesamtlaufzeit von Algorithmus SlopeSelection (Algorithmus 5.3) in $\mathcal{O}(n \cdot \log_2 n)$.

Überblick über die speichereffiziente Realisierung

Im Folgenden wird eine speichereffiziente Realisierung des soeben beschriebenen *slope selection*-Algorithmus vorgestellt, die die gleiche erwartete asymptotische Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ wie der ursprüngliche Algorithmus von Matoušek [Mat91b] besitzt.

Satz 5.3.2 *Die Bestimmung der nach ihrer Steigung k-ten Geraden aus der Menge der Geraden, die durch je zwei Punkte aus einer Menge von n Punkten der Ebene induziert werden, ist* in-place *und in $\mathcal{O}(n \cdot \log_2 n)$ erwarteter Zeit möglich.*

Da die randomisierte Suche durch Interpolation nach der zu bestimmenden k-ten Steigung nur konstant viele Suchschritte benötigt, genügt für den Nachweis von Theorem 5.3.2 die Angabe einer *in-place*-Realisierung eines einzelnen Suchschrittes mit einer erwarteten Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$.

Der Ablauf eines solchen Suchschrittes lässt sich wiederum in drei algorithmische Teilschritte untergliedern, für die jeweils $\mathcal{O}(n \log_2 n)$ Zeit und konstant viel zusätzlicher Speicher genutzt werden kann.

Zufällige Auswahl: Die zufällige und speichereffiziente Auswahl von r Schnittpunkten (Zeile 4 in Algorithmus 5.3) ist der algorithmisch herausforderndste Teilschritt. Eine *in-place*-Realisierung wird in den Abschnitten 5.3.3 bis 5.3.6 beschrieben.

Schätzung: Die Schätzung eines verkleinerten Suchstreifens $\langle b', e' \rangle$ durch speichereffiziente Selektion zweier durch ihre Ränge (bzgl. $<_x$) vorgegebene Schnittpunkte aus $\mathcal{R}$ (Zeile 5 bis 7 in Algorithmus 5.3) wird in Abschnitt 5.3.7 erläutert.

Überprüfung: Eine *in-place*-Realisierung der Überprüfung, ob der Streifen $\langle b', e' \rangle$ den zu bestimmenden k-ten Schnittpunkt enthält (Zeile 8 in Algorithmus 5.3), wird im folgenden Abschnitt 5.3.2 beschrieben.

[18] In dieser Arbeit wird an späterer Stelle auf Eingaben in nicht allgemeiner Lage, d.h. mit mindestens drei kollinearen Punkten sowie auf Duplikate enthaltende Eingaben genauer eingegangen.

Der letzte Schritt der Suche durch randomisierte Interpolation erfolgt deterministisch, nachdem die Menge der zu betrachtenden Schnittpunkte auf $\mathcal{O}(r)$ reduziert wurde. Diese abschließende deterministische Phase (Zeile 12 in Algorithmus 5.3) ist ein Spezialfall des randomisierten Suchschrittes und wird in Abschnitt 5.3.7 erläutert.

5.3.2 Zählen von Schnittpunkten innerhalb eines Streifens

Die Überprüfung in Zeile 8 in Algorithmus 5.3, ob der (bzgl. $<_x$) k-te Schnittpunkt sich in dem Streifen $\langle b', e' \rangle$ befindet, ist auf das Bestimmen der Anzahl der Schnittpunkte in den (rechts offenen) Streifen $\langle \infty, b' \rangle$ und $\langle b', e' \rangle$ zurückzuführen. Diese Aufgabe wird nun genauer betrachtet.

Zählen von Schnittpunkten durch Zählen von Inversionen

Um Inversionen und Schnittpunkte in Beziehung zu setzen, sei vorab der Begriff der Inversion konkretisiert und die Ordnung von Geraden an den Berandungen von Streifen $\langle b, e \rangle$ definiert:

Definition 5.3.3 Die lexikographische Ordnung $<_{\xi_-}$ (bzw. $<_{\xi_+}$) für $\xi \in \mathbb{R}$ sei für Geraden $g_i(x) = a_i \cdot x + b_i$ in der Ebene wie folgt definiert:
$g_1 <_{\xi_-} g_2 \;:\Leftrightarrow\; g_1(\xi) < g_1(\xi) \;\vee\; (g_1(\xi) = g_1(\xi) \wedge a_i < a_2)$
(bzw. $g_1 <_{\xi_+} g_2 \;:\Leftrightarrow\; g_1(\xi) < g_1(\xi) \;\vee\; (g_1(\xi) = g_1(\xi) \wedge a_i > a_2)$)

Bemerkung 5.3.4 Die Ordnungen $<_{\xi_-}$ und $<_{\xi_+}$ sind nicht für vertikale Geraden definiert. Dies ist im Kontext des *slope selection*-Problems auch nicht notwendig, da die originäre Eingabe nur aus Punkten mit endlichen Koordinaten besteht, woraus folgt, dass die duale Eingabe keine vertikalen Geraden enthält.

Definition 5.3.5 Seien zwei Permutationen π_1 bzw. π_2 einer Eingabe E gegeben, die Sortierungen bezüglich einer linearen Ordnung $<_1$ bzw. $<_2$ von E entsprechen. Eine *Inversion zwischen Sortierungen* π_1 und π_2 ist ein Paar von nicht identischen Elementen e und e' aus E mit $e <_1 e' \;\wedge\; e' <_2 e$ oder $e' <_1 e \;\wedge\; e <_2 e'$.

Für das effiziente Zählen von Schnittpunkten, die in einen y-Streifen $\langle b, e \rangle$ fallen, kann, sofern die Schnittpunkte ausschließlich von Geraden induziert sind, folgende bekannte Beobachtung ausgenutzt werden:

Lemma 5.3.6 *Sei $\mathcal{P}$ eine Menge von Gerade in der Ebene. Die Anzahl der Schnittpunkte,die von Elementen von $\mathcal{P}$ gebildet werden un in einen (rechtseitig offenen) Streifen $\langle b, e \rangle$ fallen, entspricht genau der Anzahl der Inversionen zwischen der $<_{b_-}$-Sortierung und der $<_{e_-}$-Sortierung von $\mathcal{P}$.*

Beweis: Betrachte die lineare Differenzfunktion h zweier nicht paralleler Geraden g_1 und g_2 aus $\mathcal{P}$. Eine Inversion von g_1 und g_2 bezüglich ihrer $<_{b_-}$- und $<_{e_-}$-Sortierung entspricht einem unterschiedlichen Vorzeichen der Differenzfunktion an $x = b$ und $x = e$. Da h streng monoton sowie linear und damit stetig ist, besitzt h zumindest eine Nullstelle in $\mathcal{I}(b, e)$. Somit existiert ein Schnittpunkt von g_1 und g_2 in $\langle b, e \rangle$.

Dieser Schnittpunkt ist der einzige von g_1 mit g_2, da h als lineare Funktion keine weiteren Nullstellen besitzt.

Besteht andererseits keine Inversion von g_1 und g_2 bezüglich ihrer $<_{b\_}$- und $<_{e\_}$-Sortierung, so hat h an $x = b$ und $x = e$ dasselbe Vorzeichen. Daraus folgt wiederum aus der Linearität von h, dass h keine Nullstelle in $\mathcal{I}(b, e)$ besitzt.

Der Fall zweier paralleler oder identischer Geraden g_1 und g_2 entspricht nach Konvention einem Schnittpunkt bei $x = +\infty$, d. h. außerhalb von $\langle b, e\rangle$. Zu zeigen ist, dass g_1 und g_2 keine Inversion bezüglich $<_{b\_}$- und $<_{e\_}$-Ordnung implizieren. Für zwei parallele Geraden ist dies klar, da deren vertikale Ordnung an $x = b$ ihrer vertikalen Ordnung an $x = e$ entspricht. Für zwei identische Geraden folgt aus Definition , dass sie keine Inversion herbeiführen. □

Somit kann die Bestimmung der Schnittpunktzahl in einem Streifen $\langle b, e\rangle$ durch Adaption des nachfolgend beschriebenen bekannten Algorithmus für das Zählen von Inversionen (siehe etwa [KT05]) formuliert werden, welcher eine Erweiterung des iterativen *mergesort*-Algorithmus ist.

Laufzeiteffizientes Zählen von Inversionen Der erwähnte Algorithmus hat für das Zählen der Inversionen, die genau den Schnittpunkten in einem Streifen $\langle b, e\rangle$ entsprechen, folgende Vorbedingung herzustellen: Die gesamte Eingabe hat in der $<_{b\_}$-Ordnung, d. h. der (lexikografischen) vertikalen Ordnung am linken Rand des Intervalls $\langle b, e\rangle$, vorzuliegen. Der Algorithmus folgt nun dem *divide & conquer*-Prinzip [CLR01], d. h. er unterteilt die Eingabe rekursiv, um dann Teilprobleme von konstanter Größe zu lösen, im konkreten Fall die Überführung der Eingabeteile in die $<_{e\_}$-Ordnung. Anschließend verschmilzt der Algorithmus rekursiv Teile der Eingabe, so dass sich die gesamte Eingabe nach Terminierung in $<_{e\_}$-Ordnung befindet: In jedem Verschmelzungsschritt werden zwei in $\mathtt{A}$ zusammenhängende, bereits $<_{e\_}$-sortierte Teile $\mathtt{A}_1 = \mathtt{A}[i_1, \ldots, i_1 + c_1 - 1]$ und $\mathtt{A}_2 = \mathtt{A}[i_2, \ldots, i_2 + c_2 - 1]$ der Eingabe zu einem $<_{e\_}$-sortierten Eingabeteil $\mathtt{A}_1 \cup \mathtt{A}_2$ verschmolzen. Diese Eingabeteile sind jeweils benachbart, insbesondere gilt für sie, dass alle Elemente von $\mathtt{A}_1$ bezüglich $<_{b\_}$ kleiner als alle Elemente von $\mathtt{A}_2$ sind. Inversionen zwischen der $<_{b\_}$- und der $<_{e\_}$-Ordnung können in einem Verschmelzungsschritt protokolliert bzw. — für das vorliegende Problem ausreichend — gezählt werden, indem die (zum Zwecke der Etablierung der $<_{e\_}$-Ordnung notwendigen) Positionsveränderungen individueller Eingabeelemente wie folgt beobachtet werden.

Jedes Eingabeelement $l_i \in \mathtt{A}_1$ induziert eine Inversion mit allen Elementen in $\mathtt{A}_2$, die „kleiner“ bezüglich der $<_{e\_}$-Ordnung als l_i sind. Alle Inversionen können mit Hilfe einer Zählvariablen gezählt werden, die in folgender Weise erhöht wird: Wird beim Verschmelzungsprozess eine Gerade l_i des ersten Eingabeteiles $\mathtt{A}_1$ in die korrekte $<_{e\_}$-Sortierung eingefügt, wird die Zählvariable um die Anzahl der Geraden aus dem zweiten Eingabeteil $\mathtt{A}_2$ inkrementiert, die bereits einsortiert wurden, d. h. die bzgl. der $<_{e_}$Sortierung „kleiner“ als l_1 sind. Diese Anzahl entspricht genau der Anzahl der Inversionen, die l_i gemeinsam mit Geraden aus $\mathtt{A}_2$ bildet. Das Verfahren zum Zählen der Inversionen ist nachfolgend als Algorithmus 5.4 skizziert.

In Abbildung 5.3 ist die Abarbeitung der Verschmelzungsschritte des Inversionen zählenden Algorithmus, angewendet auf eine vierelementige Beispieleingabe,

Algorithmus 5.4 Algorithmus CountInversions($\mathtt{A}, b_1, b_2, c_1, c_2, \langle b, e\rangle, c$) erhöht die Zählvariable c für die bisher gefundenen Schnittpunkte in $\langle b, e\rangle$ um die Anzahl der Schnittpunkte in $\langle b, e\rangle$, die von je einer Geraden aus $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und einer Geraden aus $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ gebildet werden.

Vorbedingung: $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ sind bzgl. $<_{e\_}$ sortiert; $l_1 \in \mathtt{A}[b_1, \ldots, b_1 + c_1] \wedge l_2 \in \mathtt{A}[b_2, \ldots, b_2 + c_2]$:$l_1 <_{b\_} l_2$

Nachbedingung: $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ sind bzgl. $<_{e\_}$ sortiert.

```
1: i_1 := 0; i_2 := 0.
2: for i = 0 to c_1 + c_2 + 1    /* Das i-te Element in sortierter Reihenfolge ist A[i_1]
   oder A[i_2]. */  do
3:    if A[b_1 + i_1] <_e A[b_2 + i_2]    /* ♯(Durch A[b_1 + i_1] bedingte Inversionen) =
      ♯(Elemente in A[b_2, ..., b_2 + c_2] vor A[b_1 + i_1]) bzgl. <_e). */ then
4:       c := c + i_2 + 1.                                  /* Zähle Schnittpunkte. */
5:       i_1 := i_1 + 1.                                    /* Inkrementiere i_1. */
6:    else
7:       i_2 := i_2 + 1.                                    /* Inkrementiere i_2. */
8:    end if
9: end for
```

skizziert. Im linken Teil der Abbildung ist die Eingabe im Streifen $\langle b, e\rangle$ dargestellt. Die Ordnungen der Geraden am linken bzw. rechten Intervallrand entsprechen der $<_{b\_}$- bzw. der $<_{e\_}$-Ordnung der Geraden. Im rechten Teil der Abbildung ist die rekursive Abarbeitung der links dargestellten Eingabe (nach der Aufteilung der Eingabe in einelementige Teilmengen) veranschaulicht: Auf der untersten Ebene des Rekursionsbaumes ist sowohl die rekursive Aufteilung der Eingabe in einelementige Teilmengen als auch die initiale $<_{b\_}$-Sortierung der Eingabeelemente zu erkennen. In jeder der beiden höheren Rekursionsebenen werden die (bereits $<_{e\_}$-sortierten) Teile der Eingabe in $<_{e\_}$-Ordnung verschmolzen. Die hierfür notwendigen Positionsveränderungen der Elemente sind durch Pfeile gekennzeichnet. Den jeweiligen Positionsveränderungen entsprechen die im linken Teil der Abbildung (farblich korrespondierend) dargestellten Schnittpunkte.

Zur Korrektheit des Inversionen zählenden Algorithmus Die Korrektheit des Inversionen zählenden Algorithmus beruht darauf, dass jeweils nur benachbarte Teile der Eingabe verschmolzen werden, sowie auf den folgenden beiden Beobachtungen:

Beobachtung 5.3.7 *Seien $\mathtt{A}_1$ und $\mathtt{A}_2$ zwei (in dieser Reihenfolge) aufeinander folgende Eingabeteile von $\mathtt{A}$, die in einem Schritt des Inversionen zählenden Algorithmus zu verschmelzen sind. Dann ist jede Gerade in $\mathtt{A}_1$ bezüglich der $<_{b\_}$-Ordnung „kleiner“ als alle Geraden in $\mathtt{A}_2$.*

Beobachtung 5.3.8 *Seien $\mathtt{A}_1$ und $\mathtt{A}_2$ zwei (in dieser Reihenfolge) aufeinanderfolgende Eingabeteile von $\mathtt{A}$, die in einem Schritt des Algorithmus zum Zählen von Inversionen zu verschmelzen sind. Des Weiteren sei $g_{\min}$ die bezüglich der $<_{b\_}$-Ordnung „kleinste“ Gerade aus $\mathtt{A}_1 \cup \mathtt{A}_2$ sowie $g_{\max}$ die bezüglich der $<_{b\_}$-Ordnung*

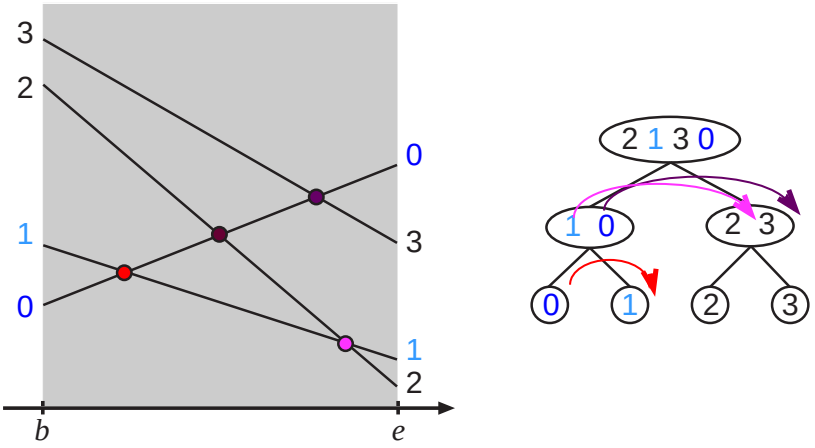


Abbildung 5.3: Beispielhafte Abarbeitung des Zählens von Inversionen (bzw. Schnittpunkten) durch Verschmelzen.

„größte" Gerade aus $\mathbf{A}_1 \cup \mathbf{A}_2$*. Dann besteht* $\mathbf{A}_1 \cup \mathbf{A}_2$ *aus genau den Geraden* g *aus* $\mathbf{A}$*, die* $g_{\min} \leq_{b\_} g \leq_{b\_} g_{\max}$ *erfüllen.*

Beide Beobachtungen ergeben sich direkt daraus, dass die Eingabe initial $<_{b\_}$-sortiert ist und der Algorithmus als Nachbedingung jeder Verschmelzung die $<_{e\_}$-Sortierung des verschmolzenen Eingabeteiles zusichert. Aus den obigen Beobachtungen ergibt sich zudem, dass die durch den Algorithmus zu verschmelzenden Teile jeweils zusammenhängende Blöcke der Eingabe bezüglich der ursprünglich etablierten $<_{b\_}$-Sortierung darstellen.

Da vor einem jeweiligen Verschmelzungsschritt jeder der beiden Eingabeteile bereits rekursiv betrachtet wurde, wurden auch alle Schnittpunkte, die jeweils von zwei Geraden nur eines Eingabeteiles gebildet werden, bereits gezählt.

Zur Laufzeit des Inversionen zählenden Algorithmus Der beschriebene Inversionen zählende Algorithmus besitzt als Variante des *mergesort*-Algorithmus dessen asymptotische Laufzeitkomplexität, die sich wie folgt ergibt: Jeder Verschmelzungsschritt verursacht Kosten, die linear in der Anzahl der zu verschmelzenden Elemente sind. Da jedes Element der (n-elementigen) Eingabe auf jeder der $\Theta(\log_2 n)$ Rekursionsebenen der Abarbeitung in genau einem Verschmelzungsschritt betrachtet wird, resultiert eine Laufzeit von $\Theta(n \cdot \log_2 n)$.

Die zusätzliche Protokollarbeit des Inversionen zählenden Algorithmus wird in den einzelnen Verschmelzungsschritten vollzogen und verursacht in diesen nur konstante Kosten pro betrachtetem Element der Eingabe, wie der Beschreibung durch Algorithmus 5.4 zu entnehmen ist.

Bemerkung 5.3.9 In Algorithmus 5.4 ist nur das eigentliche Zählen der Inversionen skizziert. Da der Inversionen zählende Algorithmus sich jedoch auf die Invariante stützt, dass zwei jeweils zu verschmelzenden Eingabeteile in $<_{e\_}$-Ordnung vorliegen, muss diese Ordnung tatsächlich hergestellt werden und also das das in Algorithmus 5.4 nicht dargestellte tatsächliche Verschmelzen jeweils beider Eingabeteile durch-

geführt werden. Für die spätere Betrachtung der *in-place*-Variante des Inversionen-Zählens ist die Beobachtung wesentlich, dass es sich nicht auf die Korrektheit und die asymptotische Laufzeitkomplexität des Inversionen zählenden Algorithmus auswirkt, separat zunächst das Zählen der Inversionen wie in Algorithmus 5.4 vorzunehmen und erst anschließend einen beliebigen Algorithmus mit linearer Laufzeit für das tatsächliche Verschmelzen der Eingabeteile zu nutzen.

Bemerkung 5.3.10 Um mit dem oben beschriebenen Algorithmus Schnittpunkte in offenen, links offenen oder geschlossenen Streifen zu zählen, müssen nur die verwendeten Sortierungen der Eingabe angepasst werden: Um die Schnittpunkte an $x = b$ von der Zählung auszuschließen, verwende man die $<_{b_+}$- statt der $<_{b_-}$-Ordnung; um die Schnittpunkte an $x = e$ in die Zählung mit einzubeziehen, verwende man die $<_{e_+}$- statt der $<_{e_-}$-Ordnung.

Für den Fall, dass der k-te Schnittpunkt zu suchen ist, allerdings weniger als k „echte“ Schnittpunkte, d. h. Schnittpunkte im rechts offenen Streifen $\langle -\infty, +\infty \rangle$, existieren, ist ein „Schnittpunkt“ zweier paralleler Geraden zurückzugeben.

Ein solcher Schnittpunkt an $x = +\infty$ kann durch den Inversionen zählenden Algorithmus, angewendet auf den geschlossenen Streifen $\langle +\infty, +\infty \rangle$, identifiziert werden. Es genügt allerdings auch ein linearer Durchlauf durch die $<_{+\infty}$-Sortierung der Eingabe: Parallele oder identische Geraden sind durch Betrachten von bezüglich dieser Sortierung benachbarten Elemente in linearer Zeit zu identifizieren.

Es ist zusätzlich zu beachten, dass die Stabilität des *mergesort*-Sortieralgorithmus auch für den Inversionen zählenden Algorithmus wesentlich ist, sofern die Eingabe $\mathcal{P}$ Duplikate enthält: Die „Stabilität“ des Inversionen zählenden Algorithmus garantiert, dass dieser nicht den „Schnittpunkt“ zweier identischer Geraden irrtümlich als Inversion zählt.

Eine Betrachtung degenerierter Konfigurationen, welche durch parallele Geraden oder Schnittpunkte mit gleichen x-Koordinaten entstehen, sowie Alternativen zur Auflösung solcher Konfigurationen durch den Inversionen zählenden Algorithmus findet man auch bei Dillencourt *et al.* [DMN92, Kapitel 5.1].

Speichereffizientes Zählen von Inversionen

Der beschriebene und bereits von Matoušek [Mat91b] zur Lösung des *slope selection*-Problems instrumentalisierte Algorithmus zum Zählen von Inversionen kann mit der gleichen Zeitkomplexität als *in-place*-Variante realisiert werden. Die initiale Sortierung bezüglich der $<_{b_-}$-Ordnung kann durch *heapsort* erfolgen (siehe Abschnitt 4.1.2). Die Laufzeit des Zählens innerhalb eines Verschmelzungsschrittes, wie in Algorithmus 5.4 dargestellt, ist linear in der Größe der zu verschmelzenden Eingabeteile. Zusätzlich zur Eingabe vorzuhalten sind nur die Zählvariable c, die Zeiger i_1 und i_2 sowie die übergebenen Aufrufparameter. Für das nachfolgende tatsächliche Verschmelzen (d. h. für die hierfür notwendigen Positionsänderungen von Elementen) kann ein *in-place* Verschmelzungs-Algorithmus mit linearer Laufzeit verwendet werden, etwa derjenige von Geffert *et al.* [GKP00], siehe auch Abschnitt 4.1.2.

Zusätzlich ist folgendes zu beachten: Da der Algorithmus zum Zählen von Inversionen dem *divide & conquer* -Prinzip folgt und eine Rekursionstiefe von $\Omega(\log_2 n)$

besitzt, verursacht der zugehörige Rekursionsstack bei konventioneller Realisierung zusätzliche Speicherkosten von $\Omega(\log_2 n)$ Wörtern.

Die *in-place*-Realisierung ähnlicher rekursiver Algorithmen ist bereits von Bose *et al.* [BMM+06] beschrieben worden (vergleiche Abschnitt 4.1.2 sowie Abschnitt 6.2.1 für Beispielrealisierungen von konkreten rekursiven Algorithmen). Im vorliegenden Fall des Inversionen zählenden Algorithmus ist es zudem möglich, den „Rekursionsbaum“ in Form eines an den Blättern beginnenden Breitendurchlaufs abzuarbeiten, so dass das *divide & conquer*-Prinzip iterativ statt rekursiv umgesetzt wird und kein expliziter Rekursionsstack vorgehalten werden muss.

Bemerkung 5.3.11 Letztere Form der Rekursionsrealisierung ist zudem auch dann für das vorliegende Problem speichereffizient realisierbar, wenn der zugehörige „Rekursionsbaum“ nicht vollständig balanciert ist, also die Größe n der Eingabe keiner Zweierpotenz entspricht: Ein Eingabeteil der Mächtigkeit m wird, sofern m keiner Potenz von 2 entspricht, in zwei Teile der Größe $2^{\lfloor \log_2 m \rfloor}$ und einen Teil der Größe $2^{\lceil \log_2 m \rceil} - m$ aufgeteilt. Auf diese Weise wird auf jeder „Rekursionsebene“ [19] des Algorithmus höchstens ein Eingabeteil abgearbeitet, dessen Mächtigkeit nicht einer Potenz von 2 entspricht. Die jeweiligen Aufteilungen der Eingabe ergeben sich dabei deterministisch aus der mit konstantem zusätzlichem Speicher vorzuhaltenden Eingabegröße n.

Somit ergibt sich für die beschriebene *in-place*-Variante dieselbe Laufzeitkomplexität wie für den ursprünglichen Inversionen zählenden Algorithmus:

Lemma 5.3.12 *Das Zählen aller Inversionen zwischen zwei Sortierungen einer n-elementigen (Multi-)Menge ist* in-place *in einer Zeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ möglich.*

Als Folgerung ergibt sich mit Lemma 5.3.6 und unter Beachtung von Bemerkung 5.3.10:

Korollar 5.3.13 *Das Zählen aller Schnittpunkte einer (Multi-)menge von n Geraden in einem Streifen $\langle b, e \rangle$ ist* in-place *in einer Zeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ möglich.*

5.3.3 Speichereffiziente Auswahl und Protokollierung von r Schnittpunkten: Überblick

Zur Erzeugung des Suchstreifens $\langle b', e' \rangle$ wird eine Teilmenge $\mathcal{R}$ von Schnittpunkten aus dem aktuellen Suchstreifen $\langle b, e \rangle$ zufällig ausgewählt, um durch Betrachtung dieser die Position des zu suchenden Elementes interpolieren zu können. Dementsprechend hat eine Realisierung des skizzierten SlopeSelection-Algorithmus in jedem Suchschritt die Auswahl von r Kandidaten, d. h. Schnittpunkten aus dem aktuellen Suchstreifen, zu leisten (Zeile 4 in Algorithmus 5.3).

[19] Aus Gründen der Anschaulichkeit werden die Begriffe der Rekursion und der Rekursionsebene des Inversionen zählenden Algorithmus verwendet, auch wenn im Weiteren ausschließlich die iterative Variante des Algorithmus betrachtet wird.

Konstruktion einer zufälligen Auswahl

Für die Effizienz des *slope selection*-Algorithmus und dessen Analyse ist wesentlich, dass die Auswahl der Schnittpunkte jeweils zufällig erfolgt.[20]

Da die Menge der Schnittpunkte nicht explizit vorliegt und von der Mächtigkeit $\binom{n}{2}$ und damit zu groß ist, um explizit in akzeptabler Zeit aufgezählt zu werden, kann eine zufällige Auswahl nicht in üblicher Weise effizient durchgeführt werden. Dieses Problem lässt sich jedoch durch erneute Adaption des vorgestellten Inversionen zählenden Algorithmus beheben: Da dieser Algorithmus alle Schnittpunkte innerhalb eines beliebigen, vorgegebenen Intervalles in $\mathcal{O}(n \cdot \log_2 n)$ Zeit entdeckt, lässt sich jedem dieser Schnittpunkte sein Rang in der Reihenfolge zuordnen, in der der Inversionen zählende Algorithmus die Schnittpunkte entdeckt.

Folglich genügt es, zunächst r Zahlen zufällig (mit Zurücklegen) aus den Rängen von 1 bis $|\mathcal{I}(b, e)|$ auszuwählen. Die Zufälligkeit der Auswahl der Ränge ist auch bei Verwendung von nur konstantem zusätzlichem Speicher zu gewährleisten, wie im Beweis von Lemma 5.2.4 ausgeführt. Die so generierten Zahlen werden in einer Liste $\mathcal{D}_R$ abgelegt. Anschließend können durch Verwendung des Inversionen zählenden Algorithmus genau die Schnittpunkte mit den in $\mathcal{D}_R$ vermerkten Rängen (bezüglich der Reihenfolge ihrer Entdeckung durch den Algorithmus) in einer Datenstruktur $\mathcal{D}_I$ protokolliert werden.

Diese Adaption des Inversionen zählenden Algorithmus, wie sie bereits von Matoušek [Mat91b] beschrieben wurde, ist in Algorithmus 5.5 skizziert. Die *in-place*-Realisierungen der Datenstrukturen $\mathcal{D}_R$ und $\mathcal{D}_I$ werden in den Abschnitten 5.3.4 und 5.3.5 näher erläutert.

Bemerkung 5.3.14 Die randomisierte Auswahl eines Suchschrittes muss eventuell modifiziert werden, wenn viele Schnittpunkte auf dem linken Rand $x = b$ des Streifens liegen;[21] für diesen Fall bietet es sich an, die Punkte auf dem Rand separat zu zählen, was in linearer Zeit durch Betrachtung der ersten Komponente der $<_{b_+}$-Sortierung, bzw. der $<_{e_-}$-Sortierung der Geraden möglich ist, vergleiche wiederum [DMN92, Kapitel 5.1]. Ergibt die Zählung, dass der zu suchende k-te Schnittpunkt auf einem Intervallrand liegt, kann die Suche beendet und ein beliebiger dieser Schnittpunkte zurückgegeben werden. Andernfalls kann im folgenden Suchschritt der beidseitig offene Streifen als Suchdomäne verwendet werden (für die Behandlung offener und halboffener Streifen vergleiche Bemerkung 5.3.10).

[20] Geschähe die Auswahl nicht zufällig, sondern würde einem festen Muster folgen, wäre die gewünschte erwartete Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ nicht für alle Eingaben zu garantieren,vergleiche auch [MMN98, Kapitel 3]. Damit wäre das in diesem Kapitel angestrebte Ziel verfehlt, da sich für randomisierte Algorithmen die Komplexität ihrer erwarteten Laufzeit als die erwartete Laufzeit für die ungünstigste aller mögliche Eingaben bestimmt.

[21] Es besteht sonst die Möglichkeit, dass dass der neue Suchstreifen nur aus dem linken Rand des bisherigen Suchstreifens besteht; liegt der gesuchte Punkt nicht auf diesem Rand, ist im aktuellen Suchschritt keinerlei Verfeinerung des Suchstreifens zu erzielen und in entarteten Fällen könnte der Algorithmus nicht terminieren.

Algorithmus 5.5 Algorithmus CountAndRecord($\mathtt{A}, b_1, b_2, c_1, c_2, \langle b, e\rangle, c$) erhöht den Zähler für die bisher gefundenen Schnittpunkte in $\langle b, e\rangle$ um die Anzahl der Schnittpunkte in $\langle b, e\rangle$, die von je einer Geraden aus $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und einer Geraden aus $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ gebildet werden. Zusätzlich werden Schnittpunkte, deren Ränge in $\mathcal{D}_R$ vermerkt sind, in $\mathcal{D}_I$ protokolliert.

Vorbedingung: $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ sind bzgl. $<_{e\_}$ sortiert; $l_1 \in \mathtt{A}[b_1, \ldots, b_1 + c_1] \wedge l_2 \in \mathtt{A}[b_2, \ldots, b_2 + c_2] : l_1 <_{b\_} l_2$

Nachbedingung: $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ sind bzgl. $<_{e\_}$ sortiert.

```
1:  i_1 := 0; i_2 := 0.
2:  for i = 0 to c_1 + c_2 + 1    /* Das i-te Element in sortierter Reihenfolge ist A[i_1]
    oder A[i_2]. */  do
3:    if A[b_1 + i_1] <_e_ A[b_2 + i_2]    /* ♯(Durch A[b_1 + i_1] bedingte Inversionen) =
      ♯(Elemente in A[b_2, ..., b_2 + c_2] vor A[b_1 + i_1]) bzgl. <_e_). */ then
4:      for each Rang ρ in D_R ∩ [c + 1, ..., c + i_2 + 1] do
5:        Aktualisiere D_I: Füge das Paar (A[b_1 + i_1], A[b_2 + ρ − c − 1]) als den
          Schnittpunkt mit Rang ρ hinzu.
6:      end for
7:      c := c + i_2 + 1.                    /* Zähle Schnittpunkte. */
8:      i_1 := i_1 + 1.                      /* Inkrementiere i_1. */
9:    else
10:     i_2 := i_2 + 1.                      /* Inkrementiere i_2. */
11:   end if
12: end for
```

Zur Laufzeit von Matoušeks Konstruktionsalgorithmus

Stehen $\Omega(r)$ Wörter an zusätzlichem Speicher zur Verfügung, kann die zufällige Konstruktion und Speicherung der r Ränge in $\mathcal{O}(r \log_2 n)$ Zeit erfolgen, was die Kosten für die Sortierung dieser Auswahl einschließt. Dabei belaufen sich die zusätzlichen Laufzeitkosten der Variante CountAndRecord gegenüber dem Algorithmus CountInversions (Algorithmus 5.4) auf $\mathcal{O}(r)$; es ist zu beachten, dass der Zugriff auf den Rang (in $\mathcal{D}_R$) des jeweils nächsten zu protokollierenden Schnittpunktes wegen der vorab erfolgten Sortierung von $\mathcal{D}_R$ in konstanter Zeit erfolgen kann.

Strategien zur effizienten Speichernutzung bei der Schnittpunktprotokollierung

Herausforderungen Die verbleibende Schwierigkeit, eine *in-place*-Variante von Matoušeks Algorithmus mit unveränderter asymptotischer Laufzeit zu entwickeln, liegt in der Protokollierung einer Menge $\mathcal{R}$ von r zufälligen Schnittpunkten, ohne hierfür mehr als konstant viel zusätzlichen Speicher zu verwenden. Dies kann durch die in Kapitel 4.1.1 beschriebene Kodierungstechnik erfolgen, d. h. die Protokollinformationen können implizit durch lokale Permutationen der Eingabe vorgehalten werden, sofern die Größe r der Auswahl $\mathcal{R}$ genügend klein gewählt ist. Dies entspricht der Vorgehensweise für die randomisierte Suche durch Interpolation innerhalb einer Menge von Zahlen (vergleiche den Beweis zu Lemma 5.2.4). Eine wesentliche Erschwernis ergibt sich dadurch, dass diese Protokollierung sukzessive während des

Zählens von Inversionen zu erfolgen hat. Problematisch hierbei ist insbesondere, dass der beschriebene Algorithmus zum Zählen von Inversionen — als *mergesort*-Variante — häufig Positionsänderungen innerhalb der Eingabe durchzuführen hat.

Mit Hilfe der nachfolgend beschriebenen Techniken lassen sich sukzessive entwickelnde Protokollinformationen durch Bit-Kodierung einer Eingabe, auf der zugleich Verschmelzungen durchzuführen sind, aufrecht erhalten. Die Zeiteffizienz dieser Techniken ist insoweit gegeben, als dass sie die Kosten des *slope selection*-Algorithmus nicht dominieren und somit die angestrebte erwartete Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ bei Verwendung von nur konstantem zusätzlichem Speicher erreicht wird.

Lösungsansatz Es gilt zu vermeiden, dass Geraden, die zur Kodierung von $\mathcal{R}$ verwendet werden sollen, durch Verschmelzungen verschoben werden: Die kodierenden Eingabeelemente sollten also von den aktuell durch den Algorithmus CountAndRecord (Algorithmus 5.5) zu bearbeitenden Eingabeelementen separiert werden. Um dies sicher zu stellen, unterteilt sich der Ablauf des folgend beschriebenen *in-place*-Algorithmus in drei Phasen: In der ersten Phase werden nur die (Schnittpunkte der) Geraden in der ersten Hälfte $\mathtt{A}[0, \ldots, n/2 - 1]$ des Eingabefeldes verarbeitet und zur Kodierung von Schnittpunktinformationen ausschließlich Geraden aus der zweiten Hälfte $\mathtt{A}[n/2, \ldots, n - 1]$ des Eingabefeldes genutzt.[22]

In der zweiten Phase vertauschen sich die Rollen der beiden Hälften des Eingabefeldes. In der abschließenden dritten Phase wird die oberste Ebene des „Rekursionsbaumes“ des Algorithmus, also die finale Verschmelzung, abgearbeitet, d. h. es werden ausschließlich Schnittpunkte entdeckt, die von je einer Geraden aus der ersten sowie einer Geraden aus der zweiten Hälfte des Eingabefeldes induziert werden. Wie erläutert werden wird, permutiert der Inversionen zählende Algorithmus in dieser abschließenden Phase des Protokollierens keine Eingabeelemente , so dass die kodierten Informationen „sicher“ in der ersten Hälfte der Eingabefeldes vorgehalten und ergänzt werden können.

5.3.4 Speichereffiziente Auswahl und Protokollierung von r Schnittpunkten: Datenstrukturen

Die Unterteilung des Ablaufs des Algorithmus in drei Phasen garantiert, dass während jeder Phase in einem zusammenhängenden (und sortierten) Teil des Eingabefeldes von $n/2$ Elementen keine Verschmelzungen erfolgen. Die folgende Darstellung bezieht sich auf die erste Phase des Algorithmus, so dass dieser „sichere“ Teil die zweite Eingabehälfte $\mathtt{A}[n/2, \ldots, n - 1]$ ist und für die Kodierung von Schnittpunktinformationen genutzt werden kann; die zweite und dritte Phase des Algorithmus wird in Abschnitt 5.3.7 beschrieben. Zur Kodierung einzelner Bits durch einander benachbarte Elemente kann die $<_{b\_}$-Ordnung verwendet werden, die zu Beginn des

[22] Um sicher zu stellen, dass zumindest $\lfloor n/2 \rfloor$ Eingabeunkte als kodierende Elemente verwendet werden können, wird die rekursive Aufteilung der Eingabe durch den Inversionen zählenden Algorithmus wie folgt durchführt: Die erste Aufteilung erfolgt in (nahezu) gleich große Eingabeteile $\mathtt{A}[0, \ldots, n/2 - 1]$ und $\mathtt{A}[n/2, \ldots, n - 1]$. Die weitere Aufteilung dieser beiden Teile kann dann entsprechend den Ausführungen in Abschnitt 5.3.2 erfolgen.

Algorithmus durch Anwendung von *heapsort* (siehe Abschnitt 4.1.2) auf der Eingabe etabliert wird. Bilden zwei Geraden g und h in der Eingabe Bit-Nachbarn und gilt $g <_{b\_} h$, so kodiert die Permutation gh eine binäre 0 und die Permutation hg eine binäre 1 (zur näheren Erläuterung dieser Kodierungstechnik siehe Kapitel 4.1.1).

Bemerkung 5.3.15 Es ist zu beachten, dass die beschriebene Kodierung durch Geraden voraussetzt, dass der zur Kodierung verwendete Teil der Eingabe keine vertikalen Geraden und keine Duplikate enthält. Ersteres ist durch die Gestalt der ursprünglichen Eingabe gesichert (siehe Bemerkung 5.3.4). Die Separierung aller Duplikate von dem zur Kodierung verwendeten Eingabeteil kann *in-place* und in $\mathcal{O}(n \cdot \log_2 n)$ Zeit erfolgen: Zunächst sind die Geraden lexikografisch zu sortieren, etwa per *heapsort*; anschließend können die Duplikate durch einen linearen Durchlauf erkannt und durch den Teilmengenextraktionsalgorithmus von Bose *et al.* [BMM$^+$06] vom verbleibenden Eingabeteil separiert werden; die verringerte Größe des zur Kodierung verwendeten Eingabeteils verringert entsprechend die Anzahl der kodierbaren Informationen, vergleiche Bemerkung 4.1.3. Eine Analyse für duplikatbehaftete Eingaben wird nach Beschreibung des *slope selection*-Algorithmus nachgeliefert.

Mit Hilfe der beschriebenen Kodierungstechnik werden drei Datenstrukturen $\mathcal{D}_R$, $\mathcal{D}_L$ und $\mathcal{D}_I$ aufrechterhalten, die Informationen über die noch zu protokollierenden und die bereits protokollierten Schnittpunkte vorhalten:

Datenstruktur $\mathcal{D}_R$: Diese Datenstruktur ist der Art nach eine „sortierte Liste“ und speichert die r zu Beginn des Algorithmus zufällig gewählten Zahlen, die den Rängen der auszuwählenden Schnittpunkte in der Reihenfolge ihrer Entdeckung durch den Algorithmus entsprechen. Da diese Zahlen sämtlich aus dem begrenzten Bereich $[0, \ldots, n^2 - 1]$ stammen, kann jede dieser Zahlen durch $2 \cdot \log_2 n$ binäre Informationen dargestellt werden, also durch (lokale) Permutation von insgesamt $4 \cdot r \cdot \lceil \log_2 n \rceil$ hierfür reservierten Eingabeelementen.

Datenstruktur $\mathcal{D}_L$: Diese Datenstruktur wird ebenfalls als eine „sortierte Liste“ verwendet. Sie protokolliert (Referenzen auf) alle Geraden, die zu bereits protokollierten Schnittpunkten beitragen. Dabei wird jede dieser bis zu $2 \cdot r$ vielen Geraden in $\mathcal{D}_L$ durch ihre aktuelle Position innerhalb des Eingabefeldes referenziert, d. h. durch eine Zahl aus dem Bereich $[0, \ldots, n - 1]$: Eine Zahl i ist genau dann in $\mathcal{D}_L$ vermerkt, wenn die in $\mathtt{A}[i]$ gespeicherte Gerade an mindestens einem der bereits protokollierten Schnittpunkte beteiligt ist.[23] Die bis zu $2 \cdot r$ vielen Referenzen in $\mathcal{D}_L$ können durch Permutation von $4 \cdot r \cdot \lceil \log_2 n \rceil$ dargestellt werden.

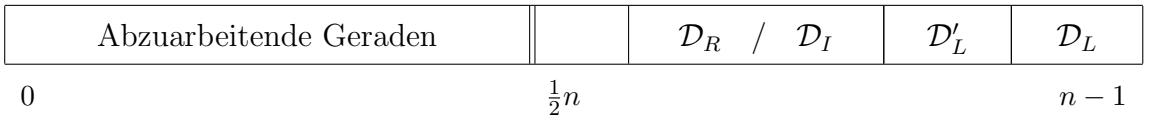
Datenstruktur $\mathcal{D}_I$: Diese Datenstruktur wird wie eine „fortlaufende Liste“ verwendet und speichert alle protokollierten Schnittpunkte in der Reihenfolge, in der der Inversionen zählende Algorithmus sie entdeckt. Ein Schnittpunkt s wird in $\mathcal{D}_I$ durch zwei Referenzen l_1 und l_2 in die Datenstruktur $\mathcal{D}_L$ repräsentiert.

[23] Da eine Gerade an mehr als einem Schnittpunkt beteiligt sein kann, folgt, dass die aktuelle Anzahl der Elemente in $\mathcal{D}_L$ sich nicht direkt aus der Anzahl der bereits protokollierten Schnittpunkte ergibt. Daher wird die aktuelle Anzahl der in $\mathcal{D}_L$ vermerkten Referenzen explizit (aber durch Verwendung von nur $\mathcal{O}(1)$ zusätzlichen Speicherwörtern) vorgehalten.

Die beiden referenzierten Referenzen $l_1 = \mathcal{D}_L[j_1]$ und $l_2 = \mathcal{D}_L[j_2]$ in $\mathcal{D}_L$ zeigen auf die beiden Geraden $\mathtt{A}[i_1]$ und $\mathtt{A}[i_2]$, die den Schnittpunkt s induzieren. Für diese Form der Kodierung eines Schnittpunktes bedarf es $4 \cdot \lceil \log_2 n \rceil$ Bits; für die Kodierung aller r Schnittpunkte werden also $8 \cdot r \cdot \lceil \log_2 n \rceil$ Eingabeelemente benötigt.

Analyse des impliziten Speicherbedarfs Der Bedarf an kodierenden Eingabeelementen für die genannten Datenstrukturen kann in folgender Weise optimiert werden: ein in $\mathcal{D}_I$ neu protokollierter Schnittpunkt impliziert, dass der entsprechende in $\mathcal{D}_R$ vermerkte Rang nicht mehr vorgehalten werden muss. Da die Informationen in der Datenstruktur $\mathcal{D}_R$ mit Auffinden der in $\mathcal{D}_R$ festgelegten Schnittpunkte also obsolet sind, können dieselben Eingabeelemente für die Kodierung sowohl von $\mathcal{D}_R$ als auch von $\mathcal{D}_I$ verwendet werden. Hierbei wird ausgenutzt, dass $\mathcal{D}_R$ linear fortlaufend abgebaut und $\mathcal{D}_I$— induziert jeweils durch den Abbau in $\mathcal{D}_R$— linear fortlaufend gefüllt wird.

Aus Gründen, die in Abschnitt 5.3.5 ausgeführt werden, wird zusätzlich nicht nur Platz für die Datenstruktur $\mathcal{D}_L$, sondern auch für eine temporäre Kopie $\mathcal{D}'_L$ von $\mathcal{D}_L$ benötigt. Der resultierende Gesamtbedarf an Eingabeelementen für die Kodierung aller Schnittpunktinformationen beläuft sich somit auf $8 \cdot r \cdot \lceil \log_2 n \rceil$ Elemente. Das Eingabefeld ist während der ersten Phase des Ablaufs des Algorithmus wie folgt partitioniert:



Die beschriebene Strategie, die zu protokollierenden Schnittpunktinformationen kodiert in einer Hälfte der Eingabeelemente vorzuhalten, impliziert eine Beschränkung für die Größe der protokollierbaren Auswahl $\mathcal{R}$ an Schnittpunkten. Diese Beschränkung ist als Konkretisierung von Beobachtung 4.1.2 wie folgt quantifizierbar:

Korollar 5.3.16 *Die Verwendung von $n/2$ geordneten Elementen zur Kodierung erlaubt die Protokollierung durch die beschriebenen Datenstrukturen von bis zu und nicht mehr als $r = n/(32 \cdot \lceil \log_2 n \rceil)$ Schnittpunkten.*

Für hinreichend mächtige Eingabemengen kann also $r = \sqrt{n}$ gewählt werden. Für Eingaben von geringerer Mächtigkeit ist eine entsprechende Verringerung von r (um einen konstanten Faktor) erforderlich.

Zur Motivation für die verwendete Indirektion Das Design und die Funktionsweise der Datenstrukturen $\mathcal{D}_R$, $\mathcal{D}_L$ und $\mathcal{D}_I$ für die Schnittpunktprotokollierung wird im Folgenden konkreter beschrieben. Dieses Design bedarf in insbesondere einem Punkte einer Rechtfertigung: Dies ist die verwendete (doppelte) Indirektion, genauer das Protokollieren eines Schnittpunktes durch Verweise in die Struktur $\mathcal{D}_L$ statt *direkt* auf Elemente der Eingabe.

Da die Protokollierung der Schnittpunkte während der Abarbeitung des modifizierten Inversionen zählenden Algorithmus (vergleiche Algorithmus 5.5) erfolgt, unterliegen die Geraden der Eingabe Positionsveränderungen im Zuge von Verschmelzungen. Da dieser Algorithmus eine *mergesort*-Erweiterung ist, erfolgen im allgemeinen Fall sogar $\mathcal{O}(n \cdot \log_2 n)$ viele solcher Positionsänderungen. Um für die Protokollierung von $\mathcal{R}$ eine Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ zu erreichen, muss eine jede dieser Positionsänderungen in jeweils bezüglich der Eingabegröße n konstanter Zeit an die Protokollierung der zufällig ausgewählten Schnittpunkte propagiert, d. h. alle Referenzen auf die umpositionierte Gerade aktualisiert werden.

In Abbildung 5.4 ist diese Problematik und die Lösung durch die beschriebene Verwendung von Indirektion skizziert: Findet der Inversionen zählende Algorithmus einen Schnittpunkt der Geraden g (etwa denjenigen mit der Gerden h), impliziert dies im Allgemeinen, dass im Zuge der Verschmelzung in die $<_{e\_}$-Ordnung die Gerade g neu positioniert werden muss. Durch Verwendung der Datenstruktur $\mathcal{D}_L$ wird es möglich, diese Positionsveränderung an alle protokollierten Schnittpunkte in $\mathcal{R}$, an denen die Gerade g beteiligt ist, zu propagieren, indem nur *ein* Zeiger in der Struktur $\mathcal{D}_L$ aktualisiert wird. Zusätzlich ist die Struktur $\mathcal{D}_L$ so konzipiert, dass besagter zu aktualisierender Zeiger in konstanter Zeit identifiziert werden kann.

Zum Nachweis von Korrektheit und Laufzeiteffizienz der skizzierten Strategie werden zunächst Realisierungen für die auf den vorgestellten Datenstrukturen durchzuführenden Operationen vorgestellt und deren Kosten analysiert. In Abschnitt 5.3.5 wird dann auf die Techniken eingegangen, die die Konsistenz und Korrektheit der Schnittpunktinformationen in $\mathcal{D}_L$ und $\mathcal{D}_I$ während des Zählens von Inversionen sichern.

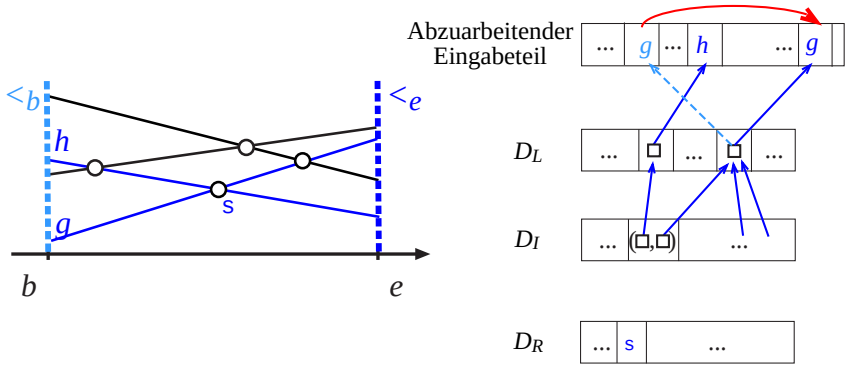


Abbildung 5.4: Aktualisieren der Referenzen auf eine während eines Verschmelzungsvorgangs neu zu positionierende Gerade.

Initialisierung und Aufrechterhaltung der Datenstruktur $\mathcal{D}_R$

Die Datenstruktur $\mathcal{D}_R$ wird initialisiert, indem zunächst r natürliche Zahlen aus dem Bereich $[0, \ldots, n^2 - 1]$ iterativ und jeweils zufällig (d. h. mit Wiederholungen) generiert und in $\mathcal{D}_R$ kodiert abgelegt werden. Anschließend werden diese Zahlen in $\mathcal{D}_R$

per *heapsort* aufsteigend sortiert, was wegen deren Darstellung als Permutationen von Eingabeelementen $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$ Zeit erfordert, siehe Korollar 4.1.5.

Die Datenstruktur $\mathcal{D}_R$ wird linear, d. h. entsprechend der etablierten Sortierung, abgebaut. Die Zahl in $\mathcal{D}_R$, die den Rang des nächsten zu protokollierenden Schnittpunktes angibt, wird in konstantem Zusatzspeicher vorgehalten. Ebenso wird als Information über den aktuellen Status der Abarbeitung von $\mathcal{D}_R$ die Anzahl der bereits abgearbeiteten Zahlen in $\mathcal{D}_R$, d. h. die Anzahl der bereits protokollierten Schnittpunkte, in konstantem zusätzlichem Speicher aufrecht erhalten. Jede der insgesamt r Aktualisierungen von $\mathcal{D}_R$ erfordert die Dekodierung des Ranges des nächsten zu protokollierenden Schnittpunktes in $\mathcal{D}_R$ und damit $\mathcal{O}(\log_2 n)$ Zeit. Somit ist der globale Zeitaufwand für das Initialisieren der Datenstruktur $\mathcal{D}_R$ und für das Aufrechterhalten der benötigten Statusinformationen von der Komplexität $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$.

Initialisierung und Aufrechterhaltung der Datenstruktur $\mathcal{D}_L$

In der Datenstruktur $\mathcal{D}_L$ werden alle Geraden (durch Referenzen auf diese) vermerkt, die zu bereits gefundenen Schnittpunkten, deren Rang in $\mathcal{D}_R$ vermerkt war, beitragen. Diese (Referenzen auf) Geraden werden in $\mathcal{D}_L$ sortiert vorgehalten, und zwar entsprechend der $<_{b\_}$-Ordnung der referenzierten Geraden. Initial ist $\mathcal{D}_L$ somit leer und das Einfügen einer neuen Referenz auf eine Gerade kann analog dem klassischen *insertion sort*-Verfahren erfolgen (siehe Cormen *et al.* [CLR01, Kapitel 2]), d. h. alle dem einzufügenden Element in der $<_{b\_}$-Ordnung nachfolgenden Elemente in $\mathcal{D}_L$ werden um einen Platz in $\mathcal{D}_L$ nach rechts verschoben. Duplikate werden hierbei ignoriert, d. h. nicht eingefügt. Der Zeitaufwand für das Einfügen einer Referenz auf eine Gerade in die Struktur $\mathcal{D}_L$ ist somit in $\mathcal{O}(r \cdot \log_2 n)$, so dass sich die globalen Kosten für das Aufrechterhalten der Datenstruktur $\mathcal{D}_L$ zu $\mathcal{O}(r^2 \cdot \log_2 n)$ Zeit summieren.[24] Das Entfernen von Referenzen aus $\mathcal{D}_L$ wird generell nicht erforderlich.

Da die Elemente in der Datenstruktur $\mathcal{D}_I$ Referenzen sind, die in die Datenstruktur $\mathcal{D}_L$ zeigen, muss jede Aktualisierung der Datenstruktur $\mathcal{D}_L$ wie folgt an die Datenstruktur $\mathcal{D}_I$ propagiert werden: Sei das Element x gerade in $\mathcal{D}_L$ eingefügt worden, und zwar an die j-te Position in $\mathcal{D}_L$, d. h. als j-te Gerade bzgl. der $<_{b\_}$-

[24] Alternativ kann eine effizientere Methode der Aufrechterhaltung der Sortierung unter sukzessivem Einfügen verwendet werden: *Library sort* [BFM04] ist eine stabil sortierende *insertion sort*-Variante, die eine erwartete Laufzeit von $\mathcal{O}(r \cdot \log_2 r)$ für eine r-elementige Eingabe besitzt. Wie *insertion sort* ist auch diese Variante ein *on-line*-Algorithmus, d. h. sie bedarf keiner Vorab-Kenntnis über die noch einzufügenden Elemente (abgesehen von dem jeweils als nächstes einzufügenden Element). Der ursrprüngliche *library sort*-Algorithmus permutiert die Menge der einzufügenden Elemente vorab in zufälliger Weise, um die angegebene Laufzeit für *jede* Eingabe erwarten zu können. Diese Permutation ist für den Einsatz von *library sort* innerhalb des *slope selection*-Algorithmus nicht notwendig: Die Zufälligkeit der einzufügenden Elemente ist bereits durch die zufällige Auswahl der r zu protokollierenden Schnittpunkte gesichert. Die erwartete Laufzeit des *library sort*-Algorithmus erhöht sich durch die Notwendigkeit des Kodierens und Dekodierens von Elementen in $\mathcal{D}_L$ um einen logarithmischen Faktor auf $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$. *Library sort* garantiert seine günstige erwartete Gesamtlaufzeit dadurch, dass an geeigneten Stellen freie Speicherplätze zwischen den bereits eingefügten Elementen bereitgestellt werden, so dass eine Einfügung nur zu erwartet amortisiert $\mathcal{O}(\log_2 r)$ vielen Positionsveränderungen führt. Diese Strategie bedingt, dass der Speicherbedarf für die Datenstruktur $\mathcal{D}_L$ durch die Verwendung von *library sort* um einen Faktor $1 + \epsilon$ wächst, wobei beispielsweise $\epsilon = 1$ gewählt werden kann, ohne dass dies die asymptotische Laufzeitkomplexität erhöht.

Sortierung der in $\mathcal{D}_L$ referenzierten Geraden. Ist x bereits in $\mathcal{D}_L$ vermerkt, so ist keine Propagation notwendig, da in $\mathcal{D}_L$ generell keine Duplikate gespeichert werden, $\mathcal{D}_L$ im vorliegenden Fall also nicht modifiziert werden muss. Andernfalls, d. h. falls x noch nicht in $\mathcal{D}_L$ vermerkt war, führt das Einfügen von x in $\mathcal{D}_L$ (durch *insertion sort*) dazu, dass alle an Position j oder höher in $\mathcal{D}_I$ gespeicherten Elemente sich fortan an der nächst höheren (relativ zu ihrer bisherigen) Position befinden.

Diese Positionsveränderungen von Referenzen in $\mathcal{D}_L$ an die Datenstruktur $\mathcal{D}_I$ zu propagieren, erfordert einen kompletten Durchlauf durch alle (bis zu $2r$) Referenzen in $\mathcal{D}_I$ und das entsprechende Aktualisieren all jener Referenzen, die auf Positionen in $\mathcal{D}_L$ mit Index $i \leq j$ zeigen. Ein solcher Durchlauf kostet jeweils $\mathcal{O}(r \cdot \log_2 n)$ Zeit, da jeweils $\mathcal{O}(r)$ Referenzen dekodiert werden müssen. Der globale Zeitaufwand für das Propagieren von Änderungen in $\mathcal{D}_L$ entspricht also in der asymptotischen Analyse den globalen Kosten von $\mathcal{O}(r^2 \cdot \log_2 n)$ für das Aktualisieren innerhalb von $\mathcal{D}_L$ unter Verwendung von *insertion sort*.[25]

Initialisierung und Aufrechterhaltung der Datenstruktur $\mathcal{D}_I$

Die Datenstruktur $\mathcal{D}_I$ speichert die bislang gefundenen Schnittpunkte durch jeweils zwei Referenzen in die Datenstruktur $\mathcal{D}_L$, deren konsistente Aktualisierung bereits in vorigen Abschnitt 5.3.4 analysiert wurde. Um nun einen soeben entdeckten, zu protokollierenden Schnittpunkt von zwei Geraden g_1 and g_2 in $\mathcal{D}_I$ einzufügen, werden zunächst (Referenzen auf) g_1 and g_2 in die $<_{b\_}$-Sortierung der Geraden aus $\mathcal{D}_L$ eingefügt (wie im Abschnitt 5.3.4 beschrieben). Nachfolgend wird das Paar (i, j), das diese beiden Geraden referenziert, hinter dem bislang letzten Eintrag in $\mathcal{D}_I$ vermerkt.

Die Laufzeitkosten für das Einfügen in $\mathcal{D}_I$ belaufen sich auf insgesamt $\mathcal{O}(r \cdot \log_2 n)$ zusätzlich zu den analysierten Kosten für die Aktualisierungen in $\mathcal{D}_L$ (und für deren Propagation an $\mathcal{D}_I$).

Aus der Beschreibung der Funktionsweise und der Kostenanalyse der impliziten Datenstrukturen $\mathcal{D}_R$, $\mathcal{D}_L$ und $\mathcal{D}_I$ ergibt sich, zusammen mit den Ausführungen des folgenden Abschnitts, die Aussage des folgenden Lemmas:

Lemma 5.3.17 *Die globalen Laufzeitkosten für das Initialisieren und Aktualisieren der Datenstrukturen $\mathcal{D}_R$, $\mathcal{D}_L$ und $\mathcal{D}_I$ zur* in-place*-Protokollierung von r Schnittpunkten, die aus den durch n Geraden in der Ebene induzierten Schnittpunkten zufällig gewählt sind, belaufen sich auf $\mathcal{O}(r^2 \cdot \log_2 n)$.*

5.3.5 Speichereffiziente Auswahl und Protokollierung von r Schnittpunkten: Ablauf

Das Aufrechterhalten und das Zusammenspiel der Datenstrukturen $\mathcal{D}_L$ und $\mathcal{D}_I$ während des Protokollierens von Schnittpunktinformationen bedarf in einigen Punkten hinsichtlich Korrektheit und Laufzeitverhalten einer genaueren Erläuterung.

[25] Bei Verwendung von *library sort* (vergleiche Fußnote 24) ist die erwartete Laufzeit in $\mathcal{O}(r \cdot \log_2 n \cdot \log_2 r)$: Eine einzelne Aktualisierung in $\mathcal{D}_L$ sowie deren Propagierung nach $\mathcal{D}_I$ kosten armortisiert im erwarteten Falle jeweils $\mathcal{O}(\log_2 n \cdot \log_2 r)$ erwartete Zeit.

Verschmelzen von zwei Teilen der Eingabe

Wie beschrieben wird eine Variante des Inversionen zählenden Algorithmus zur Protokollierung der Schnittpunkte in $\mathcal{R}$ verwendet. Dieser Algorithmus — als Erweiterung des *mergesort*-Algorithmus — verschmilzt zwei zu bearbeitende Eingabeteile $\mathtt{A}_1$ und $\mathtt{A}_2$ in einen komplett $<_{e\_}$-sortierten Eingabeteil $\mathtt{A}_{1\cup 2}$. Da $\mathtt{A}_1$ und $\mathtt{A}_2$ vor der Verschmelzung bereits $<_{e\_}$-sortiert sind, kann diese Verschmelzung *in-place* und in linearer Zeit (bezüglich der Anzahl der Elemente in $\mathtt{A}_1 \cup \mathtt{A}_2$) erfolgen, etwa durch den Algorithmus von Geffert *et al.* [GKP00].

Die durch diese Verschmelzung notwendigen Aktualisierungen der Referenzen in $\mathcal{D}_L$ sind aber bereits in der dem Verschmelzen von $\mathtt{A}_1$ und $\mathtt{A}_2$ vorangegangenen Zähl- und Protokollierungsphase durch CountAndRecord (Algorithmus 5.5) berechnet worden: CountAndRecord arbeitet iterativ die Elemente in $\mathtt{A}_1$ und $\mathtt{A}_2$ ab, und zwar in der Reihenfolge ihrer *neuen* Position im verschmolzenen Resultat $\mathtt{A}_{1\cup 2}$; während der i-ten Iteration innerhalb einer Abarbeitung von CountAndRecord wird also das i-te Element (bezeichnet mit g) der $<_{e\_}$-Sortierung von $\mathtt{A}_1 \cup \mathtt{A}_2$ bestimmt (Zeile 2 in Algorithmus 5.5). In dieser i-ten Iteration ist in $\mathcal{D}_L$ die Referenz auf die Gerade g zu modifizieren, sofern g mindestens einen der bisher in $\mathcal{D}_I$ protokollierten Schnittpunkte mit induzierte. Ist dies der Fall, so besteht die notwendige Modifikation aus dem Ersetzen der (einzigen) Referenz in $\mathcal{D}_L$ auf die bisherige Position der Geraden g durch die Referenz auf die Position von g *nach* der Verschmelzung von $\mathtt{A}_1$ und $\mathtt{A}_2$, die im Anschluss an die CountAndRecord-Abarbeitung erfolgt. Aus noch zu erläuternden Konsistenzgründen wird diese vorausgreifende Modifikation von Zeigern auf während der Verschmelzung umzupositionierende Geraden nicht auf $\mathcal{D}_L$ durchgeführt, sondern auf einer Kopie $\mathcal{D}'_L$ von $\mathcal{D}_L$. Hierfür ist zu Beginn der Abarbeitung einer neuen Rekursionsebene des Inversionen zählenden Algorithmus der Inhalt von $\mathcal{D}_L$ nach $\mathcal{D}'_L$ zu kopieren, was jeweils nicht mehr als $\mathcal{O}(r \cdot \log_2 n)$ Zeit in Anspruch nimmt.

Auffinden der Zeiger in $\mathcal{D}_L$ auf zu verschmelzende Geraden

Um effizient die eventuell existierende Referenz in $\mathcal{D}_L$ auf eine Gerade g, deren Positionsänderung gerade durch CountAndRecord ermittelt wurde, aufzufinden, kann Beobachtung 5.3.8 ausgenutzt werden: Aus dieser Beobachtung und der $<_{b\_}$-Sortierung der Einträge in $\mathcal{D}_L$ folgt, dass diejenigen Einträge, die Geraden aus $\mathtt{A}_1 \cup \mathtt{A}_2$ referenzieren, einen zusammenhängenden Teil $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ von $\mathcal{D}_L = \mathcal{D}_L[0, \ldots, 2r-1]$ bilden. Die Situation in $\mathcal{D}_L$ vor der Verschmelzung von $\mathtt{A}_1$ und $\mathtt{A}_2$ stellt sich folgendermaßen dar:

Vor dem Verschmelzen zweier Eingabeteile $\mathtt{A}_1$ and $\mathtt{A}_2$ durch CountAndRecord ist es daher nützlich, die Indizes j_{lo} und j_{hi} zu kennen, d. h. sie vorab zu berechnen. Dies ist in iterativer Weise möglich; durch die Beobachtungen 5.3.7 und 5.3.8 ist gewährleistet, dass für eine Verschmelzung der nächsten beiden Eingabeteile $\mathtt{A}_3$ und

$\mathtt{A}_4$ die beiden Indizes j_{lo} und j_{hi} effizient aktualisiert werden können: Der neue Wert von j_{lo} entspricht dem bisherigen Wert von j_{hi} und der neue Wert von j_{hi} kann durch lineares Suchen in $\mathcal{D}_L$, beginnend bei $\mathcal{D}_L[j_{\text{hi}}]$, bestimmt werden: Genauer bestimmt sich der Wert von j_{hi} als der Index in $\mathcal{D}_L$ der letzten (Referenz auf eine) Gerade, die bezüglich der $<_{b\_}$-Sortierung „höher " als die „höchste" Gerade g_{max} in $\mathtt{A}_3 \cup \mathtt{A}_4$ ist.[26] Die Laufzeitkosten für diese Vorarbeiten für das Verschmelzen zweier Eingabeteile ergeben sich daher wie folgt:

Lemma 5.3.18 *Die globalen Laufzeitkosten für das Bestimmen der Indizes j_{lo} and j_{hi} sind in $\mathcal{O}(n \cdot \log_2 n + r \cdot \log_2^2 n)$.*

Beweis: Die Kostenanalyse erfolgt pro „Rekursionsebene" des CountAndRecord-Algorithmus. Auf jeder dieser Ebenen wird die zu bearbeitende Eingabe komplett durchlaufen: Für jedes benachbarte Paar $(\mathtt{A}_1, \mathtt{A}_2)$ von zu verschmelzenden Eingabeteilen ist die „höchste" Gerade g_{max} zu bestimmen. Die resultierenden Kosten sind pro „Rekursionsebene" linear in der Eingabe, d. h. global in $\mathcal{O}(n \cdot \log_2 n)$. Ebenso wird die Datenstruktur $\mathcal{D}_L$ in jeweils linearer Zeit durchlaufen und jede in $\mathcal{D}_L$ referenzierte Gerade mit einer Geraden g_{max} verglichen. Hierfür sind alle maximal $\mathcal{O}(r)$ vielen Einträge in $\mathcal{D}_L$ zu dekodieren. Da das Dekodieren jeweils $\mathcal{O}(\log_2 n)$ Zeit kostet, belaufen sich die Kosten auf $\mathcal{O}(r \cdot \log_2 n)$ pro „Rekursionsebene" und insgesamt auf $\mathcal{O}(r \cdot \log_2^2 n)$. Zusätzlich wird nur $\mathcal{O}(\log_2 n)$ Zeit für jedes Paar $(\mathtt{A}_1, \mathtt{A}_2)$ von benachbarten zu verschmelzenden Eingabeteilen, von deren Geraden keine einzige in $\mathcal{D}_L$ referenziert wird, erforderlich. Diese Zeit ist aufzuwenden, um festzustellen, dass die nächste in $\mathcal{D}_L$ referenzierte Gerade außerhalb von $\mathtt{A}_1 \cup \mathtt{A}_2$ liegt. Da insgesamt nur $\mathcal{O}(n)$ solcher Paare $(\mathtt{A}_1, \mathtt{A}_2)$ verschmolzen werden, folgt die Behauptung. □

Aktualisieren der relevanten Referenzen in $\mathcal{D}_L$

Die Teilliste $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$, die den zu verschmelzenden Eingabeteilen $\mathtt{A}_1$ und $\mathtt{A}_2$ entspricht und deren begrenzende Indizes nach Lemma 5.3.18 effizient bestimmt werden können, wird vor der Verschmelzung von $\mathtt{A}_1$ und $\mathtt{A}_2$ bezüglich der $<_{e\_}$-Ordnung sortiert. Ebenso wird mit $\mathcal{D}'_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ verfahren. Auf diese Weise können während des Durchlaufens durch die $<_{e\_}$-sortierten Eingabeteile $\mathtt{A}_1$ und $\mathtt{A}_2$ synchron die Teillisten $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ und $\mathcal{D}'_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ durchlaufen werden.

Für die Synchronisation der Durchläufe wird die (bzgl. $<_{e\_}$) „niedrigste "in $\mathcal{D}_L$ referenzierte Gerade μ vorgehalten, die nicht „niederiger"als die aktuell durch den CountAndRecord-Algorithmus in der i-ten Iteration abzuarbeitende Gerade g ist; die Gerade μ ist damit die nächste durch CountAndRecord abzuarbeitende Gerade, für deren Referenz in $\mathcal{D}_L$ eine Aktualisierung nötig werden kann. Die Gerade μ wird mit konstantem Extraspeicher und nicht explizit vorgehalten, sondern über ihren Index j, so dass $\mathcal{D}_L[j] =: \tau$ auf $\mu = \mathtt{A}[\tau]$ verweist.

[26] Die Beschreibung bezieht sich auf Verschmelzungen der $\mathtt{A}_1$ und $\mathtt{A}_2$ rechts benachbarten Eingabeteile. Sind solche nicht existent, impliziert dies, dass die Verschmelzung auf der nächsten „Rekursionsebene" von CountAndRecord fortgeführt wird; der Index j_{lo} ergibt sich dann als 0, der Index j_{hi} wird durch lineare Suche analog zum geschilderten Fall, aber beginnend bei $\mathtt{A}[0]$, bestimmt.

Die Positionsänderung der Referenz $\mathcal{D}_L[j]$ auf μ wird bearbeitet, wenn die in der Verschmelzung aktuell abzuarbeitende Gerade g die in $\mathtt{A}[j]$ befindliche Gerade ist, so dass $\mathcal{D}_L[j]$ auf g verweist.

Diese Referenz $\mathcal{D}_L[j]$ auf μ hat (nach der Verschmelzung von $\mathtt{A}_1$ und $\mathtt{A}_2$) auf die i-te Gerade des verschmolzenen Eingabeteiles $\mathtt{A}_{1\cup 2}$ zu verweisen. Um diese Aktualisierungen und den synchronisierten Durchlauf in den skizzierten CountAndRecord-Algorithmus (Algorithmus 5.5) zu integrieren, sind die Anweisungen in den Zeilen 8 und 10 zu ergänzen: Jeweils muss der Index der aktuell abzuarbeitenden Gerade g mit dem in $PS[j]$ vermerkten Index verglichen werden. Besteht Gleichheit, so muss der Aktualisierungsbedarf von $\mathcal{D}_L[j]$ vermerkt und der Index j inkrementiert werden. Dieses Vermerken, bzw. das sofortige Ausführen, dieser erst nach der Verschmelzung von $\mathtt{A}_1$ und $\mathtt{A}_2$ gültigen Aktualisierung von $PS[j]$ geschieht nicht (bzw. nicht sofort) in $\mathcal{D}_L$, sondern in der Kopie $\mathcal{D}'_L$ von $\mathcal{D}_L$.

Der zusätzliche Zeitaufwand für die synchronisierten Durchläufe und das Aktualisieren bestehender Referenzen in $\mathcal{D}'_L$ ergibt sich wie folgt: Jede Aktualisierung einer Referenz in $\mathcal{D}'_L$ kostet $\mathcal{O}(\log_2 n)$ Zeit für das Dekodieren der ursprünglichen und für das Kodieren der modifizierten Referenz. Die Anzahl dieser Aktualisierungen ist begrenzt durch $2 \cdot r$ pro „Rekursionsebene“, da der Index j nur inkrementiert wird, also jede Referenz in $\mathcal{D}'_L$ nur einmal pro „Rekursionsebene“ aktualisiert werden kann. Insgesamt resultieren hieraus Laufzeitkosten von $\mathcal{O}(r \cdot \log_2^2 n)$. Dieselbe asymptotische Laufzeitkomplexität ergibt sich für die globalen Kosten für das jeweilige Sortieren des zu den zu verschmelzenden Eingabeteilen $\mathtt{A}_1$ und $\mathtt{A}_2$ assoziierten Teilen der Datenstrukturen $\mathcal{D}_L$ und $\mathcal{D}'_L$ in $<_{e\_}$-Ordnung. Zusätzlich ist zu beachten, dass der dekodierte Wert $\mathcal{D}_L[j]$ mit konstantem Extraspeicher in der Weise vorgehalten werden kann, dass in konstanter Zeit auf diesen zugegriffen werden kann.

Konsistente Aktualisierung von $\mathcal{D}_L$ und $\mathcal{D}_I$ bei Auffinden eines neuen Schnittpunktes

Es bleibt zu beschreiben, wie $\mathcal{D}_L$ und $\mathcal{D}_I$ effizient und konsistent zu aktualisieren sind, wenn der Algorithmus CountAndRecord einen weiteren zu protokollierenden Schnittpunkt entdeckt.

Eine solche Aktualisierung besteht aus dem Einfügen eines Paares von Referenzen in $\mathcal{D}_I$, die auf Elemente in $\mathcal{D}_L$ zeigen (vergleiche Abschnitt 5.3.4). Entsprechend müssen in $\mathcal{D}_L$ bis zu zwei neue Elemente, d. h. Referenzen auf Geraden, eingefügt werden. Das Einfügen einer Referenz τ auf eine Gerade $g = \mathtt{A}[\tau]$ in $\mathcal{D}_L$ ist in mehrerlei Hinsicht problematisch und auch der Grund für das Verwenden der Kopie $\mathcal{D}'_L$ von $\mathcal{D}_L$: Eine neu eingefügte Referenz τ in $\mathcal{D}_L$ muss in die in $\mathcal{D}_L$ herrschende Ordnung eingefügt werden: Da im aktuellen Verschmelzungsschritt nur Schnittpunkte von Geraden aus $\mathtt{A}_1$ mit Geraden aus $\mathtt{A}_2$ gefunden werden können, muss i_j in den zu $\mathtt{A}_1$ und $\mathtt{A}_2$ assoziierten — und damit $<_{e\_}$-sortierten — Teil $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ von $\mathcal{D}_L$ eingefügt werden. Dabei ist zu beachten, dass dieser Teil durch Einfügen von τ um einen Eintrag vergrößert wird und der Wert j_{hi} entsprechend inkrementiert werden muss.

Der Eintrag in $\mathcal{D}_I$, der den neu gefundenen Schnittpunkt repräsentiert, muss aber auf die Position von τ nicht bezüglich der in $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ etablierten

$<_{e\_}$-Sortierung , sondern bezüglich der im Rest von $\mathcal{D}_L$ herrschenden $<_{b\_}$-Sortierung verweisen. Da τ aber bezüglich der $<_{e\_}$-Sortierung in $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ eingefügt wurde, muss die Position in der $<_{b\_}$-Sortierung von $\mathcal{D}_L$ zunächst berechnet werden. Hierfür ist die Anzahl aller in $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ referenzierten Geraden zu bestimmen, die bezüglich der $<_{b\_}$-Ordnung „niedriger“ als die durch τ referenzierte Gerade sind. Ein Durchlauf durch die Referenzen in $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ liefert die zu bestimmende Position j_τ von τ, da in $\mathcal{D}_L$ die Positionen aller Geraden *vor* der Verschmelzung von $\mathtt{A}_1$ mit $\mathtt{A}_2$ vermerkt sind. Es ist zu beachten, dass diese Information nicht aus $\mathcal{D}_L$ zu extrahieren wäre, wenn die Umpositionierungen von Geraden durch das Verschmelzen von $\mathtt{A}_1$ mit $\mathtt{A}_2$ *direkt* in $\mathcal{D}_L$ (statt in der Kopie $\mathcal{D}'_L$) vermerkt worden wären; die Referenzen würden *vor* der eigentlichen Verschmelzung die falschen Geraden referenzieren.

Das Einfügen der Referenz τ auf eine Gerade in $\mathcal{D}_L$ impliziert das Verschieben aller Referenzen auf Geraden, die in einer komplett $<_{b\_}$-sortierten Struktur $\mathcal{D}_L$ rechts der Position j_τ der neuen Referenz τ ständen. Diese Verschiebungen müssen zusätzlich an $\mathcal{D}_I$ propagiert werden: Die Referenzen in $\mathcal{D}_I$ auf Elemente in $\mathcal{D}_L[j_\tau, \ldots, |\mathcal{D}_L| - 1]$ müssen aktualisiert (d. h. die vermerkten Adressangaben um eins inkrementiert) werden, um auf die neuen Positionen der verschobenen Einträge in $\mathcal{D}_L$ zu zeigen. Hierfür wird das in Abschnitt 5.3.4 beschriebene *insertion sort*-basierte Verfahren verwendet (vergleiche auch Fußnote 24 auf Seite 121). Das beschriebene Einfügen der Referenz τ wird nicht nur in $\mathcal{D}_L$ sondern auch in $\mathcal{D}'_L$ durchgeführt.

Um die Konsistenz von $\mathcal{D}_I$ mit $\mathcal{D}_L$ wiederherzustellen, sind zunächst die in $\mathcal{D}_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ kodierten Referenzen durch die in $\mathcal{D}'_L[j_{\text{lo}}, \ldots, j_{\text{hi}} - 1]$ zu ersetzen und anschließend entsprechend der $<_{b\_}$-Ordnung zu sortieren.
Diese Wiederherstellung der Konsistenz hat im Anschluss an das tatsächliche Verschmelzen von $\mathtt{A}_1$ mit $\mathtt{A}_2$, etwa durch den Algorithmus von Geffert *et al.* [GKP00], zu geschehen.

Lemma 5.3.19 *Die gegenüber Lemma 5.3.17notwendigen Zusatzkosten für die Konsistenzerhaltung von Referenzen in $\mathcal{D}_L$ und $\mathcal{D}_I$ belaufen sich auf $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$.*

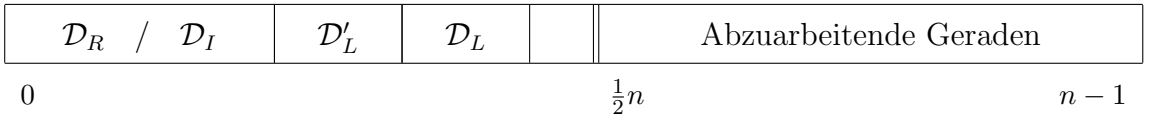
Beweis: Das Kopieren von Referenzen von $\mathcal{D}_L$ nach $\mathcal{D}'_L$ und von $\mathcal{D}'_L$ nach $\mathcal{D}_L$ kostet insgesamt $\mathcal{O}(r \cdot \log_2 n)$ Zeit für jede der $\mathcal{O}(\log_2 n)$ „Rekursionsebenen“. Des Weiteren ist jede Referenz in $\mathcal{D}_L$ oder $\mathcal{D}'_L$ auf jeder „Rekursionsebene“ Gegenstand von maximal zwei Sortiervorgängen, nämlich je einem bezüglich $<_{b\_}$ und einem bezüglich $<_{e\_}$: Jede Referenz ist Element nur eines zu verschmelzenden Eingabeteiles und jeder dieser Eingabeteile nimmt pro „Rekursionsebene“ nur an einer Verschmelzung teil. Da nicht mehr als $4 \cdot r$ Elemente in $\mathcal{D}_L$ und $\mathcal{D}'_L$ gespeichert sind, lassen sich die globalen Laufzeitkosten für dieses Sortieren (einschließlich der Kosten für Kodieren und Dekodieren von Zeigern) nach Korollar 4.1.5 durch $\mathcal{O}(r \cdot \log_2^2 r \cdot \log_2 n)$ abschätzen.

Keine der sonstigen in Abschnitt 5.3.5 beschriebenen Operationen führt zu einer Erhöhung der bereits in Lemmm 5.3.17 analysierten asymptotischen Laufzeitkomplexität für Aktualisierungen der Datenstrukturen $\mathcal{D}_R$, $\mathcal{D}_L$ und $\mathcal{D}_I$.

□

5.3.6 Speichereffiziente Auswahl und Protokollierung von r Schnittpunkten: Komplettierung

Abarbeiten der zweiten Hälfte des Eingabefeldes Nachdem die Schnittpunkte innerhalb der ersten Hälfte der Eingabe abgearbeitet worden sind, vertauschen die Eingabehälften ihre Funktionen bezüglich der Schnittpunktbearbeitung: Der modifizierte Inversionen zählende Algorithmus hat nun die zweite Hälfte der Eingabe abzuarbeiten. Gleichzeitig soll die erste Hälfte zur Protokollierung der sowohl in der ersten Phase ausgewählten als auch der noch auszuwählenden Schnittpunkte verwendet werden. Hierfür müssen zunächst die Schnittpunktinformationen von der zweiten in die erste Eingabefeldhälfte übertragen, d. h. die Datenstrukturen dorthin verschoben werden. Dies ist in linearer Zeit möglich, da für Bit-Nachbarn ihr Auslesen (in der zweiten Hälfte) und Vertauschen (in der ersten Hälfte) während eines synchronisierten linearen Durchlaufs (durch die zweite und die erste Hälfte) erfolgen kann.[27] Zur Kodierung in der ersten Eingabehälfte wird ausgenutzt, dass diese, wie ein jeder vom Inversionen zählenden Algorithmus abgearbeitete Eingabeteil, bezüglich $<_{e\_}$ sortiert vorliegt. Dementsprechend wird diese Ordnung für die Kodierung der Datenstrukturen $\mathcal{D}_R$, $\mathcal{D}_L$, $\mathcal{D}'_L$ und $\mathcal{D}_I$ verwendet. Nachfolgend ist die Struktur des Eingabefeldes vor der Abarbeitung der zweiten Phase der Schnittpunktprotokollierung skizziert.



Die nun in der zweiten Phase des Algorithmus fortzusetzende Zählung und Protokollierung von Schnittpunkten kann analog den Schilderungen für die erste Phase des Algorithmus erfolgen.

Verschmelzen der ersten mit der zweiten Eingabefeldhälfte Nach der Abarbeitung der zweiten Eingabefeldhälfte $\mathtt{A}[n/2, \ldots, n-1]$, bleiben noch die Schnittpunkte zu betrachten, die jeweils von einer Geraden aus $\mathtt{A}[0, \ldots, n/2-1]$ und einer Geraden aus $\mathtt{A}[n/2, \ldots, n-1]$ induziert werden.

Für die Korrektheit der *in-place*-Variante der Schnittpunktprotokollierung ist wesentlich, dass diese letzte Verschmelzung „virtuell“ erfolgen kann, d. h. dass Permutationen von Eingabeelementen nicht erforderlich werden: Die einem Zählen von Inversionen innerhalb eines Verschmelzungsschrittes nachfolgende Verschmelzung dient nur der Herstellung der korrekten $<_{e\_}$-Ordnung der Vereinigung der verschmolzenen Teile. Diese Ordnung ist (im vorliegenden Problemfall) nur als Vorbedingung für die effiziente Verschmelzung auf der jeweils nächsthöheren Rekursionsebene notwendig. Somit kann die Wiederherstellung der Ordnung auf der höchsten Rekursionsebene und nach Beendigung der Zählphase unterbleiben; eine „virtuelle“ Verschmelzung der Teile $\mathtt{A}[0, \ldots, n/2-1]$ und $\mathtt{A}[n/2, \ldots, n-1]$, wie im CountAndRecord-Algorithmus (Algorithmus 5.5) skizziert, ist ausreichend. Somit besteht

[27] Die Vorgehensweise beim Verschieben von kodierten Informationen wird in Abschnitt 4.1.1 erläutert.

nicht die Gefahr, dass die kodierten Datenstrukturen während der dritten Phase der Schnittpunktprotokollierung durch Umpositionierungen von Geraden korrumpiert würden.

Die finale dritte Phase erfordert jedoch die Abarbeitung von Eingabeelementen g, die zur Kodierung einer der Datenstrukturen $\mathcal{D}_R$, $\mathcal{D}_L$ oder $\mathcal{D}_I$ dienen. Dies ist unproblematisch, da das (entsprechend der $<_{e\_}$-Sortierung) jeweils nächste abzuarbeitende Eingabeelement g durch die Kodierung maximal um eine Position innerhalb des Eingabefeldes verschoben ist; es ist also nur die zusätzliche Betrachtung des Bit-Nachbarn von g erforderlich, um g zu identifizieren. Diese zusätzlichen Betrachtungen erfordern nur konstant viel zusätzlichen Speicher und verursachen konstante zusätzliche Laufzeitkosten pro abzuarbeitendem Eingabeelement, d. h. insgesamt nur lineare Zusatzkosten.

5.3.7 Komplettieren einzelner Suchschritte und Komplettieren der Suche

Bestimmen eines neuen Suchstreifens durch Betrachtung von $\mathcal{R}$

Nach Abarbeitung der dritten Phase sind alle Schnittpunkte ausgewählt und protokolliert, deren Ränge in der Datenstruktur $\mathcal{D}_R$ vermerkt waren: Die Datenstruktur $\mathcal{D}_I$ referenziert diese Schnittpunkte durch r Paare von kodierten Zeigern in die Datenstruktur $\mathcal{D}_L$, in der bis zu $2 \cdot r$ Zeiger auf Geraden, d. h. auf Eingabeelemente in `A`, verweisen.

Ein neuer (noch abschließend zu überprüfender) Suchstreifen wird nun durch die x-Koordinaten b' und e' zweier geeigneter Schnittpunkte der so protokollierten Auswahl $\mathcal{R}$ bestimmt: Hierfür sind die Schnittpunkte mit den Rängen $\kappa_{b'}$ und $\kappa_{e'}$ (bezüglich der x-Ordnung in $\mathcal{R}$) zu ermitteln (Zeile 6 in Algorithmus 5.3). Das folgende Lemma impliziert, dass die Kosten dieser Bestimmung von den Kosten der Protokollierung von $\mathcal{R}$ dominiert werden:

Lemma 5.3.20 *Sei eine Auswahl $\mathcal{R}$ von r Schnittpunkten in einer Eingabe von n Geraden wie oben beschrieben kodiert. Die Bestimmung der Schnittpunkte mit den Rängen $\kappa_{b'}$ und $\kappa_{e'}$ (bezüglich der $<_x$-Ordnung in $\mathcal{R}$) ist* in-place *in $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$ Zeit möglich.*

Beweis: Die in $\mathcal{D}_I$ kodierten Schnittpunkte können per *heapsort* sortiert werden, was $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$ Zeit gemäß Lemma 4.1.5 erfordert. Etwaige Duplikate in der Eingabe beeinflussen die Korrektheit des Resultats nicht: Unabhängig davon, ob die durchgeführte Sortierung stabil ist oder nicht, ergeben sich dieselben Intervallgrenzen b' und e' für den neuen Suchstreifen. Das abschließende Bestimmen der x-Koordinaten der gesuchten Schnittpunkte erfolgt in $\mathcal{O}(\log_2 n)$ Zeit durch das Auslesen der $\kappa_{b'}$-ten und der $\kappa_{e'}$-ten Referenzen in $\mathcal{D}_I$ und das abschließende Auslesen der von diesen Referenzen referenzierten Zeiger in $\mathcal{D}_L$. □

Der einen jeden Suchschritt abschließende Test, ob sich der zu bestimmende Schnittpunkt im so ermittelten gemutmaßten Suchstreifen $\mathcal{I}(b', e')$ befindet, ist wie in Abschnitt 5.3.2 beschrieben in $\mathcal{O}(n \cdot \log_2 n)$ Zeit durchführbar.

Abschließen der Suche

Ist die den Suchschritt abschließende Überprüfung erfolgreich, kann die Suche auf die im Suchstreifen $\langle b', e' \rangle$ befindlichen Schnittpunkte eingeschränkt werden. Da deren Anzahl $|\mathcal{I}(b', e')|$ für diese Überprüfung bereits bestimmt wurde, kann die Abbruchbedingung des *slope selection*-Algorithmus (Zeile 11 in Algorithmus 5.3) in konstanter Zeit überprüft werden. Ist diese Abbruchbedingung erfüllt, d. h. gilt $|\mathcal{I}(b', e')| \leq r$, so kann der beschriebene Algorithmus zur *zufälligen* Auswahl einer Menge von Schnittpunkten in modifizierter Form angewendet werden, um *alle* Schnittpunkte in $\langle b', e' \rangle$ zu protokollieren. Der gesuchte Schnittpunkt mit Rang k bezüglich der $<_x$-Sortierung der Schnittpunkte der Eingabe bestimmt sich nun als der Schnittpunkt mit Rang $\kappa = (r/N)(k - |\mathcal{I}(-\infty, b)|)$ innerhalb der Menge $\mathcal{R}$ der protokollierten Schnittpunkte; die Selektion dieses Schnittpunktes erfolgt analog zur Selektion der Grenzen für einen gemutmaßten Suchstreifen (siehe den Beweis zu Lemma 5.3.20).[28]

Diese Bestimmung komplettiert den *in-place*-Algorithmus zur Auswahl einer Geraden mit bezüglich ihres Ranges ausgewählter Steigung. Wählt man die Stichprobengrößen r der randomisierten Suche geeignet, etwa einheitlich für alle Suchschritte durch $r \in \Theta(\sqrt{n})$ mit $r \leq n/(32 \cdot \lceil \log_2 n \rceil)$ (vergleiche Korollar 5.3.16), so ergibt sich eine erwartete Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ pro Suchschritt.

Durch Wahl von $r \in \Theta(\sqrt{n})$ ist zudem gesichert, dass ein $0 < \beta < 1$ existiert, so dass in jedem Suchschritt $r \leq |\mathcal{S}|^\beta$ gilt. Somit ist die wesentliche Voraussetzung für Lemma 5.2.2 und Lemma 5.2.4 erfüllt. Diese Lemmata lassen sich auf den vorliegenden Fall der Bestimmung des (bezüglich $<_x$) k-ten Schnittpunktes übertragen, sofern $<_x$ eine strenge lineare Ordnung auf den Kandidaten bildet. Dies ist genau dann der Fall, wenn keine drei Punkte der originären Eingabe für das *slope selection*-Problem kollinear sind. An dieser Stelle — wie schon bei Matoušek [Mat91b] und von Dillencourt *et al.* [DMN92] — sei darauf verwiesen, dass Techniken von Edelsbrunner und Mücke [EM90] im vorliegenden Kontext zur (virtuellen) Perturbation von Eingabeelementen verwendet werden können, um Kollinearitäten in der Eingabe aufzulösen. Wie sich Kollinearitäten im konkreten Anwendungsfall ohne Verwendung der Techniken von Edelsbrunner und Mücke auflösen lassen, ist bereits in Bemerkung 5.3.10, analog zu den Ausführungen von Dillencourt *et al.* [DMN92], beschrieben worden.

Da somit nur eine konstante Anzahl von Suchschritten zu erwarten ist und diese Suchschritte jeweils eine erwartete Laufzeit von $\mathcal{O}(n \log_2 n)$ und einen Bedarf an zusätzlichem Speicher in $\mathcal{O}(1)$ aufweisen, ist auch der konstruktive Nachweis von Satz 5.3.2 erbracht.

[28] Bezüglich der Eindeutigkeit des Ergebnisses der abschließenden Elementselektion gilt: Eine fehlende Stabilität der vorangegangenen Sortierung ist für die Korrektheit des Ergebnisses des Algorithmus nicht erforderlich, da ein bezüglich der x-Koordinate k-ter Schnittpunkt erwartet wird; existieren mehrere Schnittpunkte $\sigma_1, \ldots, \sigma_m$ mit gleicher x-Koordinate, die also diese Bedingung (jeweils bezüglich einer anderen Anordnung von $\sigma_1, \ldots, \sigma_m$ bezüglich $<_x$) erfüllen, wird zwischen diesen nicht unterschieden, und ein beliebiger dieser Schnittpunkte kann zurück gegeben werden. Die originäre *slope selection*-Problemstellung erwartet zudem als Rückgabe keine Gerade (also keinen dualen Schnittpunkt), sondern nur den numerischen Wert der medianen Steigung. Dieser ergibt sich für alle σ_i als deren x-Koordinate identisch.

5.3.8 Generalisierungen, Optimierungen und Varianten

Eine randomisierte Optimierung der Schnittpunktprotokollierung

Wird statt dem klassischen *insertion sort*-Verfahren für das Einfügen in die Sortierung in $\mathcal{D}_L$ das randomisierte und erwartet schnellere *library sort* von Bender *et al.* [BFM04] verwendet (vergleiche die Fußnoten 24 und 25 auf den Seiten 121 und 122), so ist die *erwartete* Laufzeit für die Protokollierung der Auswahl $\mathcal{R}$ in $\mathcal{O}(r \cdot \log_2 r \cdot \log_2^2 n)$.

Dies impliziert, dass auch (noch kodierbare) Auswahlgrößen r aus $\mathcal{O}(n/\log_2^2 n)$ die erwartete asymptotische Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ aus Satz 5.3.2 nicht erhöhen. Es bleibt zu beachten, dass die Mächtigkeit r zusätzlich durch die Notwendigkeit der Kodierung aller Informationen der Auswahl $\mathcal{R}$ durch n Eingabeelemente beschränkt ist: Diese Beschränkung von ursprünglich $r \leq n/(32 \cdot \lceil \log_2 n \rceil)$ verschärft sich durch die Verwendung von *library sort* mit einem (frei wählbaren) Parameter $\epsilon > 0$ um einen Faktor $1 + \epsilon/2$.

Generalisierung für Eingaben mit Duplikaten

Als Folgerung aus den Ausführungen in Bemerkung 5.3.10 zu duplikathaltigen Eingaben für das *slope selection*-Problem ergibt sich indirekt, dass das Resultat aus Satz 5.3.2 auch für solche Eingaben gültig ist, sofern sie genügend unterscheidbare Elemente besitzen, um die Kodierung der Auswahlen $\mathcal{R}$ während der Abarbeitung des Inversionen zählenden Algorithmus zu sichern:

Korollar 5.3.21 *Die Aussage von Satz 5.3.2 ist für ein beliebiges $c > 1/2$ auch für die Menge P_c von Eingaben, die mindestens $c \cdot n$ unterscheidbare Elementen enthalten, gültig.*

Beweis: Der Nachweis von Lemma 5.2.1, das die konstante erwartete Anzahl an notwendigen Suchschritten impliziert, wird von Dillencourt *et al.* auch für duplikatbehaftete Eingaben geführt: Der Nachweis lässt Duplikate bezüglich der die Suchaufgabe definierenden Ordnung $<_x$ zu und legt keine weiteren Annahmen über die Unterscheidbarkeit der Eingabeelemente zu Grunde (vergleiche Lemma 2.1 und 2.2 sowie Kapitel 5.1 in [DMN92]).

Somit ist die Korrektheit und Effizienz des *slope selection*-Algorithmus in seiner originalen Fassung auch für duplikathaltige Eingaben nachgewiesen. Für die Übertragung dieser Aussage auf die *in-place*-Variante ist zuvorderst zu beachten, das der *in-place*-Algorithmus von Mannila und Ukkonen [MU84] für das Verschmelzen zweier Eingabeteile innerhalb des Inversionen zählenden Algorithmus verwendet werden kann und dieser auch für Multimengen korrekt und laufzeitoptimal arbeitet.

Es bleibt die Schwierigkeit der Schnittpunktprotokollierung durch implizite Kodierung zu betrachten. Für eine duplikatbehaftete Eingabe ist zu beachten, dass nur die unterscheidbaren Elemente der Eingabe durch Permutationen Informationen kodieren können (vergleiche Bemerkung 4.1.3). Die Anzahl der Duplikate m in der Eingabe kann in $\mathcal{O}(n \cdot \log_2 n)$ Zeit und *in-place* durch Sortieren etwa bezüglich der lexikografischen $<_{b-}$-Ordnung für ein $b \in \mathbb{R}$ und einen anschließenden linearen Durchlauf ermittelt werden. Die Separierung der Duplikate eines Eingabeteiles

$\mathtt{A}[0, \ldots, n/2-1]$, bzw. $\mathtt{A}[n/2, \ldots, n-1]$ von den unterscheidbaren Elementen in der jeweiligen Eingabefeldhälfte, die zur Kodierung verwendet werden soll, ist in linearer Zeit durch Anwendung des *in-place*-Teilmengenextraktionsalgorithmus von Bose *et al.* [BMM+06] möglich. Die im nachzuweisenden Korollar geforderte Anzahl an unterscheidbaren Elementen sichert, dass ein duplikatfreier Teil von $(c - 1/2) \cdot n$ Elementen für die Kodierung von Schnittpunktinformationen zur Verfügung steht. Diese Anzahl ist ausreichend, um eine Auswahl $\mathcal{R}$ zu kodieren, die groß genug ist, um die Voraussetzung von Lemma 5.2.1 zu erfüllen und damit eine erwartet konstante Anzahl an Suchschritten zu garantieren. Während der dritten Phase der Protokollierung, d. h. für die finale Anwendung des Verfahrens CountAndRecord (Algorithmus 5.5), ist zusätzlich der Durchlauf durch die linke Eingabefeldhälfte zu modifizieren: Dieser Durchlauf hat nun synchron durch die Menge der kodierenden Elemente und durch die (vorab zu sortierende) Menge der Duplikate zu erfolgen. Diese Modifikation hat keine Auswirkung auf die asymptotische Laufzeitkomplexität des Inversionen-Zählens von $\mathcal{O}(n \cdot \log_2 n)$ und ist mit konstantem zusätzlichem Speicher (für einen Zeiger in den Eingabeteil, der die Duplikate enthält) zu realisieren. □

Eine alternative Realisierung der Protokollierung von Schnittpunkten

Im Folgenden wird ein weiterer Algorithmus zur Schnittpunktprotokollierung vorgestellt, welcher Kopien von Eingabegeraden statt Referenzen auf diese speichert, d.h. implizit kodiert.

Motivation für die Protokollierung durch Referenzen und eine Alternative hierzu Der vorgestellte *slope selection*-Algorithmus vermeidet es, Eingabeelemente $\mathtt{A}^*[i]$ direkt, d. h. durch ihren in $\mathtt{A}$ vermerkten Wert, zu kodieren.[29] Stattdessen werden Referenzen auf Eingabeelemente kodiert. Dies ist zuforderst darin begründet, dass im Allgemeinen von einer Zugreifbarkeit der Darstellung der für Eingabeelemente verwendeten Zahl- bzw. Referenztypen nicht auszugehen ist. Insbesondere ist somit die bitweise Kodierung der Werte eines Eingabeelementes im Allgemeinen nicht möglich, da diese Werte nicht auf Bit-Ebene auszulesen sind. In diesem Abschnitt wird eine Lösung des *slope selection*-Problems vorgestellt, die jedoch von der bitweisen Kodierbarkeit von Eingabeelementen ausgeht und dadurch deutlich simpler als die soeben vorgestellte Lösung konzipiert ist. Insofern kann diese Variante als eine Anregung verstanden werden, die Vorteile einer transparenten (bitweisen) Darstellung von Eingabedaten für (die Entwicklung von) *in-place*-Algorithmen zu evaluieren.

Zusätzlich zu genannter Einschränkung bezüglich der Verfügbarkeit kodierbarer Bit-Darstellungen von Eingabeelementen ist das Kodieren von Eingabewerten statt von Referenzen in einer weiteren Hinsicht problematisch: Die Kodierung eines

[29] Mit $\mathtt{A}^*[i]$ sei im folgenden das i-te Element des *übergebenen* Eingabefeldes bezeichnet, um von der Notation $\mathtt{A}[i]$ zu unterscheiden, die das i-te Element des Eingabefeldes in seiner jeweiligen aktuellen Konfiguration bezeichnet. Es ist zudem zu beachten, dass ein Element $\mathtt{A}^*[i]$, entsprechend den Ausführungen in Kapitel 4, ein Eingabedatum oder einen Verweis in ein externes Speichermedium auf ein solches Eingabedatum darstellen kann.

Eingabeelementes $\mathtt{A}^*[i]$ erfordert (je nach Anwendungshintergrund unterschiedlich deutlich) mehr impliziten Speicherplatz, d. h. permutierbare Eingabeelemente, als die Kodierung einer Referenz i auf $\mathtt{A}^*[i]$. Sofern $\mathtt{A}$ keine Duplikate enthält, sind die Kosten (in der Anzahl kodierender Eingabeelemente) für die Kodierung von $\mathtt{A}^*[i]$ mindestens so hoch wie die Kosten für die Kodierung der Zahl i, was sich aus der Unterscheidbarkeit der n Eingabeelemente ergibt.

Die Kodierung von Werten $\mathtt{A}^*[i]$ ist im Verhältnis um so teurer, je kleiner die Eingabegröße n ist und je höher die numerische Genauigkeit der Koordinaten der Eingabepunkte ist. Eine eventuell noch gravierendere Diskrepanz zwischen den Kosten der genannten Alternativen ergibt sich, wenn das Format für Eingabeelemente neben Punktinformationen noch weitere (für die *slope selection*-Problemstellung irrelevante) Attribute enthält. Ein Verwerfen der entsprechenden Attributwerte oder ein irreversibles Trennen dieser Attributwerte von den für das Lösen der *slope selection*-Probleminstanz relevanten Daten ist nicht zulässig. Dies besagt das in Kapitel 4 beschriebene Modell und begründet sich durch die mögliche Relevanz der Daten für die Anwendung, die die Probleminstanz $\mathtt{A}$ liefert.

Die Protokollierung von Schnittpunkten während eines Suchschrittes kann jedoch signifikant vereinfacht werden, wenn Elemente $\mathtt{A}^*[i]$ der Eingabe explizit statt durch einen Index i vorgehalten werden. Nachfolgend ist eine Variante für die Protokollierung von Schnittpunktinformationen während eines Suchschrittes des vorgestellten *slope selection*-Algorithmus skizziert, welche eine Eingabeelemente $\mathtt{A}^*[i]$ kodierende Struktur $\mathcal{D}_T$ zur Datenhaltung verwendet.

Protokollierung der Schnittpunktinformationen Die Datenstruktur $\mathcal{D}_T$ wird implizit durch Permutation von Eingabeelementen vorgehalten und ist ihrer Art nach eine sortierte Liste. Ein Eintrag in $\mathcal{D}_T$ repräsentiert einen Schnittpunkt und besteht aus einem Tripel $(r_i, \mathtt{A}^*[i_1], \mathtt{A}^*[i_2])$, wobei r_i den Rang des Schnittpunktes (wie gehabt bezüglich seiner Entdeckung durch den Inversionen zählenden Algorithmus) und $\mathtt{A}^*[i_1]$ und $\mathtt{A}^*[i_2]$ die den Schnittpunkt induzierenden Geraden darstellen. Die Struktur $\mathcal{D}_T$ wird zunächst — wie die in Abschnitt 5.3.4 erläuterte Datenstruktur $\mathcal{D}_R$— mit den r Rängen der zu kodierenden Schnittpunkte gefüllt. Diese Ränge werden anschließend sortiert.

Der wesentliche Vorteil der hier geschilderten Variante ist, dass die Positionsveränderungen der zu Schnittpunkten beitragenden Geraden nicht verfolgt werden müssen; die Ergänzung um Schnittpunktinformationen $\mathtt{A}^*[i_1]$ und $\mathtt{A}^*[i_2]$ sind die einzig notwendigen Aktualisierungen der Protokollinformationen während des Ablaufs des modifizierten Inversionen zählenden Algorithmus CountAndRecord (Algorithmus 5.5). Ein Äquivalent der beschriebenen synchronisierten Sortierungen der Datenstrukturen $\mathcal{D}_L$ und $\mathcal{D}_I$ wird nicht benötigt. Auch die Entfernung von Duplikaten in $\mathcal{D}_L$, so dass jede Gerade nur einmal protokolliert wird, auch wenn sie mehrere protokollierte Schnittpunkte induziert, ist nicht notwendig.

Die für die Aktualisierungen von $\mathcal{D}_T$ notwendigen Statusinformationen sind die bei der Erläuterung der Datenstruktur $\mathcal{D}_R$ genannten: Vorzuhalten (in konstantem zusätzlichem Speicher) ist die Anzahl j der bereits gefundenen Schnittpunkte; diese Information gibt den Index j des Eintrages in $\mathcal{D}_T$ vor, der als nächstes durch Schnittpunktinformationen $(\mathtt{A}^*[j_1], \mathtt{A}^*[j_2])$ zu ergänzen ist. Wie für $\mathcal{D}_R$ wird auch für

$\mathcal{D}_T$ der Rang $\mathcal{D}_T[j]$ dieses nächsten zu protokollierenden Schnittpunktes dekodiert in konstantem zusätzlichem Speicher vorgehalten.

Für die Analyse der implizit vorzuhaltenden Struktur $\mathcal{D}_T$ bezeichne $\int$ im Folgenden die Anzahl der Bits, die für die Repräsentation eines Elementes des Eingabefeldes erforderlich ist. Für die Kodierung der ausgewählten Ränge der Schnittpunkte werden $4 \cdot r \cdot \lceil \log_2 n \rceil$ Eingabeelemente benötigt. Für die explizite Kodierung der insgesamt $2 \cdot r$ Eingabeelemente (mit Vielfachheiten gezählt), die als Geraden interpretiert die Schnittpunkte aus $\mathcal{R}$ induzieren, sind $4 \cdot r \cdot \int$ Eingabeelemente notwendig. Hieraus ergibt sich eine maximal darstellbare Größe r der Auswahl $\mathcal{R}$ von $n/(8 \cdot (\int + \lceil \log_2 n \rceil))$.

Der Zeitaufwand für das initiale Sortieren der Ränge in $\mathcal{D}_T$ ist in $\mathcal{O}(r \cdot \log_2 r \cdot \log_2 n)$. Das Dekodieren des jeweils nächsten Ranges erfordert global nur $\mathcal{O}(r \cdot \log_2 n)$ Zeit. Der Zeitaufwand für das Kodieren der Eingabeelemente $\mathtt{A}^*[i]$, die (als Geraden interpretiert) die zu protokollierenden Schnittpunkte induzieren, beläuft sich auf $\mathcal{O}(r \cdot \int)$ Zeit. Der globale Zeitaufwand für das Protokollieren von Schnittpunktinformationen ist damit in $\mathcal{O}(r \cdot (\int + \log_2 r \cdot \log_2 n))$.

Komplettieren eines Suchschrittes Wie der bereits beschriebene Algorithmus protokolliert auch diese Variante die Schnittpunkte in derselben dreiphasigen Unterteilung. Nach Abschluss der dritten Phase sind alle Schnittpunkte, deren Ränge in $\mathcal{D}_R$ vermerkt waren, in $\mathcal{D}_T$ kodiert. Die Schnittpunkte in $\mathcal{D}_T$ können nun per *heapsort* bezüglich ihrer x-Koordinate *in-place* und in $\mathcal{O}(r \cdot \log_2 r \cdot (\int + \log_2 n))$ Zeit sortiert werden.

Ein verkleinerter (noch abschließend zu überprüfender) Suchstreifen kann nun analog der Beschreibung in Kapitel 5.3.7 durch direkten Zugriff auf zwei durch ihre Ränge in der Sortierung von $\mathcal{R}$ bestimmten Elemente gemutmaßt werden. Der Zeitaufwand für diese Nachbearbeitungen der in $\mathcal{D}_T$ gesammelten Schnittpunktinformationen wird daher von den Sortierkosten dominiert und beläuft sich auf $\mathcal{O}(r \cdot \log_2 r \cdot (\int + \log_2 n))$ Zeit.

Um eine Gesamtlaufzeit von $\mathcal{O}(n \cdot \log_2 n)$ für diese Variante des *slope selection*-Algorithmus zu erhalten, muss für die Mächtigkeit r der Auswahl $\mathcal{R}$ an Schnittpunkten $(r \log_2 r) \in \mathcal{O}((n \cdot \log_2 n)/(log_2 n + \int))$ gesichert sein. Für $\int \in \mathcal{O}(\log_2 n)$ erlaubt diese Bedingung eine Mächtigkeit r der Auswahl $\mathcal{R}$ von $\mathcal{O}(n/\log_2 n)$, ohne die asymptotische Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ zu verletzen.

5.4 Randomisierte Berechnung des *repeated median*-Schätzers

Der *repeated median*-Schätzer (vergleiche die Auflistung von Geradenschätzern auf Seite 97 in Abschnitt 5.1) lässt sich durch Betrachten der dualen Problemstellung berechnen (wie auch der Theil-Sen-Schätzer, siehe Seite 105 in Abschnitt 5.3): Im dualen Raum ist zu jeder der (dualen) Geraden g_i der Eingabe ihr (bzgl. $<_x$) medianer Schnittpunkt m_i zu betrachten. Das Dual des *repeated median* ist dann der Median über alle Mediane $(m_i)_{i=1,\ldots,n}$.

Der *repeated median*-Schätzer ist mit einem *breakdown value* von 50 % von optimaler Robustheit und insbesondere robuster als der Theil-Sen-Schätzer. Nach Rousseeuws These, dass „es tieferliegende Gründe dafür zu geben scheint, dass besonders robuste Schätzer für die Regressionsanalyse sich nicht in günstiger Weise berechnen lassen“ [RL87, Seite 29], besteht ein Zusammenhang zwischen der Robustheit eines Geradenschätzers und seinem Berechnungsaufwand. Insofern ist es nicht zwangsläufig, dass auch für den *repeated median*-Schätzer ein Algorithmus mit erwarteter Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ konstruierbar ist. Matoušek *et al.* [MMN98] stellten drei Varianten eines randomisiert suchenden Algorithmus vor, von denen eine die genannte erwartete Laufzeit bietet.

Diese effizienteste Variante nutzt Bereichssuchen und -zähltechniken, die in der verwendeten Form nicht *in-place* realisierbar scheinen. Diese Techniken fußen zudem auf theoretischen Resultaten über Halbraum-Bereichsabfragen und die entsprechenden asymptotisch optimal arbeitenden Datenstrukturen gelten als wenig praktikabel und „machen diese Variante für eine Implementierung unattraktiv“ [MMN98].[30] Die randomisierte Suche durch Interpolation arbeitet, wie ausgeführt, für viele Problemstellungen effizienter als die binäre Suche (vergleiche [Mat91b, MMN98] sowie Abschnitt 5.3 dieser Arbeit). Im vorliegenden Problemfall bietet die Suche durch Interpolation nicht die erwartet konstante Anzahl an zu erwartenden Suchschritten wie für die in den Abschnitten 5.2 und 5.3.1 vorgestellten Aufgabenstellungen. Fundiert durch praktische Laufzeitevaluationen führen Matoušek *et al.* jedoch an, dass für viele nicht entartete Eingabekonfigurationen die randomisierte Suche durch Interpolation zu nur erwartet konstant vielen Suchschritten führt. In diesen Fällen ergibt sich zudem dieselbe erwartete asymptotische Laufzeit wie für die Variante des Algorithmus, die sich auf komplizierte Datenstrukturen zur Bereichsabfrage stützt.

In diesem Abschnitt werden die beiden suboptimalen Varianten mit einer erwarteten Laufzeit von $\mathcal{O}(n \cdot \log_2^2 n)$ betrachtet und zu diesen eine jeweilige *in-place*-Realisierung vorgestellt. Diese Realisierungen haben gegenüber den Algorithmen zur Berechnung des Theil-Sen-Schätzers einen erhöhten Umfang an statistischen Informationen zu verwalten. Diese Verwaltung ist zwar *in-place* und durch implizite Kodierung von Informationen zu leisten, allerdings ergibt sich dadurch ein zusätzlicher logarithmischer Faktor in der asymptotischen erwarteten Laufzeitkomplexität:

Satz 5.4.1 *Die Berechnung des* repeated median*-Schätzers für eine Menge von n Punkten der Ebene ist* in-place *und in erwarteter Zeit von $\mathcal{O}(n \cdot \log_2^3 n)$ möglich.*

Gliederung Beide im Folgenden betrachteten Varianten zur Berechnung des *repeated median*-Schätzers verfeinern iterativ — wie der beschriebene *slope selection*-Algorithmus von Matoušek [Mat91b] — einen Suchstreifen $\langle b, e \rangle$, in dem sich der zu suchende Schnittpunkt (hier der *repeated median*) befindet; die zum aktuellen Suchstreifen korrespondierende Kandidatenmenge besteht aus genau den Geraden g_i der Eingabe, deren medianer Schnittpunkt m_i in den Suchstreifen fällt. Die einfachere dieser beiden Varianten verwendet „nur“ die Technik der randomisierten binären

[30] Die Aussage von Matoušek *et al.* im originalen Wortlaut: „Although the algorithm [...] is asymptotically efficient, its reliance on data structures for half-space range queries makes it less attractive for practical implementation.“

Suche (vergleiche Abschnitt 5.2.2), die zweite Variante verwendet die randomisierte Suche durch Interpolation und bietet Laufzeitvorteile für in praktischen Szenarien zu erwartende Eingaben [MMN98]. Aus Gründen der Übersichtlichkeit der Darstellung wird zunächst die randomisiert binär suchende Variante und eine entsprechende *in-place*-Realisierung vorgestellt. Anschließend wird eine *in-place*-Realisierung der interpolierend suchenden Variante dargestellt und die beiden *in-place*-Varianten werden hinsichtlich ihrer Praktikabilität verglichen.

5.4.1 Bestimmung des *repeated median*-Schätzers durch randomisierte binäre Suche

Der ursprüngliche randomisiert binär suchende Algorithmus

Bei der randomisiert suchenden Variante von Matoušek *et al.* [MMN98] wird die Verfeinerung des Suchstreifens $\langle b, e\rangle$, der den *repeated median* enthält, durch Wahl eines einzelnen Schnittpunktes s aus $\langle b, e\rangle$, dessen x-Koordinate mit x_s bezeichnet sei, erreicht. Es ist anschließend zu bestimmen, ob sich der *repeated median* in $\langle x_s, e\rangle$ oder $\langle b, x_s\rangle$ befindet, um unter diesen beiden Teilstreifen die neue Suchdomäne zu identifizieren. Bezeichne $\mathcal{M}(b, e)$ die Menge der Eingabegeraden, deren medianer Schnittpunkt im Streifen $\langle b, e\rangle$ liegt. Der rechte Teilstreifen $\langle x_s, e\rangle$ ist genau dann der neue Suchstreifen, wenn die Anzahl $|\mathcal{M}(x_s, e)|$ der in ihm befindlichen Mediane größer als die Anzahl $|\mathcal{M}(b, x_s)|$ der im linken Teilstreifen befindlichen Mediane ist.

Die Verfeinerung des Suchstreifens wird so lange iteriert, bis dieser den *repeated median* als einzigen Schnittpunkt enthält. Nachfolgend ist dieser iterierte Ablauf als Algorithmus RepeatedMedianBinary in Algorithmus 5.6 skizziert.

Algorithmus 5.6 Algorithmus RepeatedMedianBinary($\mathtt{A}[0, \ldots, n-1]$) bestimmt durch randomisierte binäre Suche den *repeated median* unter den Schnittpunkten der Geraden in $\mathtt{A}$ [MMN98].

1: $b := -\infty$; $e := \infty$. /* Erste „Schätzung“ für einen den *repeated median* enthaltenden Suchstreifen. */
2: **repeat**
3: Wähle zufällig einen Schnittpunkt s aus, der im Streifen $\langle b, e\rangle$ liegt.
4: **if** $|\mathcal{M}(x_s, e)| > |\mathcal{M}(b, x_s)|$ **then**
5: $b := x_s$. /* Der *repeated median* befindet sich in $\langle x_s, e\rangle$. */
6: **else**
7: $e := x_s$. /* Der *repeated median* befindet sich in $\langle b, x_s\rangle$. */
8: **end if**
9: **until** $|\mathcal{M}(b, e)| = 1$ /* Der *repeated median* ist bestimmt. */
10: Gebe den verbliebenen medianen Schnittpunkt (bzw. sein Dual) als den gesuchten *repeated median* zurück.

Die einzelnen Teilschritte des in Algorithmus 5.6 skizzierten Verfahrens lassen sich wie folgt realsieren.

Auswahl Für die zufällige Wahl eines Schnittpunktes (in Zeile 3 in Algorithmus 5.6) wählt man zunächst zufällig einen Rang aus den Zahlen $1, \ldots, |\mathcal{M}(b, e)|$.[31] Nun kann der Inversionen zählende Algorithmus (wie für das *slope selection*-Problem in Abschnitt 5.3.3 beschrieben) adaptiert werden, um den Schnittpunkt mit dem ausgewählten Rang zu bestimmen.

Überprüfung Die Überprüfung, in welchem der beiden Teilstreifen sich der *repeated median* befindet, erfordert das Zuordnen der Mediane m_i der einzelnen Geraden zu $\mathcal{M}(b, x_s)$ bzw. zu $\mathcal{M}(x_s, e)$ (Zeile 4 in Algorithmus 5.6). Dabei gilt mit L_i (bzw. R_i) als der Anzahl der Schnittpunkte in $\langle b, x_s \rangle$ einer Eingabegeraden g_i:

$g_i \in |\mathcal{M}(b, x_s)| \Leftrightarrow L_i > R_i$

Die Bestimmung der Werte $|\mathcal{M}(b, x_s)|$ bzw. $|\mathcal{M}(x_s, e)|$ in $\langle b, x_s \rangle$, bzw. in $\langle x_s, e \rangle$ — separat für jede einzelne Gerade der Eingabe — zu leisten. Das Zählen von Schnittpunkten in Streifen kann nach Lemma 5.3.6 auf das Zählen von Inversionen im entsprechenden Streifen zurückgeführt werden.

Die für die Effizienz dieser Strategie zur Bestimmung des *repeated median* entscheidende, zusätzliche Beobachtung ist, dass dieses Zählen für *alle* Geraden der Eingabe gleichzeitig während einer einzigen Ausführung des — geeignet zu modifizierenden — Inversionen zählenden Algorithmus geschehen kann.[32] Nachfolgend ist diese Modifikation von Algorithmus 5.4 als Algorithmus 5.7 dargestellt.

Algorithmus 5.7 Algorithmus CountPerLine $(\mathtt{A}, b_1, b_2, c_1, c_2, \langle b, e \rangle, \mathtt{L})$ erhöht für jede Eingabegerade g_i aus $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ den Zähler $\mathtt{L}[i]$ (für die bisher gefundenen Schnittpunkte von g_i in $\langle b, e \rangle$) um die Anzahl der von g_i mit Geraden aus $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ gebildeten Schnittpunkte.

Vorbedingung: $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ sind bzgl. $<_{e\_}$ sortiert; $l_1 \in \mathtt{A}[b_1, \ldots, b_1 + c_1] \wedge l_2 \in \mathtt{A}[b_2, \ldots, b_2 + c_2] : l_1 <_{b\_} l_2$

Nachbedingung: $\mathtt{A}[b_1, \ldots, b_1 + c_1]$ und $\mathtt{A}[b_2, \ldots, b_2 + c_2]$ sind bzgl. $<_{e\_}$ sortiert.

```
1: i_1 := 0; i_2 := 0.
2: for i = 0 to c_1 + c_2 + 1    /* Das i-te Element in sortierter Reihenfolge ist A[i_1]
   oder A[i_2]. */  do
3:    if A[b_1 + i_1] <_e_ A[b_2 + i_2]    /* ♯(Durch A[b_1 + i_1] bedingte Inversionen) =
      ♯(Elemente in A[b_2, ..., b_2 + c_2] vor A[b_1 + i_1]) bzgl. <_e_). */ then
4:       L[i_1] := L[i_1] + i_2 + 1.                    /* Zähle Schnittpunkte. */
5:       i_1 := i_1 + 1.                                /* Inkrementiere i_1. */
6:    else
7:       i_2 := i_2 + 1.                                /* Inkrementiere i_2. */
8:    end if
9: end for
```

---

[31] Der Wert $|\mathcal{M}(b, e)|$ kann durch den Inversionen zählenden Algorithmus berechnet oder effizienter als Nebenprodukt der Überprüfung des letzten Suchschrittes (Zeile 4 in Algorithmus 5.6) ausgelesen werden.

[32] Wie noch erläutert werden wird, ist das simultane Lokalisieren der Mediane m_i *aller* n Geraden g_i nur dann während einer einzigen Abarbeitung des Inversionen zählenden Algorithmus möglich, wenn für die Speicherung der Zähler $\mathcal{O}(n)$ Wörter zusätzlich zur Eingabe zur Verfügung stehen.

Die in Zeile 4 in Algorithmus CountPerLine auszuwertende Funktion L, die explizit durch ein Feld `L` realisiert dargestellt ist, dient der Identifizierung des Zählers, der zu der (vor der Verschmelzung von $\mathtt{A}_1 = \mathtt{A}[b_1, \ldots, b_1 + c_1]$ und $\mathtt{A}_2 = \mathtt{A}[b_2, \ldots, b_2 + c_2]$) in $\mathtt{A}[i_1]$ befindlichen Geraden gehört. Die Funktion L stellt einen Automorphismus auf der Menge $\{0, \ldots, n-1\}$ der Indizes von `A` dar und kann daher durch ein Feld der Größe n realisiert werden, dass die Funktionswerte von L enthält und welches im folgenden Abschnitt zur Laufzeitanalyse genauer erläutert wird.

Es ist zu beachten, dass für jeden Aufruf des Algorithmus CountPerLine ein weiterer Aufruf mit vertauschten Parametern $\mathtt{A}_2$ und $\mathtt{A}_1$ (d. h. mit einem Parametertupel $(\mathtt{A}, b_2, b_1, c_2, c_1, \langle b, e\rangle, \mathtt{L})$) notwendig wird, da in Algorthmus CountPerLine nur die Schnittpunkte für die im ersten Eingabeteil $\mathtt{A}_1$ befindlichen Geraden gezählt werden: Um die Schnittpunkte für die Geraden in $\mathtt{A}_2$ zu zählen, ist in CountPerLine die Abarbeitungsrichtung der Schleifenanweisung in Zeile 2 und die Parameterreihenfolge in der Ordnungsüberprüfung in Zeile 3 umzukehren.[33]

Des Weiteren ist für die Überprüfung in Zeile 4 von Algorithmus 5.6 die geschilderte Adaption des Inversionen zählenden Algorithmus separat für das Zählen im Streifen $\langle b, x_s\rangle$ bzw. für das Zählen im Streifen $\langle x_s, e\rangle$ aufzurufen. In beiden Fällen ist für jede der n Geraden g_i ein Schnittpunktzähler L_i bzw. R_i aufrecht zu halten. Für die Kürze der Darstellung wird im Folgenden ausschließlich die Aufrechthaltung der Zähler $L_1, \ldots, L_n$ betrachtet, da sich die Aufrechthaltung der Zähler $R_1, \ldots, R_n$ analog ergibt.

Laufzeitanalyse Die Laufzeitanalyse wird zunächst für den von Matoušek *et al.* [MMN98] publizierten binären Algorithmus analysiert, bevor die *in-place*-Realisierung vorgestellt und analysiert wird.

Die Laufzeit eines Suchschrittes wird von den konstant vielen Aufrufen zweier Varianten des Inversionen zählenden Algorithmus dominiert: Dies ist zum einen die Variante für die zufällige Wahl eines Schnittpunktes (Zeile 3 in Algorithmus 5.6), zum anderen die Variante für das Zählen der Schnittpunkte einzelner Geraden in Intervallen (Zeile 4 in Algorithmus 5.6). Beide Varianten besitzen die gleiche asymptotische Laufzeitkomplexität wie der originale Algorithmus: Die erste ist eine Vereinfachung des Algorithmus CountAndRecord (Algorithmus 5.5) und die zweite führt dieselbe Anzahl an Inkrementierungen wie der Algorithmus CountInversions (Algorithmus 5.4) durch; die Modifikation besteht nur darin, dass die einzelnen Inkrementierungen verschiedene Zähler L_i betreffen. Die Identifikation des zu inkrementierenden Zählers L_j der aktuell durch den Inversionen zählenden Algorithmus abzuarbeitenden Geraden g_j ist in konstanter Zeit möglich: Die Funktion L ist durch ein Feld `L` der Größe n realisiert, in dem (für $i = 1, \ldots, n$) in $\mathtt{L}[i]$ der Index j, an dem sich die aktuell in $\mathtt{A}[i]$ vermerkte Gerade vor dem Beginn des Verschmelzungsprozesses in `A` befand, gespeichert wird. Die einzig notwendigen Aktualisierungen von `L` sind jeweils in Zeile 4 in Algorithmus 5.7 vorzunehmen, indem die an $\mathtt{L}[i]$ und $\mathtt{L}[i_1 + c_{i_1}]$ gespeicherten Indizes vertauscht werden.[34] Die Laufzeitkosten für das Aus-

[33] Die beiden aufzurufenden Varianten des Algorithmus CountPerLine können alternativ gemeinsam als ein einzelner Durchlauf durch die zu bearbeitenden Eingabeteile $\mathtt{A}_1$ und $\mathtt{A}_2$ realisiert werden.

[34] Es ist zu beachten, dass diese Aktualisierung nur im zweiten, modifizierten Aufruf (mit ver-

lesen und Aktualisieren eines Indexes in `L` sind jeweils konstant. Die globalen Kosten entsprechen somit der Anzahl der Positionsveränderungen in `A` während der Abarbeitung durch den Inversionen zählenden Algorithmus. Die asymptotische Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ wird durch die Verwendung der wie oben beschrieben realisierten Funktion L nicht erhöht und die jeweils zu inkrementierenden Zähler L_i sind in konstanter Zeit durch Auswertung von L bestimmbar, so dass auch für diese Variante des Inversionen zählenden Algorithmus die Laufzeitkosten konstant pro Betrachtung einer Geraden der Eingabe und damit global in $\mathcal{O}(n \log_2 n)$ sind.

Für eine jede der n Eingabegeraden g_i lässt sich nach der Zählphase durch Vergleich der Zähler L_i und R_i in konstanter Zeit bestimmen, in welchem der beiden Teilstreifen ihr medianer Schnittpunkt m_i liegt. Dieser lineare Zeitaufwand für das Extrahieren der Anzahlen $|\mathcal{M}(b, x_0)|$ bzw. $|\mathcal{M}(x_0, e)|$ der Mediane m_i im linken, bzw. im rechten Teilstreifen aus den Zählwerten L_i bzw. R_i wird somit von der Gewinnung dieser Zählwerte durch den adaptierten Inversionen zählenden Algorithmus dominiert.

Die binäre randomisierte Suche erfordert im erwarteten Fall $\mathcal{O}(\log_2 n)$ der soeben analysierten Suchschritte, um die Menge der initial n Kandidaten für den *repeated median* auf diesen selbst zu reduzieren [MMN98]. Somit ergibt sich eine erwartete Gesamtlaufzeit von $\mathcal{O}(n \log_2^2 n)$, sofern linear viele Wörter an zusätzlichem Speicher zur Verfügung stehen.

Eine speichereffiziente Realisierung

Die wesentliche Schwierigkeit, eine *in-place*-Version des soeben dargestellten binär suchenden Algorithmus zu realisieren, liegt in der Speicherung und Aufrechterhaltung der n Schnittpunktzähler L_i (sowie der Realisierung der Funktion L) während des Inversionen-Zählens zum Zwecke der Überprüfung, ob der linke oder rechte Teilstreifen den *repeated median* enthält (Zeile 4 von Algorithmus 5.6).

Unterteilung der Überprüfung in Blockbearbeitungen In einer Eingabemenge der Größe n sind nach Beobachtung 4.1.2 nur $\lfloor n/2 \rfloor$ Bits durch Permutation von Eingabeelementen kodierbar. Folglich lassen sich in einer Eingabehälfte nur $\lfloor n/4 \rfloor / \lceil \log_2 n \rceil$ Zähler L_i speichern, da diese Werte aus $1, \ldots, n$ annehmen können. Glücklicherweise ist das separate Betrachten der Zähler L_i von je $\lfloor n/4 \rfloor / \lceil \log_2 n \rceil$ Geraden in insgesamt $n/(\lfloor n/4 \rfloor / \lceil \log_2 n \rceil) \approx 4\lceil \log_2 n \rceil$ Abarbeitungen des adaptierten Inversionen zählenden Algorithmus möglich.

Es sei für die Einfachheit der Darstellung die Eingabegröße n zunächst als eine Zweierpotenz 2^m für ein $m \in \mathbb{N}$ vorausgesetzt. Somit kann die Eingabe (entsprechend der Rekursion des Inversionen zählenden Algorithmus) durch balancierte Aufteilung von der Rekursionstiefe $(\log_2 \log_2 n)+4$ in $16 \cdot \log_2 n$ *Blöcke* $\mathcal{B}$ der Größe $n/(16 \cdot \log_2 n)$ zergliedert werden.

Abarbeiten eines Blockes Für jeden der $16 \cdot \log_2 n$ Blöcke $\mathcal{B}$ wird die folgende Variante des Inversionen zählenden Algorithmus aufgerufen: Diese zählt Inversionen

tauschten Parametern $\mathtt{A}_1$ und $\mathtt{A}_2$) von CountPerLine durchzuführen ist, d. h. nur einmal pro Verschmelzungsschritt.

wie Algorithmus 5.7 während des Verschmelzens von Eingabeteilen, beschränkt sich dabei allerdings auf das Inkrementieren der nur $n/(16 \cdot \log_2 n)$ Zähler für die Schnittpunkte der Geraden in $\mathcal{B}$. Es ist gesichert, dass die Geraden eines jeden Blockes zu einem vollständigen Teilbaum in dem Rekursionsbaum des Inversionen zählenden Algorithmus assoziiert sind, da sowohl die Eingabegröße n als auch die Größe eines jeden Blockes eine Zweierpotenz darstellen: Jeder Block entspricht genau einem Eingabeteil $\mathtt{A}_j$ eines Aufrufes von Algorithmus 5.7 auf der $((\log_2 \log_2 n) + 4$-tiefsten) Rekursionsebene des Inversionen-Zählens, angewendet auf $\mathtt{A}$.

Die Protokollierung der Schnittpunkte aller Geraden eines solchen Blockes $\mathcal{B}$ erfolgt nun in zwei Phasen.

- Zunächst werden alle Schnittpunkte zwischen zwei Geraden aus $\mathcal{B}$ betrachtet. Hierfür kann der bereits beschriebene Inversionen zählende Algorithmus (vergleiche auch Algorithmus 5.4 für einen entsprechenden Zählschritt) mit $\mathcal{B}$ als Eingabe aufgerufen werden.

- Zum Zählen aller Schnittpunkte zwischen Geraden aus $\mathcal{B}$ und Geraden aus $\mathcal{P} \backslash \mathcal{B}$ wird eine reduzierte Version desselben Algorithmus verwendet, die allerdings mit $\mathcal{P}$ als Eingabe aufgerufen wird und nur genau die genannte Art von Schnittpunkten betrachtet.

Diese phasenunterteilte Abarbeitung lässt sich am besten durch die Betrachtung des Rekursionsbaumes des Inversionen zählenden Algorithmus, angewendet auf die Eingabe $\mathcal{P}$, motivieren: In der ersten Phase wird der zu $\mathcal{B}$ korrespondierende Teilbaum $\mathcal{T}_{\mathcal{B}}$ abgearbeitet. Hiernach sind die Geraden in $\mathcal{B}$ bezüglich $<_{e\_}$ sortiert. Um nun alle weiteren Inversionen der Geraden aus $\mathcal{B}$ zu finden, sind nur noch genau die Verschmelzungsschritte des Inversionen zählenden Algorithmus notwendig, die den Knoten des Rekursionsbaumes entlang des Pfades von der Wurzel von $\mathcal{T}_{\mathcal{B}}$ zur Wurzel des Rekursionsbaumes entsprechen.

In jedem dieser Verschmelzungsschritte bildet $\mathcal{B}$ einen der beiden übergebenen Eingabeteile. Die tatsächliche Verschmelzung beider Eingabeteile (in die $<_{e\_}$-Ordnung) ist in keinem dieser Schritte notwendig: Die Invariante für die Rekursion des Inversionen zählenden Algorithmus erfordert zwar die $<_{e\_}$-Ordnung beider zu verschmelzenden Teile der Eingabe. Diese Invariante bleibt aber für jeden nachfolgenden Verschmelzungsschritt zumindest für den aus $\mathcal{B}$ bestehenden Teil erfüllt, da dieser (unverändert) in jedem weiteren Verschmelzungsschritt als Argument übergeben wird.

Dieselbe Invariante ist in jedem Verschmelzungsschritt für den anderen Teil der Eingabe zu gewährleisten, was aufwändiger ist: Die globalen Kosten für das Etablieren dieser benötigten $<_{e\_}$-Ordnungen auf (mit $\mathcal{B}$ zu verschmelzenden) Teilen der Eingabe lassen sich grob durch die Kosten für das Sortieren all dieser Eingabeteile abschätzen. Auf Grund der Eigenschaften der geometrischen Reihe werden diese Kosten von der Sortierung des letzten, d. h. an der Wurzel, mit $\mathcal{B}$ zu verschmelzenden Teiles der Eingabe, dominiert, d. h. durch eine Laufzeit in $\mathcal{O}(n \log_2 n)$.

Die Funktion L kann als Automorphismus allein auf $\{1, \ldots, |\mathcal{B}|\}$ gewählt werden, da die Geraden aus $\mathcal{B}$ ihre Positionen nur untereinander tauschen, aber nicht mit Elementen anderer Blöcke verschmolzen werden. Ein die Funktion L realisierendes Feld

L kann somit *in-place* kodiert durch Permutation von nur $n/(16 \cdot \log_2 n) \cdot \log_2(n/(16 \cdot \log_2 n))$ Eingabeelementen vorgehalten werden. Die zu einem Block $\mathcal{B}$ von Geraden gehörenden Zähler können durch insgesamt $n/(8 \cdot \log_2 n) \cdot \log_2 n = n/8$ Eingabeelemente kodiert werden, da ein jeder Zähler ausschließlich Werte aus $\{0, \ldots, n-1\}$ annimmt.

Die Laufzeit für die Aktualisierung der Zähler sowie des Feldes L während einer kompletten Bearbeitung eines Blockes $\mathcal{B}$ ergibt sich wie folgt: Auf jeder der $\mathcal{O}(\log_2 n)$ Rekursionsebenen einer Blockbearbeitung wird jede Gerade g_i aus $\mathcal{B}$ einmal betrachtet. Im schlimmsten Fall werden bei jeder Betrachtung eine oder mehrere Inversionen von g_i entdeckt, so dass der zugehörige Zähler genau einmal aktualisiert werden muss. Dies kostet jeweils jeweils $\mathcal{O}(\log_2 n)$ Zeit, da das Kodieren und Dekodieren des Zählers erforderlich wird.[35] Somit ergibt sich der globale Zeitaufwand für das Aktualisieren der $\mathcal{O}(n/\log_2 n)$ Zähler zu $\mathcal{O}(n \log_2 n)$. Zusätzliche Kosten entstehen durch Umpositionierungen von Geraden, die bei Auffinden von Inversionen während einer Verschmelzung (von zwei Teilen von $\mathcal{B}$) notwendig werden. Diese Positionsänderungen müssen im implizit kodierten Feld L vermerkt werden. Dieses kann im schlimmsten Fall für $\mathcal{O}(n/\log_2 n)$ Geraden des Blockes auf allen $\log_2 n - \log_2(16 \cdot \log_2 n) = \log_2 n - \log_2(\log_2 n) - 4$ Rekursionsebenen, in denen die Geraden des Blockes verschmolzen werden, auftreten. Die Kosten für eine jede solche Aktualisierung von L sind in $\mathcal{O}(\log_2(n/\log_2 n) = \mathcal{O}(\log_2 n)$, die sich also zu globalen Kosten von erneut $\mathcal{O}(n \cdot \log_2 n)$ summieren.

Lemma 5.4.2 *Für eine jede der $\mathcal{O}(n/\log_2 n)$ Geraden eines Blockes können die jeweiligen Anzahlen ihrer Schnittpunkte in $\langle b, x_s \rangle$ bzw. in $\langle x_s, e \rangle$* in-place *und in insgesamt $\mathcal{O}(n \cdot \log_2 n)$ Zeit gezählt werden.*

Arrangieren der Blockbearbeitungen Die Bearbeitung der gesamten Eingabe kann blockweise und *in-place* durchgeführt werden, da die durch Bearbeitung eines Blockes $\mathcal{B}$ bestimmten Anzahlen die Lokalisierung genau der Mediane der Geraden des Blockes $\mathcal{B}$ (jeweils bezüglich der Menge $\mathcal{M}(b, x_s)$ bzw. bezüglich der der Menge $\mathcal{M}(x_s, e)$) ermöglicht.[36] Die Speicherung von Schnittpunktenanzahlen ist für Geraden g_i aus bereits abgearbeiteten Blöcken nicht vonnöten. Noch nicht einmal die hierdurch ermittelte Zugehörigkeit von g_i zu $\mathcal{M}(b, x_s)$ oder zu $\mathcal{M}(x_s, e)$ wird explizit vermerkt, sondern führt nur zu einer Inkrementierung von einem der Zählerwerte $|\mathcal{M}(b, x_s)|$ oder $|\mathcal{M}(x_s, e)|$, welche sich beide mit konstantem zusätzlichem Speicher vorhalten lassen. Nach insgesamt $\mathcal{O}(\log_2 n)$ Blockbearbeitungen sind die finalen Werte von $|\mathcal{M}(b, x_s)|$ und von $|\mathcal{M}(x_s, e)|$ bestimmt und somit der *repeated median* in $\langle b, x_s \rangle$ oder in $\langle x_s, e \rangle$ lokalisiert.

Da nach Lemma 5.4.2 eine einzelne Blockbearbeitung *in-place* und in $\mathcal{O}(n \cdot \log_2 n)$ Zeit möglich ist und in jedem Suchschritt $\mathcal{O}(\log_2 n)$ solcher Bearbeitungen durchgeführt werden, folgt mit der erwarteten Anzahl an Suchschritten von $\mathcal{O}(\log_2 n)$

[35] Eine amortisierte Analyse über *alle* Inkrementierungen eines Zählers liefert leider keine geringere Laufzeitkomplexität, da für eine Inkrementierung eines Zählers im Mittel die Manipulation von in n logarithmischen vielen Bits notwendig wird.

[36] Hierfür ist zu beachten, dass während einer Blockbearbeitung nicht nur Eingabeelemente für die Kodierung der Zähler L_i, sondern auch für die Kodierung der Zähler R_i, verwrndet werden müssen, damit oben genannte Lokalisierung möglich ist.

die Aussage von Satz 5.4.1. Die Verallgemeinerung auf Eingabemengen, die keine Zweierpotenz darstellen, ergibt sich dabei analog zu Bemerkung 5.3.11: Die Größe der Blöcke wird nach wie vor als Zweierpotenz gewählt, mit Ausnahme des letzten Blockes; auch dieser letzte Block entspricht einem Teilbaum des (nicht vollständigen) Rekursionsbaumes der Abarbeitung der kompletten Eingabe.

5.4.2 Bestimmung des *repeated median*-Schätzers durch randomisierte Suche durch Interpolation

Motivation und Einordnung der randomisierten Suche durch Interpolation Die randomisierte Suche durch Interpolation ist hinsichtlich der Sucheffizienz eine Verbesserung gegenüber der binären Suche (siehe [Mat91b, MMN98]). Im vorliegenden Problemfall jedoch benötigen beide Suchtechniken $\mathcal{O}(\log_2 n)$ Suchschritte im erwarteten Fall. Fundiert durch praktische Laufzeitevaluationen führen Matoušek *et al.* jedoch an, dass für viele Eingabekonfigurationen eine konstante Anzahl an Suchschritten zu erwarten ist. Da allerdings „pathologische Situationen vorstellbar sind, die die Konvergenzrate des Algorithmus beeinflussen“ [MMN98],[37] wird bei Erkennen einer schlechten Konvergenzrate die Suchstrategie gewechselt und der in Abschnitt 5.4.1 randomisiert binär suchende Algorithmus verwendet.

Das Gerüst des Algorithmus Das Gerüst des Algorithmus basiert auf dem generischen Gerüst für die randomisierte Suche durch Interpolation (siehe Abschnitt 5.2, bzw. Algorithmus 5.1): Wie auch für den vorgestellten *slope selection*-Algorithmus wird in mehreren Iterationen ein Suchstreifen, von dem ermittelt wurde, dass er den gesuchten Schnittpunkt enthält, sukzessive verkleinert. In jeder Iteration wird hierfür zunächst für die Lage des gesuchten *repeated median* eine Schätzung berechnet und nachfolgend ein Suchstreifen geschätzt, vom dem vermutet wird, dass er den gesuchten Schnittpunkt enthält. Abschließend wird in jedem Iterationsschritt diese Vermutung überprüft und im positiven Falle mit dem vermuteten (andernfalls mit dem bisherigen) Suchstreifen weiter iteriert. Wie beim *slope selection*-Algorithmus kann die so iterierte Suche abgebrochen werden, sobald der Suchstreifen nur noch linear viele Kandidaten, d. h. Mediane m_i (der x-Koordinaten der Schnittpunkte von Geraden g_i) enthält. Die Suche kann nun deterministisch beendet werden, ohne dass die Komplexität der erwarteten Gesamtlaufzeit hierdurch wächst. Der entsprechende Algorithmus ist nachfolgend als Algorithmus 5.8 skizziert.

Auswahl: Die zufällige Auswahl erfolgt nicht wie für das *slope selection*-Problem aus der Menge der Schnittpunkte im aktuellen Suchintervall $\mathcal{S}$. Stattdessen wird zunächst eine Menge $\mathcal{R}$ von r Geraden zufällig (mit Zurücklegen) ausgewählt. Diese Auswahl erfolgt dabei aus der Menge C der Geraden, für die die mediane x-Koordinate unter all ihren Schnittpunkten im aktuellen Suchintervall $\mathcal{S}$ liegt. Pro Gerade werden anschließend $\tilde{r}$ ihrer Schnittpunkte zufällig ausgewählt.

[37] Die Aussage von Matoušek *et al.* im originalen Wortlaut: „[...] pathological situations can be imagined in which the estimates are so skewed that the algorithm's rate of convergence is affected.“

Schätzung: Die x-Koordinate des Medians m_i unter den Schnittpunkten einer ausgewählten Gerade g_i kann durch Medianbildung über ihre $\tilde{r}$ zufällig ausgewählten Schnittpunkte geschätzt werden. Die x-Koordinate des *repeated median* kann dann durch Medianbildung über die r so ermittelten x-Koordinaten geschätzt werden.

Die Bestimmung eines neuen Suchstreifens, vom dem vermutet wird, dass er die x-Koordinate des tatsächlichen *repeated median* enthält, erfolgt nun wie folgt: Bezeichne C, L bzw. R die Menge aller Geraden g, für die die mediane x-Koordinate unter allen Schnittpunkten von g im aktuellen Suchintervall $\mathcal{S}$ bzw. links davon bzw. rechts davon liegt. Bezeichne zudem $k := \lceil n/2 \rceil - |L|$ den Rang (bezüglich der Ordnung nach der x-Koordinate) des Medians der *geschätzten* Mediane aller Geraden in C. Definiere nun zusätzlich die folgenden Ränge innerhalb dieser Ordnung:

$$k_{b'} := \max(1,\ \lfloor r \cdot k/|C| \ -\ 3 \cdot \sqrt{r}/2 \rfloor), \tag{5.3}$$

$$k_{e'} := \min(r,\ \lfloor r \cdot k/|C| \ +\ 3 \cdot \sqrt{r}/2 \rfloor) \tag{5.4}$$

Der neue Suchstreifen $\mathcal{S}' = \langle b', e' \rangle$ definiert sich nun durch die x-Koordinaten der Schnittpunkte mit den Rängen $k_{b'}$ und $k_{e'}$ bezüglich genannter Ordnung.

Überprüfung: Die eine jede Iteration einer randomisierten Suche durch Interpolation abschließende Überprüfung des gemutmaßten verkleinerten Suchstreifens $\mathcal{S}' = \langle b', e' \rangle$ erfordert im vorliegenden Problemfall den tatsächlichen *repeated median* relativ zu diesem Streifen zu lokalisieren. Dies wiederum erfordert die Lokalisierung aller (bzw. für die Bestimmung des repeated median hinreichend vieler) Mediane m_i der einzelnen Geraden der Eingabe.

Diese Verortung erfolgt durch das Zählen der Schnittpunkte jeder Geraden g_i, die sich links, innerhalb bzw. rechts des Suchintervalls $\mathcal{S}' = \langle b', e' \rangle$ befinden, durch entsprechende Zähler L_i, C_i und R_i.[38] Dieses Zählen von Schnittpunkten und Lokalisieren von Medianen kann für alle Geraden simultan wie für die randomisierte binäre Suche in Abschnitt 5.4.1 beschrieben erfolgen.

Eine speichereffiziente Variante Die *in-place*-Realisierung des beschriebenen Algorithmus adaptiert für die zufällige Auswahl die Resultate aus Abschnitt 5.3. Da nur Geraden bzw. nur Schnittpunkte zu einer Geraden zu protokollieren sind, ist während dieser Protokollierung keine Abarbeitung des Inversionen zählenden Algorithmus vonnöten, was die Kodierung der Auswahl erheblich erleichtert. Der Auswahlschritt ist daher in $\mathcal{O}(r \cdot \tilde{r} \cdot \log_2 n)$ Zeit und *in-place* zu bewältigen.

Der *in-place* schwieriger zu realisierende Teil des Algorithmus ist — wie für die randomisierte binäre Suche nach dem *repeated median*— die Lokalisierung des *repeated median* bezüglich eines Streifens, um Klarheit über die im nächsten Suchschritt zu verwendende Suchdomäne zu erzielen. Die Lösung dieser Schwierigkeit ergibt sich ebenfalls analog zu den in Abschnitt 5.4.1 skizzierten Ausführungen: Es bedarf des Zählens der Schnittpunkte, separat für n Geraden und in jeweils einer konstanten Anzahl von Streifen. Folglich ist mit Lemma 5.4.2 die Verortung des für den randomisiert durch Interpolation suchenden Algorithmus in derselben Laufzeit wie für

[38] Da für jede Gerade g_i nach Abschluss des Zählvorgangs $L_i + C_i + R_i = n$ gilt, genügt es, nur die Zähler L_i und C_i vorzuhalten.

den binär randomisiert suchenden Algorithmus möglich. Da diese Verortung für geeignete Wahlen von r und $\tilde{r}$ die Laufzeitkosten eines Suchschrittes dominiert, ist die erwartete Laufzeit der Suche durch Interpolation nach dem *repeated median* in $\mathcal{O}(n \log_2^3 n)$.

Es bleibt zu bemerken, dass die Beobachtung von Matoušek *et al.* [MMN98], dass in der Praxis für viele nicht entartete Eingabekonfigurationen die benötigte Anzahl an Suchschritten konstant (in der Eingabegröße) ist, ohne Einschränkungen auch für die *in-place*-Variante zu erwarten ist — insbesondere da die *in-place* darstellbare Gesamtgröße $r \cdot \tilde{r}$ der zufälligen Auswahl in einem Suchschritt mit $\mathcal{O}(n \log_2 n)$ in der asymptotischen Komplexität nahe an der von Matoušek *et al.* verwendeten linearen Auswahlgröße liegt.

5.4.3 Generalisierung zur Behandlung entarteter Konfigurationen und duplikatbehafteter Eingaben

Matoušek *et al.* [MMN98] verweisen für die Behandlung von nicht in allgemeiner Lage befindlichen Eingaben, also für Eingaben mit zumindest drei kollinearen Punkten, auf einen technischen Bericht von Mount und Netanyahu [MN91]. Diese Konfigurationen wie auch duplikatbehaftete Eingaben lassen sich wie für die Bestimmung des Theil-Sen-Schätzers in Bemerkung 5.3.10 ausgeführt behandeln.

Somit ergibt sich für die Bestimmung des *repeated median*-Schätzers für Duplikate enthaltende Eingaben als Verallgemeinerung von Satz 5.4.1 ein ähnliches Resultat wie für die Bestimmung des Theil-Sen-Schätzers, vergleiche Korollar 5.3.21:

Korollar 5.4.3 *Die Aussage von Satz 5.3.2 ist für ein beliebiges $c > 1/2$ auch für die Menge P_c von Eingaben, die mindestens $c \cdot n$ unterscheidbare Elementen enthalten, gültig.*

Beweis: Der Beweis verläuft analog zum Beweis von Korollar 5.3.21. Allerdings werden nicht die Aussagen der Lemmata 5.2.1, 5.2.2 und 5.2.4 benötigt, da keine konstante Anzahl an Iterationen der randomisierten Suche durch Interpolation zum Erhalten der in Satz 5.4.1 angegebenen Laufzeit erforderlich ist. Um die gewünschte Laufzeit zu gewährleisten, ist im vorliegenden Problemfall und für den vorgestellten Algorithmus der Bedarf an kodierenden Eingabeelementen in $\Theta(n)$, vergleiche die Laufzeitanalyse in Abschnitt 5.4.1. Dieser Bedarf ist durch die Wahl von c gedeckt: In jeder beliebigen Aufteilung der Eingabe in zwei gleich große Hälften stehen zumindest $(c - 1/2) \cdot n \in \Theta(n)$ unterscheidbare Eingabeelemente zur Verfügung. □

Algorithmus 5.8 Algorithmus RepeatedMedianInterpolated($\mathtt{A}[0,\ldots,n-1], r$) bestimmt durch randomisierte Suche durch Interpolation den *repeated median* unter den Schnittpunkten der Geraden in $\mathtt{A}$ [MMN98].

1: $b := -\infty$; $e := \infty$. /* Erste „Schätzung“ für einen den *repeated median* enthaltenden Suchstreifen. */
2: **repeat**
3: Wähle (mit Zurücklegen) eine Menge $\mathcal{R}$ der Größe r von zufällig aus der Eingabe ausgewählten Geraden
4: **for each** $g \in \mathcal{R}$ **do**
5: Wähle zufällig eine Menge $\mathcal{R}_g$ von $\tilde{r}$ Schnittpunkten von g aus.
6: **end for**
7: Schätze für alle $g \in \mathcal{R}$ an Hand von $\mathcal{R}_g$ die x-Koordinate des medianen Schnittpunktes m_g von g.
8: Setze $\kappa := \lfloor(\lceil n/2\rceil - |\mathcal{M}(-\infty, b)|)\cdot(r/|\mathcal{M}(b,e)|)\rfloor$, $\kappa_{b'} := max(1, \lfloor\kappa - 3\sqrt{r}/2\rfloor)$, und $\kappa_{e'} := min(r, \lfloor\kappa + 3\sqrt{r}/2\rfloor)$.
9: Bestimme Schnittpunkte mit Rängen $\kappa_{b'}$ und $\kappa_{e'}$ in $\mathcal{R}$.
10: Bestimme die Mediane m_g mit Rängen $\kappa_{b'}$ und $\kappa_{e'}$ (bzgl. $<_x$-Ordnung) in $\mathcal{R}$.
11: Wähle b' (sowie e') als die x-Koordinate des Medians in $\mathcal{R}$ mit Index $\kappa_{b'}$ (bzw. $\kappa_{e'}$).
12: **if** $|\mathcal{M}(-\infty, b')| < \lceil n/2\rceil \leq |\mathcal{M}(-\infty, e')|$ **then**
13: $b := b'$; $e := e'$. /* Der *repeated median* befindet sich in $\langle b', e'\rangle$; verwende fortan $\langle b', e'\rangle$ als Suchstreifen. */
14: **end if**
15: **until** $|\mathcal{M}(b', e')| = 1$ /* Der *repeated median* ist bestimmt. */
16: Gebe den verbliebenen medianen Schnittpunkt (bzw. sein Dual) als den gesuchten *repeated median* zurück.

Kapitel 6

Speichereffiziente Berechnung von Maximamengen und -schichten

6.1 Einleitung

In diesem Kapitel wird die Berechnung der Maximamenge bzw. der Maximaschichten von zwei- und dreidimensionalen Punktmengen mit der Zielsetzung der speicher- und laufzeitoptimalen Realisierung betrachtet [BV06a].

Entsprechend den Begriffsbildungen bei Kung *et al.* [KLP75] *dominiert* ein Punkt p einen Punkt q genau dann, wenn in jeder Dimensionen die Koordinate von p größer als die Koordinate von q ist. Ein Punkt p einer Punktmenge $\mathcal{P}$ wird genau dann als ein *Maximum* dieser Menge bezeichnet, wenn er von keinem Punkt aus $\mathcal{P}$ dominiert wird. Die Menge aller Maxima in $\mathcal{P}$ wird als die *Maximamenge* $\texttt{MAX}(\mathcal{P})$ von $\mathcal{P}$ bezeichnet.[1]

Eine Erweiterung dieser Notation führt zum Begriff der *Maximaschichten* einer Punktmenge $\mathcal{P}$: Die erste bzw. oberste dieser Schichten ist die *Maximamenge* $\texttt{MAX}(\mathcal{P})$ von $\mathcal{P}$. Die weiteren Schichten von $\mathcal{P}$ ergeben sich iterativ jeweils aus dem Entfernen der Maximamenge und dem anschließenden Berechnen der Maxima der verbliebenen Punktmenge $\mathcal{P}' := \mathcal{P} \setminus \texttt{MAX}(\mathcal{P})$, bis alle Punkte von $\mathcal{P}$ einer Maximaschicht zugeordnet sind.

Verwandte algorithmische Resultate Die Bestimmung der Maxima einer n-elementigen Menge, die auch als *skyline*-Operation bezeichnet wird, kann als Aggregationsoperation angesehen werden. Dieses Problem besitzt nach Preparata und Shamos [PS85] eine Vielzahl von Anwendungen (ähnlich wie die in Kapitel 5 vorgestellten Operationen) in der Statistik sowie zusätzlich in *Operations research*-Anwendungen [HL05], in denen Optimierungen verschiedener ökonomischer Prozesse erforscht werden. Es wurde daher als eines der ersten Probleme in der algorithmischen Geometrie eingehender untersucht: Für zwei- und dreidimensionale Eingaben

[1] In der Literatur wird häufig das vom Berechnungsaufwand äquivalente Minimaproblem betrachtet. In diesem Fall kehrt sich die Ordnungsrelation in der Definition der Dominanz um. Verallgemeinert kann die Dominanz und damit der Begriff Maximum so spezifiziert werden, dass in jeder Dimension, d. h. für jede Koordinate der zu betrachtenden Punkte, die Richtung der Ordnungsrelation vorgegeben werden kann, vergleiche Börzsönyi *et al.* [BKS01].

sind Algorithmen mit optimaler asymptotischer Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ bereits von Kung *et al.* [KLP75] publiziert worden. Die Optimalität folgt aus der Transformierbarkeit auf das Sortierproblem [KLP75, PS85]. Für Eingaben beliebiger, aber fester Dimension $d \geq 4$ liefert der Ansatz von Kung *et al.* [KLP75] einen Algorithmus mit $\mathcal{O}(n \cdot \log^{d-2} n)$ Laufzeit, vergleiche auch die Diskussion bei Bentley [Ben80].

Matoušek [Mat91a] stellte einen Algorithmus mit einer Laufzeit von $\mathcal{O}(n^{2.688})$ für den Fall $d = n$ vor. Das Problem der Maximabestimmung ist ferner für zweidimensionale dynamische Punktmengen sowie für höher-dimensionale Eingaben mit bestimmten Verteilungseigenschaften untersucht worden, vergleiche Kapoor [Kap94] bzw. Bentley *et al.* [BCL90] und Dai *et al.* [DZ04].

Für die Bestimmung der Maximaschichten zwei- und dreidimensionaler Punktmengen präsentierten Buchsbaum und Goodrich [BG04] jeweils Algorithmen mit optimaler Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$. Ihr Ansatz für dreidimensionale Eingaben basiert auf dem *plane-sweep*-Paradigma [PS85] und verwendet die Technik des *dynamischen fraktalen Kaskadierens* [MN90], um eine Datenstruktur zur Punktlokalisierung effizient aufrecht zu halten. Letztere entspricht einer dynamischen Struktur von zweidimensionalen Maximaschichten, die während eines Überstreichens der Eingabe entlang einer dritten Dimension aufrechterhalten wird.

Ein vergleichbares Problem der Aufteilung einer mehrdimensionalen Eingabe nach Schichten ist von Overmars and van Leeuwen [OvL81] behandelt worden: Sie beschreiben einen Algorithmus mit einer Laufzeit von $\mathcal{O}(n \cdot \log_2^2 n)$, der das plane-sweep-Paradigma und das dynamische fraktale Kaskadieren nutzt, um die *konvexen Schichten* einer Eingabe zu bestimmen.[2] Chazelle [Cha85] stellte später eine effizientere Variante dieses Algorithmus vor, die die optimale Laufzeit von $\mathcal{O}(n \log_2 n)$ aufweist.

Das Maximaproblem ist auch in der Datenbankforschung untersucht worden. Börzsönyi *et al.* [BKS01] definierten als Erweiterung der Anfragesprache SQL, vergleiche [SKS05, Kapitel 3 und 4], einen `skyline`-Operator, der die Maxima einer Menge von Punkten beliebiger, aber fester Dimension berechnet. Die Einbindung eines solchen Operators wird unter anderem motiviert durch Anfragen der folgenden Art: „Finde alle Hotels, die zugleich preiswert und in Nähe zum Strand gelegen sind.“.[3] Im Kontext lokationsbasierter Dienste ist die Nähe zum Standort des Anfragestellers häufig ein weiteres wichtiges Anfragekriterium, vergleiche auch Kapitel 4. Dies impliziert insbesondere, dass für diese Art der Anfrage in lokationsbasierten Diensten Ergebnisse nicht vorberechnet, sondern für einen individuellen Nutzer — eventuell wiederholt — berechnet werden müssen, was eine zusätzliche Motivation für effiziente und auf mobilen Geräten verwendbare Algorithmen für Maximaprobleme liefert.

Für die Berechnung von Maximamengen in verschiedenen praktischen Szenarien, sind in jüngerer Zeit eine Vielzahl von Strategien, die räumliche Indizes nutzen,

[2] Wie Maximaschichten ergeben sich konvexe Schichten durch iterative Berechnung der jeweils obersten Schicht (hier: der konvexen Hülle) und der anschließenden Betrachtung der verbleibenden Eingabe.

[3] Präzise formuliert liefert die skyline-Operation in diesem Kontext die Antwort auf die Anfrage: „Finde alle Hotels, die von keinem anderen der betrachteten Hotels sowohl hinsichtlich Preiswürdigkeit als auch hinsichtlich Strandnähe übertroffen werden.“

vorgestellt und untersucht worden [KRR02, PTFS05, TEO01]. Für keinen dieser Ansätze, von denen ein Großteil auf progressiven und online-Berechnungen fokussiert, sind nicht-triviale obere Laufzeitschranken bekannt.

Brönnimann *et al.* stellten, wie schon in Kapitel 4 erwähnt, *in-place*-Algorithmen für die Berechnung planarer konvexer Hüllen vor und gaben an, dass die vorgestellten algorithmischen Methoden sich auch für die ausgabesensitive *in-place*-Berechnung planarer Maximamengen in optimaler Zeit von $\mathcal{O}(n \cdot \log_2 h)$ adaptieren lassen, wobei h die Anzahl der Maxima der Eingabe bezeichnet.

Resultate dieser Arbeit In diesem Kapitel werden *in-place*-Algorithmen mit optimaler Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ für die Bestimmung der Maximamenge in zwei und drei Dimensionen sowie für die Bestimmung und *in-place*-Arrangierung der Maximaschichten in zwei Dimensionen vorgestellt.

Abgesehen von der praktischen Relevanz dieser Algorithmen ist insbesondere der letzgenannte Algorithmus von theoretischem Interesse, da er der erste laufzeitoptimale *in-place*-Algorithmus für ein geometrisches Problem ist, für dessen Lösung die Eingabe komplett zu ordnen ist, welches aber nicht allein auf Basis des divide & conquer-Prinzips oder mit Hilfe von Grahams Scan [Gra72], effizient lösbar scheint.

6.2 Berechnungen von Maximamengen

Wie erläutert ist ein Punkt p aus einer Punktmenge $\mathcal{P}$ *maximal* bezüglich $\mathcal{P}$ genau dann, wenn die Koordinatenwerte keines anderen Punktes sämtlich größer sind als die von p. Auf Grundlage dieser Definition ergeben sich der plane-sweep-Algorithmus für zweidimensionale Punktmengen und der divide-conquer-Algorithmus für Mengen höherdimensionaler Punkte, die jeweils von Kung *et al.* [KLP75] vorgestellt wurden. Zu diesen Algorithmen lassen sich speichereffiziente und gleichzeitig laufzeitoptimale Realisierungen konstruieren.

Die Ausgabe dieser speichereffizienten Varianten besteht jeweils aus einer Permutation der Eingabe `A` und einem Index k, der die Mächtigkeit der Maximamenge angibt; die Elemente der Maximamenge liegen nach Ablauf des Algorithmus absteigend nach der y-Koordinate (und aufsteigend nach der x-Koordinate) sortiert in $\texttt{A}[0, \ldots, k-1]$ vor. Ein Durchlauf durch die ersten k Punkte im Eingabefeld zeichnet demnach — von links nach rechts — die Skyline der Eingabe nach.

6.2.1 Skyline-Berechnung in zwei Dimensionen

Der Algorithmus von Kung *et al.* zur Berechnung der Skyline in zwei Dimensionen ist ein einfacher Selektionsalgorithmus, der sich für eine speichereffiziente Realisierung auf den in Abschnitt 4.1.2 beschriebenen stabilen Teilmengenextraktionsalgorithmus von Bose *et al.* [BMM$^+$06] abstützt.

Der Algorithmus von Kung *et al.* überstreicht die zuvor absteigend nach $<_y$ sortierte Punktmenge und hält dabei beide Koordinaten des bislang tiefsten Punktes m der Skyline vor. Für jeden überstrichenen Punkt p wird seine Zugehörigkeit zur Skyline überprüft. Hierfür ist es ausreichend zu testen, ob p von m nicht dominiert wird.

Dies ist genau dann der Fall, wenn $p.x \geq m.x$ gilt, da die Abarbeitungsreihenfolge sichert, dass zusätzlich $p.y \leq m.y$ gilt.

Ein speichereffizienter Algorithmus ergibt sich durch das initiale Sortieren der Eingabe nach absteigender y-Koordinate, was *in-place* und in $\mathcal{O}(n \cdot \log_2 n)$ unter Verwendung von heapsort [Flo64] möglich ist, vergleiche Kapitel 4.1.2.

Das Überstreichen kann durch Anwendung des erwähnten stabilen Teilmengenextraktionsalgorithmus in linearer Laufzeit und *in-place* durchgeführt werden. Zu selektieren sind in dieser Anwendung des Algorithmus von Bose *et al.* [BMM+06] die Maxima unter den Eingabepunkten. Das Prädikat π, das die Maximaeigenschaft eines Punktes überprüft, wird für jeden überstrichenen Punkt $\mathtt{A}[i]$ evaluiert und ist genau dann erfüllt, wenn die x-Koordinate dieses Punktes mindestens so groß ist wie die x-Koordinate des Punkts m. In diesem Fall wird m aktualisiert (d. h. durch $\mathtt{A}[i]$ ersetzt) und $\mathtt{A}[i]$ mit dem Eingabeelement $\mathtt{A}[k']$ getauscht, wobei k' die Anzahl der bislang gefundenen Maxima angibt. Die Speicherung und Aktualisierung der Koordinaten von m sowie des Indexes k' ist mit konstantem zusätzlichem Speicher möglich, so dass sich folgendes Resultat ergibt:

Satz 6.2.1 *Die* Skyline *einer Menge $\mathcal{P}$ zweidimensionaler Punkte kann* in-place *und in optimaler asymptotischer Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ berechnet werden. Ist $\mathcal{P}$ bereits nach $<_y$ sortiert, reduziert sich die Laufzeitkomplexität auf $\mathcal{O}(n)$.*

6.2.2 Skyline-Berechnung in drei Dimensionen

Für dreidimensionale Eingaben entwickelten Kung *et al.* [KLP75] einen divide & conquer-Algorithmus. Dieser unterteilt die Punktmenge $\mathcal{P}$ bezüglich der in $\mathcal{P}$ medianen z-Koordinate ζ in $\mathcal{P}_{z \leq \zeta}$ and $\mathcal{P}_{z > \zeta}$, um für jede dieser beiden Teilmengen ihre Maximamenge rekursiv zu berechnen.

Für diese Berechnung können die beiden folgenden Beobachtungen ausgenutzt werden:

Beobachtung 6.2.2 *Da alle Punkte in $\mathcal{P}_{z \leq \zeta}$ kleinere z-Koordinaten als alle Punkte in $\mathcal{P}_{z > \zeta}$ haben, kann kein Punkt in $\mathcal{P}_{z > \zeta}$ von einem Punkt in $\mathcal{P}_{z \leq \zeta}$ dominiert werden. Somit sind Maxima von $\mathcal{P}_{z > \zeta}$ generell auch Maxima von $\mathcal{P} = \mathcal{P}_{z \leq \zeta} \cup \mathcal{P}_{z > \zeta}$.*

Beobachtung 6.2.3 *Die Maxima von $\mathcal{P}$, die nicht zu $\mathcal{P}_{z \leq \zeta}$ gehören, sind genau die Maxima von $\mathcal{P}_{z \leq \zeta}$, die von keinem Punkt in (der Maximamenge von) $\mathcal{P}_{z > \zeta}$ dominiert werden.*

Die Maxima von $\mathcal{P}$, die aus $\mathcal{P}_{z \leq \zeta}$ stammen, können während des Schrittes des divide & conquer-Algorithmus zur Verschmelzung von $\mathcal{P}_{z \leq \zeta}$ und $\mathcal{P}_{z > \zeta}$ identifiziert werden. Ein solcher Verschmelzungsschritt muss das Maximaproblem nur in der (x, y)-Ebene betrachten, da die jeweiligen Teilmengen bezüglich eines z-Wertes separiert sind und somit für zwei Punkte aus *verschiedenen* Teilmengen ihre relative Ordnung bezüglich $<_z$ generell bekannt ist. Dieses dimensionsreduzierte Problem kann (ähnlich wie in Kapitel 6.2.1 beschrieben) durch einen Sweep-Algorithmus gelöst werden, in dem simultan die Maximamengen von $\mathcal{P}_{z \leq \zeta}$ und von $\mathcal{P}_{z > \zeta}$ in absteigender $<_y$-Ordnung überstrichen werden. Dabei wird das letzte (d. h. tiefste) bislang

gefundene Maximum m von $\mathcal{P}_{z>\zeta}$ aufrecht gehalten. Ein Maximum p von $\mathcal{P}_{z\leq\zeta}$, das während eines Verschmelzungsschrittes bearbeitet wird, ist genau dann ein Maximum auch von $\mathcal{P}$, wenn seine x-Koordinate nicht kleiner als die x-Koordinate von m ist: Da m vor p bearbeitet wurde und da m ein Maximum von $\mathcal{P}_{z>\zeta}$ ist, sind sowohl die y- als auch die z-Koordinate von m kleiner als die von p. Um für eine Abarbeitung des beschriebenen divide & conquer-Algorithmus nur konstanten zusätzlichen Speicher zu verwenden, kann das in Kapitel 4.1.2 erläuterte und von Bose *et al.* [BMM+06] vorgestellte Gerüst für die speichereffiziente Realisierung von rekursiven Algorithmen genutzt werden, welches als Algorithmus 6.1 schematisch skizziert ist.

Algorithmus 6.1 REKURSION($\mathtt{A}, b, e$): Rekursive divide & conquer-Abarbeitung von $\mathtt{A}[b, \ldots, e-1]$ [BMM+06].

```
1: if e − b ≤ s, mit s als der Größe der Rekursionsbasis then
2:    BASE-CODE(A, b, e)                    /* Anweisungen für kleine Instanzen. */
3: else
4:    PRE-CODE(A, b, e) /* Behandle Teilproblem 1 in A[b, ..., ⌊(b + e)/2⌋ − 1]. */
5:    REKURSION(A, b, ⌊(b + e)/2⌋)          /* Erster Rekursiver Aufruf. */
6:    MID-CODE(A, b, e) /* Behandle Teilproblem 2 in A[⌊(b + e)/2⌋, ..., e − 1]. */
7:    REKURSION(A, ⌊(b + e)/2⌋, e)          /* Zweiter Rekursiver Aufruf. */
8:    POST-CODE(A, b, e)                    /* Verschmelze Teilprobleme 1 und 2 in
      A[b, ..., e − 1]. */
9: end if
```

Das abgebildete Gerüst für einen rekursiven Algorithmus zerlegt das Teilfeld $\mathtt{A}[b, \ldots, e-1]$ solange rekursiv in jeweils zwei gleich große Teilfelder, bis diese nur noch s Eingabeelemente enthalten, wobei s die Anzahl der Elemente angibt, die — ohne eine weitere rekursive Unterteilung — durch einen Aufruf von BASE-CODE behandelt werden können. Nach einer jeweiligen Unterteilung eines Teilfeldes ruft der Algorithmus sich rekursiv einmal mit $\mathtt{A}[b, \ldots, e/2-1]$ und einmal mit $\mathtt{A}[e/2, \ldots, e-1]$ als Eingabe auf. Sind diese Teilfelder rekursiv abgearbeitet, werden sie durch die Anweisungen in POST-CODE (der Aufgabenstellung entsprechend) zusammengeführt. Die Details der speichereffizienten Organisation der rekursiven Abarbeitung sind dem Artikel von Bose *et al.* [BMM+06] zu entnehmen.

Der nachfolde beschriebene *in-place*-Algorithmus nutzt das skizzierte Gerüst für die rekursiven Berechnung aller Maxima einer Menge dreidimensionalen Punkte und hält dabei zwei Invarianten aufrecht:

Invariante (1): Vor der Abarbeitung von PRE-CODE ist das im aktuellen Aufruf betrachtete Teilfeld absteigend bezüglich $<_y$ sortiert.

Invariante (2): Nach der Abarbeitung von POST-CODE ist die Maximamenge des im aktuellen Aufruf betrachteten Teilfeldes zu Beginn dieses Teilfeldes gespeichert und absteigend bezüglich $<_y$ sortiert.

Invariante (1) kann durch Sortierung des gesamten Feldes zu Beginn der Abarbeitung hergestellt werden. Diese Sortierung kann aufrecht gehalten werden, indem

die Unterteilung in Teilprobleme durch eine stabile Medianbestimmung (bezüglich der $<_z$-Ordnung) vorgenommen wird, vergleiche Abschnitt 4.1.2 bzw. [BMM+06]. Invariante (2) kann für die Basisteilprobleme (für zwei Punkten, d. h. für $s = 2$) in trivialer Weise (in Base-Code) erfüllt werden. Somit darf per Induktion vor einer Abarbeitung von Post-Code von der nachfolgend skizzierten Struktur des Teilfeldes $\mathtt{A}[b, \ldots, e-1]$, das die im aktuellen Aufruf abzuarbeitende Punktmenge $\mathcal{P}$ enthält, ausgegangen werden. Die Indizes m_l und m_r markieren das jeweilige Ende der Maximamengen, die in den beiden vorangegangenen rekursiven Aufrufen für die entsprechenden Teilfelder $\mathtt{A}[b, \ldots, e/2-1]$ und $\mathtt{A}[e/2, \ldots, e-1]$ berechnet wurden.

...	Maxima von $\mathcal{P}_{z\leq\zeta}$		Maxima von $\mathcal{P}_{z>\zeta}$		...
	b	m_l	$\lfloor e/2 \rfloor$	m_r	e

Das Gerüst von Bose *et al.* für speichereffiziente divide & conquer-Algorithmen ermöglicht leider nicht, mit nur konstantem zusätzlichem Speicher die *Anzahl* der Maxima aller Teilprobleme aufrecht zu erhalten. Daher müssen die Indizes m_l und m_r sowie die mediane z-Koordinate ζ für die Verschmelzung von $\mathcal{P}_{z\leq\zeta}$ und $\mathcal{P}_{z>\zeta}$ (in Post-Code) algorithmisch rekonstruiert werden. Dies ist in einem einzigen linearen Durchlauf durch $\mathtt{A}[b, \ldots, e-1]$ möglich, da zum einen jede Maximamenge in absteigender $<_y$- und aufsteigender x-Ordnung vorliegt (und das erste nicht maximale Elemente im jeweiligen Teilfeld an der Verletzung dieser Sortierung erkannt werden kann) und zum anderen die mediane z-Koordinate ζ sich durch die maximale z-Koordinate in $\mathtt{A}[b, \ldots, \lfloor e/2 \rfloor - 1]$ bestimmen lässt. Anschließend kann ein *in-place*-Verschmelzungsalgorithmus mit linearer Laufzeit wie der von Geffert *et al.* [GKP00] publizierte, siehe Kapitel 4.1.2, verwendet werden, um die Maxima *beider* Teilmengen in absteigende $<_y$-Ordnung zu überführen und am Anfang des zu betrachtenden Feldes zu platzieren.

...	Maxima von $\mathcal{P}_{z\leq\zeta} \cup$ Maxima von $\mathcal{P}_{z>\zeta}$		...
	b	m	e

Diese Vereinigung der Maxima der Teilprobleme wird nun durch den in Abschnitt 4.1.2 beschriebenen stabilen Teilmengenextraktionsalgorithmus von Bose *et al.* [BMM+06] abgearbeitet, wobei das Prädikat π, das die Maxima der aktuell abzuarbeitenden Probleminstanz identifiziert, die Beobachtungen 6.2.2 und 6.2.3 ausnutzt, was in Algorithmus 6.2 dargestellt ist.

Der Algorithmus von Bose *et al.* ist in dem Sinne stabil, dass die $<_y$-Sortierung des selektierten Teiles seiner Eingabe, also der Maximamenge des aktuell abzuarbeitenden Teilfeldes, erhalten bleibt. Daher etabliert der Schritt für das Verschmelzen zweier Teilprobleminstanzen die Invariante (2). Dies geschieht in linearer Zeit, da das assoziierte Prädikat in konstanter Zeit ausgewertet werden kann.

Somit führt die Laufzeitanalyse des Algorithmus von Kung *et al.* in Kombination mit der Analyse des verwendeten divide & conquer-Gerüstes für speichereffiziente Algorithmen zu folgendem Resultat:

Algorithmus 6.2 Prädikat IstMaximum($\mathtt{A}, i, x_{\text{current}}, \zeta$) zur Identifikation der Maxima von $\mathcal{P}$ während eines Überstreichens der in absteigender y-Richtung sortierten, vereinigten Maxima von $\mathcal{P}_{z \leq \zeta}$ und von $\mathcal{P}_{z > \zeta}$.

Vorbedingung: ζ bezeichnet die für die Problemaufteilung verwendete mediane z-Koordinate und x_{current} bezeichnet die x-Koordinate des tiefsten (bislang entdeckten) Maximums von $\mathcal{P}_{z > \zeta}$.

```
1: if A[i].z ≤ ζ then /* 1. Fall: A[i] ∈ P_{z≤ζ} */
2:    if A[i].x ≥ x_current then
3:       Gib true zurück.        /* A[i] ist (in x-Richtung) nicht dominiert, also ein
         Maximum. */
4:    else
5:       Gib false zurück.       /* A[i] ist dominiert, also kein Maximum. */
6:    end if
7: else /* 2. Fall: A[i] ∈ P_{z>ζ} */
8:    if A[i].x ≥ x_current then
9:       Set x_current := A[i].x  /* Aktualisiere die x-Koordinate des untersten,
         bislang gefundenen Maximums von P_{z>ζ}. */
10:   end if
11:   Gib true zurück.            /* Jedes A[i] ∈ P_{z>ζ} ist ein Maximum. */
12: end if
```

Satz 6.2.4 *Die Skyline einer n-elementigen Menge dreidimensionaler Punkte kann* in-place *und in optimaler asymptotischer Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ berechnet werden.*

6.3 Berechnen und Arrangieren von Maximaschichten

Die Resultate aus den Sätzen 6.2.1 und 6.2.4 implizieren in trivialer Weise iterative Algorithmen für das Berechnen aller Maximaschichten einer zwei- oder dreidimensionalen Punktmenge: Jede einzelne Maximaschicht kann bestimmt werden und anschließend — entsprechend den Ausführungen in Kapitel 6.1 — die verbleibende Punktmenge iterierend betrachtet werden. Da eine Eingabe eine lineare Anzahl an Maximaschichten aufweisen kann (etwa wenn alle Punkte auf der Diagonalen $y = x$ liegen), ergibt sich die Laufzeit dieses Ansatzes (im schlimmsten Fall) zu $\Theta(n^2 \log_2 n)$.

In diesem Kapitel wird ein iterativer Algorithmus vorgestellt, der die Maximaschichten einer zweidimensionalen Punktmenge berechnet und dabei in einzelnen Iterationen mehrere oberste Schichten gleichzeitig bestimmt und entfernt, so dass seine Laufzeit asymptotisch optimal, d. h. von der Komplexität $\mathcal{O}(n \cdot \log_2 n)$ ist. Zusätzlich arrangiert dieser Algorithmus die Eingabe nach Schichten (von außen nach innen) und die Punkte jeder einzelnen Schicht sind absteigend nach ihrer y-Koordinate sortiert. Die einzelnen Iterationen dieses Algorithmus bestehen aus mehreren Sweep-Durchläufen, die allesamt als Varianten des in Kapitel 6.2.1 beschriebenen Sweep-Algorithmus angesehen werden können.

6.3.1 Berechnen der Anzahl der Maximaschichten

Der in Kapitel 6.2.1 vorgestellte Algorithmus zur Berechnung der (obersten) Maximaschicht $\mathcal{L}_0$ lässt sich insbesondere adaptieren, um die Anzahl k der Maximamaschichten $\mathcal{L}_0, \ldots, \mathcal{L}_k$ einer Punktmenge zu bestimmen. Diese Adaption nutzt den Sachverhalt aus, dass eine jede Maximaschicht monoton sowohl in x- als auch in y-Richtung ist: Eine Schicht $\mathcal{L}_i$ setzt sich vom niedrigsten ihrer konstituierenden Punkte aus vertikal nach $y = -\infty$ fort. Dies impliziert, dass während des Überstreichens durch den Algorithmus eine Schicht $\mathcal{L}_i$ hinreichend durch ihren Schnitt mit der (vertikal absteigenden) Sweep-Geraden bestimmt ist und dass dieser Schnitt durch die x-Koordinate ihres „Ausläufers“, d. h. des Punktes, der zuletzt als zur Schicht $\mathcal{L}_i$ gehörend klassifiziert wurde, bestimmt ist.

Wie für *in-place*-Algorithmen üblich, wird davon ausgegangen, dass die Eingabe, d. h. die zu bearbeitende Punktmenge $\mathcal{P}$, in einem Feld $\mathtt{A}[0, \ldots, n-1]$ übergeben wird. Während des Überstreichens kann folgende Invariante zugesichert werden:

Ausläufer-Invariante: Sei $k_{act} \in \{1, \ldots, n\}$ die Anzahl der Schichten, die von der an $y = p.y$ befindlichen Sweep-Geraden geschnitten werden, wobei p der zuletzt überstrichene Punkt sei. Dann sind die Ausläufer $\tau_0, \ldots, \tau_{k-1}$ der Maximaschichten $\mathcal{L}_0, \ldots, \mathcal{L}_{k_{act}-1}$ nach absteigender x-Koordinate in $\mathtt{A}[0, \ldots, k_{act}-1]$ gespeichert.

Im rechten Teil der Abbildung 6.1 ist diese Invariante während des Überstreichens der Eingabe und direkt vor der Abarbeitung des Punktes p dargestellt und im linken Teil der Abbildung geometrisch für Beispielpunkte $p = p_{i_1}$ und $p = p_{i_2}$ veranschaulicht.

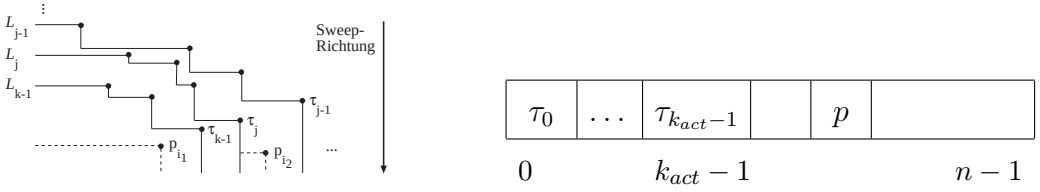


Abbildung 6.1: Klassifizierung von Punkten durch binäre Suche auf Ausläufern (links) und die resultierende Struktur des Eingabefeldes (rechts).

Die Ausläufer-Invariante ist trivialerweise nach Überstreichen des ersten zu bearbeitenden Punktes erfüllt: Dieser Punkt ist der (bezüglich y) höchste Punkt der Eingabe und daher der ersten Ausläufer der obersten Maximaschicht $\mathcal{L}_0$. Entsprechend der Invariante ist er bereits in $\mathtt{A}[0]$ gespeichert. Daher sei mit dem Induktionsprinzip angenommen, dass die Ausläufer-Invariante vor dem Überstreichen eines Punktes $p := \mathtt{A}[j]$ erfüllt ist. Um die Maximaschicht zu identifizieren, zu der p beiträgt, genügt eine binäre Suche nach der x-Koordinate von p in $\mathtt{A}[0, \ldots, k_{act} - 1]$. Falls die x-Koordinate von p kleiner als die von $\tau_{k_{act}-1} = \mathtt{A}[k_{act} - 1]$ und damit auch kleiner als die der Ausläufer aller bislang gefundenen Schichten ist, so ist p der erste Ausläufer einer neuen Schicht $\mathcal{L}_{k_{act}}$. Dieser Fall tritt in Abbildung 6.1 bei Überstreichen des Punktes $p = p_{i_1}$ ein. Die Ausläufer-Invariante wird nun wiederhergestellt,

indem $p = \mathtt{A}[j]$ mit $\mathtt{A}[k_{act}]$ vertauscht und k_{act} um eins inkrementiert wird.[4] Falls andererseits p rechts eines Ausläufers τ_l und gleichzeitig links eines Ausläufers τ_{l-1} liegt, so ersetzt p den Punkt τ_l als Ausläufer der Schicht $\mathcal{L}_j$. Folgerichtig ist $p = \mathtt{A}[j]$ mit $\mathtt{A}[l]$ zu vertauschen. Dieser Fall ergibt sich beim Überstreichen von $p = p_{i_2}$ in Abbildung 6.1. Analog ersetzt p den Ausläufer τ_0, falls p rechts von diesem liegt.[5]

Der beschriebene Algorithmus verwendet nur konstanten zusätzlichen Speicher (für den Status des Überstreichens sowie für den Index k, der die Anzahl der bisher gefundenen Schichten angibt), um die Ausläufer-Invariante aufrecht zu halten. Das Aufrechthalten selbst kostet pro überstrichenem Eingabepunkt $\mathcal{O}(\log_2 n)$ Zeit für die binäre Suche auf den bislang gefundenen Ausläufern. Die Anzahl an Maximaschichten der Eingabe ergibt sich durch $k = k_{act}$, d. h. als die Anzahl der Ausläufer nach Überstreichen aller Eingabepunkte.

Lemma 6.3.1 *Die Anzahl der Maximaschichten einer n-elementigen Eingabe zweidimensionaler Punkte kann* in-place *und in $\mathcal{O}(n \cdot \log_2 n)$ Zeit berechnet werden.*

Der beschriebene Algorithmus kann des Weiteren in seiner Funktionalität erweitert werden, so dass er in Form eines Datenstroms jeden Punkt p der Eingabe, zusammen mit dem Index der Maximaschicht, der p angehört, ausgibt:

Korollar 6.3.2 *Für jeden Punkt einer n-elementigen Eingabe kann* in-place *und in $\mathcal{O}(n \cdot \log_2 n)$ Zeit die Maximaschicht, der er angehört, identifiziert werden. Existiert die Möglichkeit der Ausgabe auf ein externes Medium, können diese Informationen während des Ablaufes des entsprechenden Algorithmus ausgelesen werden.*

6.3.2 Berechnen der Anzahl der Punkte auf den κ obersten Schichten

Der Algorithmus aus Abschnitt 6.3.1 zur Berechnung der Anzahl der Maximaschichten kann für das Zählen der Punkte der obersten κ Schichten adaptiert werden. Für die Verständlichkeit der Darstellung wird diese Adaption zunächst unter der Annahme, dass $\mathcal{O}(\kappa)$ Wörter an zusätzlichem Speicher zur Verfügung stehen, erläutert. Diese Wörter werden verwendet, um jeweils einen Zähler c_i für Anzahl der Punkte der Schicht $\mathcal{L}_i$ $(i = 1, \ldots, \kappa)$ aufrecht zu halten. Die Annahme, dass ein Speicher von (in der Eingabegröße) nicht konstanter Größe zur Verfügung steht, widerspricht den Bedingungen für einen *in-place*-Algorithmus. In Abschnitt 6.3.4 wird daher ausgeführt, wie der Algorithmus zu modifizieren ist, so dass er *in-place* und ohne Verwendung obiger Annahme operiert.

Das Zählen der Punkte auf einer Schicht $\mathcal{L}_i$ ist – zusätzlich zum Zählen der Maximaschichten der Eingabe — zu realisieren, indem bei jedem neu gefundenen Ausläufer der Schicht $\mathcal{L}_i$ der Zähler c_i um eins inkrementiert wird. Da nur die obersten κ Schichten zu betrachten sind, wird der Zähler k für die Anzahl der Maxi-

[4] Der Punkt, mit dem p getauscht wird, kann von weiteren Betrachtungen ausgeschlossen werden, da $j \geq k$ gilt und er während des Überstreichens nicht erneut Ausläufer einer Schicht werden kann.

[5] Auch in den letzten beiden Fällen ist die Korrektheit des Algorithmus gesichert, da kein ehemaliger Ausläufer erneut zu betrachten ist und daher in den bereits abgearbeiteten Teil des Feldes verschoben werden kann.

maschichten nicht mehr inkrementiert, sobald er den Wert $k = \kappa$ erreicht hat. Für jeden im Folgenden überstrichenen Punkt kann in $\mathcal{O}(1)$ Zeit entschieden werden, ob er links von $\tau_{\kappa-1}$ liegt — und daher auf einer Schicht $\mathcal{L}_i$ mit $i > \kappa - 1$. In diesem Fall braucht der Punkt nicht weiter betrachtet zu werden, da er keine Inkrementierung eines der aufrecht zu haltenden Zähler induziert. Im anderen Fall kann der zu inkrementierende Zähler durch binäre Suche auf den Ausläufern der bereits betrachteten Schichten identifiziert werden. Es ergibt sich folgendes Resultat:

Lemma 6.3.3 *Die Kardinalitäten c_i der obersten κ Maximaschichten einer n-elementigen Menge zweidimensionaler Punkte können in insgesamt $\mathcal{O}(n \cdot \log_2 n)$ Zeit berechnet werden. Falls die Punkte bezüglich y vorsortiert sind, ergibt sich eine Laufzeitkomplexität von $\mathcal{O}(n + \xi \log_2 \kappa)$ mit $\xi = \sum_{i=0}^{\kappa-1} c_i$.*

6.3.3 Arrangieren von κ Schichten

Wie bereits erwähnt, ergibt sich aus der in Abschnitt 6.2.1 beschriebenen Berechnung der Maxima (und deren anschließender Entfernung aus der zu betrachtenden Punktmenge $\mathcal{P}$) ein Algorithmus, der iterativ alle Maximaschichten von $\mathcal{P}$ bestimmt. Dieses Verfahren weist allerdings eine Gesamtlaufzeit von $\mathcal{O}(n^2 \cdot \log_2 n)$ für Punktmengen $\mathcal{P}$ mit einer linearen Anzahl von Maximaschichten auf. Der folgende beschriebene Algorithmus orientiert sich an diesem iterativen Verfahren, berechnet jedoch in einer Iteration κ Schichten.

Extrahieren aller Punkte auf den obersten κ Schichten Der folgende Algorithmus extrahiert aus einer Punktmenge $\mathcal{P}$ genau die Punkte, die κ aufeinander folgende Schichten $\mathcal{L}_c, \ldots, \mathcal{L}_{c+\kappa-1}$ konstituieren. Da, wie erwähnt, für das Bestimmen aller Schichten mehrere Iterationen notwendig sind, sei erwähnt, dass im Allgemeinen Fall bereits einige (oberste) Schichten $\mathcal{L}_0, \ldots, \mathcal{L}_{c-1}$ der Eingabemenge $\mathcal{P}$ bestimmt und in $\mathtt{A}[0, \ldots, \ell_b - 1]$ arrangiert sind und nun die verbleibende Eingbe $\mathtt{A}[\ell_b, \ldots, n-1]$ zu betrachten bleibt. Für die Kürze der Darstellung seien die *nächsten* κ zu bestimmenden Schichten im Folgenden mit $\mathcal{L}_0, \ldots, \mathcal{L}_{\kappa-1}$ (statt mit $\mathcal{L}_c, \ldots, \mathcal{L}_{c+\kappa-1}$) referenziert.

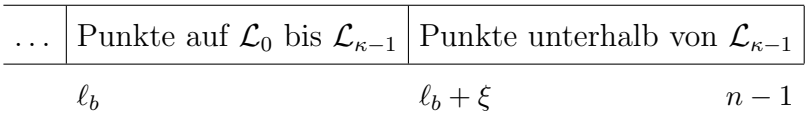
Induktiv wird über alle Iterationen folgende Invariante aufrecht erhalten, die initial durch Sortieren der Eingabe in $\mathtt{A}$ (und durch Festlegen von $\ell_b = 0$) hergestellt werden kann:

Sortier-Invariante Die Punkte in $\mathtt{A}[0, \ldots, n-1]$, die noch keiner Schicht zugeordnet sind, liegen absteigend nach ihrer y-Koordinate sortiert in $\mathtt{A}[\ell_b, \ldots, n-1]$ vor.

Es sei ferner mit ξ die Gesamtanzahl aller Punkte auf den nächsten obersten κ Schichten $\mathcal{L}_0$ bis $\mathcal{L}_{\kappa-1}$ der noch nicht zugeordneten, in $\mathtt{A}[\ell_b, \ldots, n-1]$ gespeicherten Punkte bezeichnet. Es sei diese Anzahl ξ zunächst als genügend klein angenommen, so dass $\ell_b + 2\xi \leq n$ gelte.

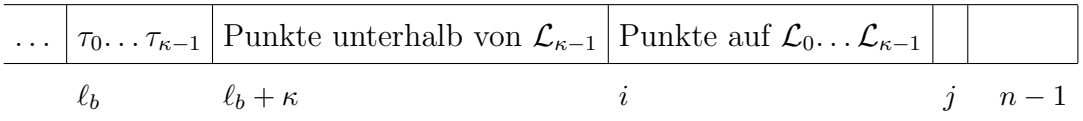
Im ersten Schritt des Algorithmus werden nun diese ξ Punkte der Schichten $\mathcal{L}_0$ bis $\mathcal{L}_{\kappa-1}$ extrahiert und nach $\mathtt{A}[\ell_b, \ldots, \ell_b + \xi - 1]$ verschoben, wobei die $<_y$-Ordnung

der weiteren Punkte, welche nach der Extraktion in $\mathtt{A}[\ell_b + \xi, \ldots, n-1]$ gespeichert sind, gewahrt bleibt:



Diese Separation wird wie folgt erreicht: Es wird eine Variante des Algorithmus aus Abschnitt 6.3.2 genutzt, die mit einer Abarbeitung des stabilen Teilmengenextraktionsalgorithmus von Bose *et al.* gekoppelt ist. Auch diese Variante hält die Ausläufer-Invariante (während des Überstreichens von $\mathtt{A}[\ell_b, \ldots, n-1]$) aufrecht, dass die jeweils aktuellen Ausläufer von τ_0 bis $\tau_{\kappa-1}$ in $\mathtt{A}[\ell_b, \ldots, \ell_b + \kappa - 1]$ gespeichert sind. Während der Abarbeitung werden zusätzlich alle Punkte, die auf einer Schicht *unterhalb* von $\mathcal{L}_{\kappa-1}$ liegen, in das bei $\mathtt{A}[\ell_b + \kappa]$ beginnende Teilfeld verschoben. Die so verschobenen Punkte werden sukzessive, beginnend bei $\mathtt{A}[\ell_b + \kappa]$ platziert, so dass sie in sortierter Reihenfolge verbleiben. Gleichzeitig werden Zähler c_i für die Anzahlen der Punkte auf den Schichten $\mathcal{L}_i$, $i = 0, \ldots, \kappa - 1$ aufrecht gehalten.[6] Für die Korrektheit des Algorithmus ist die Beobachtung wesentlich, dass keiner der in $\mathtt{A}[\ell_b, \ldots, \ell_b + \kappa - 1]$ *vor* dem Überstreichen befindlichen ersten κ Punkte unterhalb von $\mathcal{L}_{\kappa-1}$ liegen kann, da sich zumindest κ Punkte auf den obersten κ Schichten befinden.

Der Algorithmus hält offensichtlich die Invariante aufrecht, dass bei Überstreichen eines Punkte $\mathtt{A}[j]$ alle Punkte, die bereits als „unterhalb“ von $\mathcal{L}_{\kappa-1}$ liegend klassifiziert wurden, absteigend nach ihrer y-Koordinate sortiert in $\mathtt{A}[\ell_b + \kappa, \ldots, i-1]$ gespeichert sind.



Für einen jeden zu bearbeitenden, d. h. überstrichenen Punkt $\mathtt{A}[j]$ gilt, dass er entweder als „unterhalb“ von $\mathcal{L}_{\kappa-1}$ klassifiziert wird oder er den Ausläufer τ_h einer Schicht $\mathcal{L}_h$ ersetzt, wobei $h < \kappa$ gilt. Im ersteren Fall wird $\mathtt{A}[j]$ direkt hinter $\mathtt{A}[\ell_b + \kappa, \ldots, i-1]$ platziert, d. h. mit $\mathtt{A}[i]$ vertauscht, und i wird um eins inkrementiert. Im letzteren Falle hingegen wird $\mathtt{A}[j]$ mit dem zu ersetzenden Ausläufer τ_h vertauscht, also mit $\mathtt{A}[\ell_b + h]$.

Nach Beendigung des Überstreichens — und somit nach der Klassifizierung jedes Punktes in $\mathtt{A}[\ell_b, \ldots, n-1]$ hinsichtlich seiner Zugehörigkeit zu $\mathcal{L}_0 \cup \ldots \cup \ldots \mathcal{L}_{\kappa-1}$ — enthält $\mathtt{A}[\ell_b + \kappa, \ldots, i-1]$ in sortierter Reihenfolge genau die Punkte, die zu tieferen Schichten als $\mathcal{L}_\kappa$ gehören.

Des Weiteren ist gemäß Definition bzw. durch Berechnung von ξ die gemeinsame Anzahl an Punkten in den Teilfeldern $\mathtt{A}[\ell_b, \ldots, \ell_b + \kappa - 1]$ (in dem die Ausläufer enthalten sind) und $\mathtt{A}[i, \ldots, n-1]$ (in dem die die restlichen Punkte von $\mathcal{L}_0$ bis $\mathcal{L}_{\kappa-1}$ enthalten sind) exakt durch ξ vorgegeben. Nun können in linearer Zeit $\mathtt{A}[\ell_b +$

[6] Dies geschieht analog zu den Ausführungen in Abschnitt 6.3.2; die Erläuterung, wie diese Zähler *in-place* aufrecht gehalten werden können, folgt in Abschnitt 6.3.4.

$\kappa, \ldots, i-1]$ und $\mathtt{A}[i, \ldots, n-1]$ stabil vertauscht werden, so dass sich die ξ Elemente der Schichten $\mathcal{L}_0$ bis $\mathcal{L}_{\kappa-1}$ in $\mathtt{A}[\ell_b, \ldots, \ell_b + \xi - 1]$ befinden.

Für den folgenden Schritt des Algorithmus ist die Ordnung (nach absteigender y-Koordinate) auf diesen ξ Punkten wiederherzustellen. Dies ist in $\mathcal{O}(\xi \cdot \log_2 \xi) \subset \mathcal{O}(\xi \log n)$ Zeit möglich, so dass die Sortier-Invariante auf $\mathtt{A}[\ell_b, \ldots, \xi - 1]$ erneut etabliert ist. Die Gesamtlaufzeit für das Bestimmen von ξ, d. h. für das Zählen der Punkte auf den obersten κ Schichten, sowie für das Separieren dieser Punkte und die Wiederherstellung der Sortier-Invariante beläuft sich auf $\mathcal{O}(n + \xi \cdot \log_2 n)$ mit $\xi = \sum_{i=0}^{\kappa-1} c_i$.

Gruppieren von Punkten nach ihren Schichten Der zuletzt geschilderte Algorithmus hielt Zähler c_i für die jeweiligen Anzahlen der Punkte auf den Schichten $\mathcal{L}_0$ bis $\mathcal{L}_{\kappa-1}$ aufrecht. An Hand der finalen Zählerwerte lässt sich nun ein *counting-sort* [CLR01] ähnlicher Algorithmus verwenden, um die insgesamt $\xi = \sum_{i=0}^{\kappa-1} c_i$ im letzten Teilschritt separierten Punkte ihren zugehörigen Schichten nach zu gruppieren. Für jede dieser Schichten wird entsprechend ihrer Mächtigkeit ein Teilfeld in der Eingabe reserviert, in das die ihr zugehörigen Punkte zu platzieren sind. Die Positionen dieser Teilfelder ergeben sich aus den Zählerwerten c_j durch Berechnen der Präfixsummen $\hat{c}_j := \sum_{i=0}^{j} c_i$. Um diese Teilfelder *in-place* füllen zu können, ohne die Struktur wesentlicher Teile der Eingabe zu verletzen, wird das Teilfeld $\mathtt{A}[\ell_b + \xi, \ldots, \ell_b + 2\xi - 1]$ als Zwischenspeicher für die einzelnen Schichten verwendet. Die Notwendigkeit dieses Zwischenspeichers ist der Grund für die oben getroffene Annahme $\ell_b + 2\xi \leq n$.

Auch im nachfolgend beschriebenen Teilschritt des Algorithmus wird ein Eingabeteil — hier das Teilfeld $\mathtt{A}[\ell_b, \ldots, \ell_b + \xi - 1]$ — überstrichen, wiederum gemäß der (wiederhergestellten) $<_y$-Sortierung. Als Sweep-Struktur werden erneut die Ausläufer der insgesamt κ Schichten aufrecht erhalten. Während des Überstreichens wird ein jeder Ausläufer τ_h, wenn er durch einen gerade überstrichenen Punkt zu ersetzen ist, in genau das Teilfeld von $\mathtt{A}[\ell_b + \xi, \ldots, \ell_b + 2\xi - 1]$ verschoben, das für die Punkte der Schicht $\mathcal{L}_h$ reserviert ist. Dieses Teilfeld wird von links nach rechts mit Punkten gefüllt, d. h. der ersetzte Ausläufer wird direkt rechts der Punkte, die der Schicht $\mathcal{L}_h$ bereits zugeordnet wurden, ergänzt. Folglich sind die einer einzelnen Schicht zugeordneten Punkte nach ihrer y-Koordinate sortiert. Das Eingabefeld besitzt somit nach Ablauf des Überstreichens folgende Struktur:

$\ldots$	τ_0	$\ldots$	$\tau_{\kappa-1}$		$\mathcal{L}_0$	$\ldots$	$\mathcal{L}_{\kappa-1}$	$\ldots$
	ℓ_b				$\ell_b + \xi$			$\ell_b + 2\xi$

Nachdem die $<_y$-sortierten Repräsentationen der Schichten $\mathcal{L}_0$ bis $\mathcal{L}_{\kappa-1}$ komplett vorliegen, werden nun die beiden Teilfelder $\mathtt{A}[\ell_b, \ldots, \ell_b + \xi - 1]$ und $\mathtt{A}[\ell_b + \xi, \ldots, \ell_b + 2\xi - 1]$ *in-place* vertauscht, so dass die Eingabe sich wie folgt darstellt:

$\ldots$	$\mathcal{L}_0$	$\ldots$	$\mathcal{L}_{\kappa-1}$	Punkte unterhalb von $\mathcal{L}_{\kappa-1}$	$\ldots$
	ℓ_b			$\ell_b + \xi$	$\ell_b + 2\xi$

Abschließend sind die Punkte in $\mathtt{A}[\ell_b + \xi, \ldots, \ell_b + 2\xi - 1]$ absteigend bezüglich ihrer y-Koordinate zu sortieren, um die Sortier-Invariante wiederherzustellen; die Punkte in $\mathtt{A}[\ell_b + 2\xi, \ldots, n-1]$ werden während des Überstreichens nicht verschoben, so dass die Sortier-Invariante auf diesem Teil der Eingabe erhalten bleibt.

Die Laufzeit für das Gruppieren der ξ Punkte der κ Schichten beläuft sich auf $\mathcal{O}(n + \xi \cdot \log_2 n)$. Dies beinhaltet zum einen die Laufzeit für das Überstreichen und Einordnen einzelner Punkte von insgesamt $\mathcal{O}(\xi \cdot \log_2 \kappa) \subseteq \mathcal{O}(\xi \cdot \log_2 n)$ als auch die Kosten von $\mathcal{O}(\xi) \subseteq \mathcal{O}(n)$ für das Vertauschen der Blöcke $\mathtt{A}[\ell_b, \ldots, \ell_b + \xi - 1]$ und $\mathtt{A}[\ell_b + \xi, \ldots, \ell_b + 2\xi - 1]$ und das Wiederherstellen der Sortier-Invariante für $\mathtt{A}[\ell_b + \xi, \ldots, \ell_b + 2\xi - 1]$.

6.3.4 *in-place*-Arrangieren aller Schichten

Die obige Beschreibung von Teilen des Algorithmus beruhte auf zwei wesentlichen Annahmen:

(1) Der Algorithmus verfügt über genügend Speichermöglichkeiten, um κ Zähler zu pflegen und auf diese zuzugreifen.

(2) Das Teilfeld $\mathtt{A}[\ell_b, \ldots, n-1]$ der noch nach Schichten zu ordnenden Punkte ist in jeder Iteration des Algorithmus groß genug, um zwei Teilfelder der Größe ξ zu enthalten.

In diesem Kapitel wird ausgeführt, wie diese beiden Annahmen erfüllt werden können, ohne mehr als konstanten zusätzlichen Speicher zu verwenden. Das erste Problem hierbei besteht darin, eine nicht-konstante Anzahl κ von Zählern vorzuhalten, ohne dafür $\Theta(\kappa)$ zusätzlichen Speicher zu verwenden. Da ein jeder Zähler Werte bis zu der Größe von n zu speichern hat, bedarf es zur Speicherung aller Zähler insgesamt $\kappa \cdot \log_2 n$ Bits. Hierfür lässt sich erneut die in Kapitel 4.1.1 beschriebene und bereits in Kapitel 5 verwendete Kodierungstechnik von Munro [Mun86] adaptieren. Die Punkte der Eingabe sind linear anordbar, beispielsweise durch die lexikographische $<_y$-Ordnung. Zudem wird sich als hilfreich erweisen, dass auf Teilen der Eingabe diese Ordnung aufrecht erhalten wird (Sortier-Invariante).

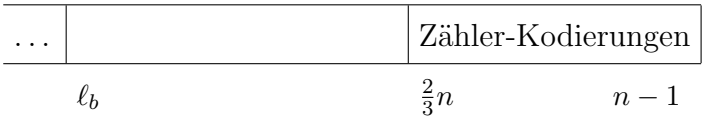
Bemerkung 6.3.4 Ist die Eingabe duplikatfrei,so ist sie sogar streng linear anordbar. Für eine Eingabe mit Duplikaten, also für eine Multimenge, können — ohne die Komplexität der Gesamtlaufzeit zu erhöhen — zu Beginn des Algorithmus alle Duplikate entfernt bzw. vom Rest der Eingabe separiert werden. Dies ist eine akzeptable Vorgehensweise, da (im Gegensatz zu dem in Kapitel 5 behandelten Problem des geometrischen Schätzens) die geometrische Beschaffenheit der Ausgabe (also der Maximaschichten) unabhängig davon ist, ob Duplikate in die Betrachtung mit einbezogen werden oder nicht.

Ein zur Kodierung von der beschriebenen Form reservierter Block von $\frac{1}{3}n$ Eingabeelementen genügt, um $\frac{1}{6}n$ Bits, d. h. $\frac{1}{6}n/\log_2 n$ Zähler vorzuhalten, so dass letztere Werte von 0 bis n darstellen können. Diese Größe impliziert eine Obergrenze von $\kappa = \frac{1}{6}n/\log_2 n$ für die Anzahl der Schichten, die in einer Iteration durch die Teilalgorithmen aus den Abschnitten 6.3.2 und 6.3.3 bearbeitet bzw. bestimmt werden

können. Die Laufzeitanalysen zum Abschluss dieses Abschnittes ergeben Kosten von jeweils $\mathcal{O}(n + \xi \cdot \log_2 n)$ für solche Iterationen. Da jede dieser Iterationen höchstens $\mathcal{O}(n/\log_2 n)$ Schichten bestimmen kann, sind zumindest $\mathcal{O}(\log_2 n)$ solcher Iterationen notwendig. Diese Anzahl ist noch akzeptabel, da sie eine optimale Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ nicht ausschließt. Um dieses Ziel unter Verwendung nur konstanten Zusatzspeichers zu erreichen, ist allerdings sicherzustellen, dass aus dem Vorhalten und Auswerten von *kodierten* Zählern keine die Komplexität erhöhenden Laufzeitkosten und keine zusätzlich durchzuführenden Iterationen resultieren.

Der Fall $\ell_b < \frac{1}{3}n$: die ersten Iterationen

Falls vor der aktuell abzuarbeitenden Iteration $\ell_b < \frac{1}{3}n$ gilt, also noch kein Drittel der Eingabepunkte in Schichten gruppiert wurde, werden die kodierten Zähler c_i im hinteren Teilfeld $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$ vorgehalten.



Zählen der Punkte auf den obersten κ Schichten Entsprechend der Sortier-Invariante sind die Elemente in $\mathtt{A}[\ell_b, \ldots, n-1]$ absteigend bezüglich ihrer y-Koordinate sortiert und somit kodieren alle Zähler initial den Wert 0. Nun wird der Algorithmus aus Abschnitt 6.3.2 zum Zählen der Punkte auf jeder der obersten κ Schichten abgearbeitet, wobei $\kappa := \frac{1}{6}n/\log_2 n$ zu wählen ist.

Jeder der κ Zähler c_i wird während des Zählvorgangs aktualisiert, indem Bit-Nachbarn vertauscht werden. Die Standardanalyse für das schrittweise Inkrementieren von Zählern liefert eine Gesamtanzahl an Bit-Operationen bzw. Vertauschungen von Eingabeelementen während des Überstreichens von $\mathcal{O}(\xi)$ mit $\xi = \sum_{i=0}^{\kappa-1} c_i$.

Es ist zu beachten, dass während der Abarbeitung des Zählalgorithmus *alle* Punkte in $\mathtt{A}[\ell_b, \ldots, n-1]$ überstrichen werden und damit auch jeder Punkt q, der in $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$ zu Kodierung verwendet wird. Dies ist insofern problematisch, da ein solcher Punkt an den Anfang des Eingabefeldes getauscht werden muss, sofern er als Element (und damit zunächst als Ausläufer) einer der obersten κ Schichten identifiziert wurde. Dennoch können während des Extrahierens von $min(\kappa, \kappa')$ Schichten (mit noch zu beschreibendem κ') in nur $\mathcal{O}(n+\xi\cdot\log_2 n)$ Zeit zusätzlich alle kodierten Zähler korrekt aufrecht erhalten und die Sortier-Invariante wiederhergestellt werden:

Lemma 6.3.5 *Im Falle $\ell_b < \frac{1}{3}n$ können die jeweiligen Mächtigkeiten der obersten $min(\kappa, \kappa')$ Schichten* in-place *und in $\mathcal{O}(n + \xi \cdot \log_2 n)$ Zeit berechnet werden. Für $\kappa = \kappa'$ schließt dies die Wiederherstellung der Sortier-Invariante mit ein; andernfalls sind die Wiederherstellungskosten in $\mathcal{O}(n \cdot \log_2 n)$.*

Der Nachweis dieser Aussage wird konstruktiv im folgenden Abschnitt erbracht, zusammen mit einem analogen Nachweis betreffend die Extraktion der Punkte der κ obersten Schichten.

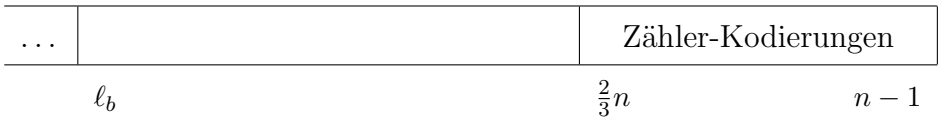
Nach dem Zählvorgang, jedoch vor der Wiederherstellung der Sortier-Invariante, werden aus den finalen Werten der Zähler c_0 bis $c_{\kappa-1}$ die Präfixsummen berechnet,

d. h. der kodierte Zählerwert c_j wird durch $\hat{c}_j := \sum_{i=0}^{j} c_i$ ersetzt. Dies kann *in-place* und in $\mathcal{O}(\log_2 n)$ Zeit pro Zähler erfolgen, also in insgesamt $\mathcal{O}(\kappa \cdot \log_2 n) = \mathcal{O}(n)$ Zeit. Während der Präfixsummenberechnung wird zusätzlich der maximale Index κ' bestimmt, so dass $\ell_b + 2\hat{c}_{\kappa'} < \frac{2}{3}n$ gilt. Dieser Index, sofern existent, spezifiziert — anstelle des Indexes κ — die Anzahl der in der aktuellen Iteration tatsächlich abzuarbeitenden Schichten.

Extrahieren der Punkte auf den κ Schichten Falls der beschriebene Index κ' existiert, wird ein leicht modifizierter Algorithmus zur Extraktion der $\xi' := \hat{c}_{\kappa'}$ Punkte auf den κ' obersten Schichten angewendet, vergleiche Abschnitt 6.3.3. Auf Grund der Wahl von κ' kann hierbei die Annahme (2), auf der dieser Algorithmus beruhte, auch ohne Verwendung von mehr als konstantem Zusatzspeicher erfüllt werden: Ein Zwischenspeicher der Größe $\hat{c}_{\kappa'}$ zur Vorhaltung aller zu extrahierenden Punkte ist innerhalb des Eingabefeldes verfügbar, ohne dass dieser Speicher das Teilfeld $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$, welches für die Kodierung der Zähler c_i reserviert wurde, mitverwendet. Des Weiteren ist durch die Sortier-Invariante gesichert, dass die Elemente in $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$ vor dem Zählvorgang absteigend bezüglich ihrer y-Koordinate sortiert sind. Diese beiden Sachverhalte garantieren, dass nach der Extraktion aller ξ' Punkte die Sortierinvariante in $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$ in $\mathcal{O}(n)$ Zeit wiederhergestellt werden kann, indem alle Zählerwerte auf 0 gesetzt werden, d. h. allein durch Vertauschung von Bit-Nachbarn.

Die sweep-Algorithmen zum Zählen der Punkte auf den obersten κ Schichten sowie zum Extrahieren der entsprechenden Punkte verkomplizieren sich dadurch, dass zu den überstrichenen Punkten auch jene gehören können, die die κ Zähler kodieren. Im nachfolgenden Abschnitt wird diese Problematik näher erläutert und aufgelöst.

Bewahren von Zähler-Kodierungen während des Extrahierens Im Falle $\ell_b < \frac{1}{3}n$ ist die Eingabe wie nachfolgend skizziert strukturiert:



Der Sortier-Invariante entsprechend ist $\mathtt{A}[\ell_b, \ldots, n-1]$ vor dem Extrahieren der Punkte der obersten der obersten $min(\kappa, \kappa')$ Schichten absteigend bezüglich $<_y$ sortiert; allerdings ist diese Sortierung durch die dort kodierten Zähler in $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$ lokal permutiert, d. h. einzelne Punkte sind eventuell mit ihren Bit-Nachbarn vertauscht worden. Im Folgenden sei in diesem Abschnitt der Einfachheit der Notation halber von κ statt von $min(\kappa, \kappa')$ als der Anzahl der zu extrahierenden Schichten gesprochen. Es gilt nun den Algorithmus zu modifizieren, der in Abschnitt 6.3.3 für zusätzlich verfügbaren Speicher von $\mathcal{O}(\kappa)$ Wörtern (für die externe Vorhaltung der Zähler c_i) beschrieben wurde. Dieser Algorithmus hält als Invariante aufrecht, dass die Ausläufer τ_0 bis $\tau_{\kappa-1}$ entsprechend der Ordnung der Schichten im Teilfeld $\mathtt{A}[\ell_b, \ldots, \ell_b + \kappa - 1]$ gespeichert sind. Zusätzlich verschiebt er alle $\nu := n - \ell_b - \xi$ Punkte, die sich „unterhalb“ der Schicht $\mathcal{L}_{\kappa-1}$ befinden, in das Teilfeld $\mathtt{A}[\ell_b+\kappa, \ldots, \ell_b+\kappa+\nu-1]$. Für die Kürze der Darstellung bezeichnen wir letztere

Punkte fortan als OUT-Punkte, da sie in der aktuellen Iteration nicht gezählt und auch nicht gruppiert werden. Dementsprechend werden die Punkte auf den Schichten $\mathcal{L}_0$ bis $\mathcal{L}_{\kappa-1}$ als IN-Punkte bezeichnet. Während die OUT-Punkte stabil durch die Teilmengenextraktion von Bose *et al.* [BMM+06] selektiert werden können und also (absteigend bezüglich $<_y$) sortiert bleiben, kann dies für die IN-Punkte nicht garantiert werden.

Nachfolgend wird induktiv aufgezeigt, wie während des Extrahierens bzw. Separierens von IN- und OUT-Punkten die Zähler-Kodierungen und die Sortierungen innerhalb des OUT-Blockes *in-place* und durch nur linearen zusätzlichen Zeitaufwand aufrecht erhalten werden können. Mit obigen Bezeichnungen und der Induktionsannahme ergibt sich unmittelbar vor Abarbeitung eines Punktes $\mathtt{A}[j]$ folgendes Bild des Eingabefeldes:

...	OUT (sortiert)	IN			Zähler-Kodierungen
	$\ell_b + \kappa$	i	j	$\frac{2}{3}n$	$n-1$

Der Punkt $\mathtt{A}[j]$ wird nun entweder als OUT-Punkt klassifiziert oder er ersetzt den bisherigen Ausläufer τ_h einer Schicht $\mathcal{L}_h$. Im ersteren Fall wird $\mathtt{A}[j]$ mit dem Nachfolgeelement $\mathtt{A}[i]$ des OUT-Blockes getauscht und i als der Index, der das Ende dieses Teilfeldes anzeigt, um eins erhöht. Im letzteren Fall wird $\mathtt{A}[j]$ mit τ_h, d. h. mit $\mathtt{A}[\ell_b + h]$, vertauscht.[7]

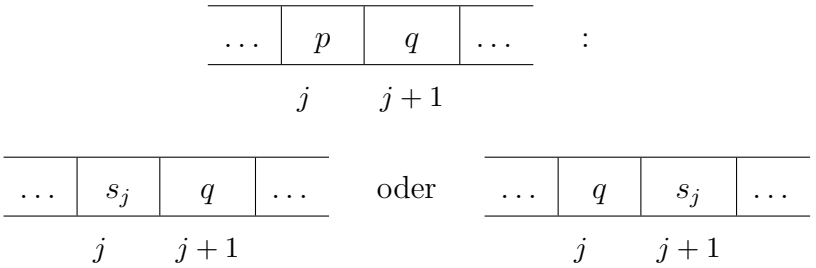
Die wesentliche Beobachtung, die die Aufrechterhaltung der in $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$ kodierten Zählerwerte trotz möglicher Vertauschungen der kodierenden Elemente erlaubt, ist, dass sich die relative Ordnung der kodierenden Elemente durch die Inkrementierungen der Zähler nur lokal ändert. Konkret kann ein kodierendes Element nicht mehr als eine Position von seinem korrekten Platz in der $<_y$-Ordnung abweichen — solange der Zeiger j außerhalb von $[\frac{2}{3}n, \ldots, n-1]$ liegt.

Sobald der Zeiger j den Wert $\frac{2}{3}n$ übersteigt, besteht die Gefahr, dass das Vertauschen überstrichener Punkte die kodierten Zähler korrumpiert. Daher werden ab diesem Zeitpunkt die Indizes der Zähler vorgehalten, in die der Index j bzw. der Index i zeigen. Es ist zu beachten, dass innerhalb eines kodierten Zählers jeweils zwei Bit-Nachbarn vertauscht sein können. Daher ist bei Überstreichen eines Punktes $\mathtt{A}[j]$ auch sein Bit-Nachbar, d. h. $\mathtt{A}[j-1]$ oder $\mathtt{A}[j+1]$, mitzubetrachten. Die Identifikation des tatsächlich als nächstes zu überstreichenden Punktes erfordert die lokale Rekonstruktion der y-Ordnung der beiden Bit-Nachbarn. Diese Überprüfung verursacht, wie bereits ausgeführt, nur einen konstanten zusätzlichen Aufwand hinsichtlich Laufzeit und Speicherplatz.

Es sei zunächst angenommen, dass der eigentlich zu überstreichende Punkt $p := \mathtt{A}[j]$ und sein Bit-Nachbar $\mathtt{A}[j+1]$ sei. Der überstrichene Punkt p ist nun zu vertauschen — entweder mit $\mathtt{A}[i]$ oder mit einem abzulösenden Ausläufer τ_h. Dasjenige Element, was nach der Abarbeitung von p an der ehemaligen Position von p platziert ist, sei mit s_j bezeichnet. Durch dieses Vertauschen von p und s_j ändert sich eventuell die relative $<_y$-Ordnung von $\mathtt{A}[j]$ und seinem Bit-Nachbarn $\mathtt{A}[j+1]$. Um

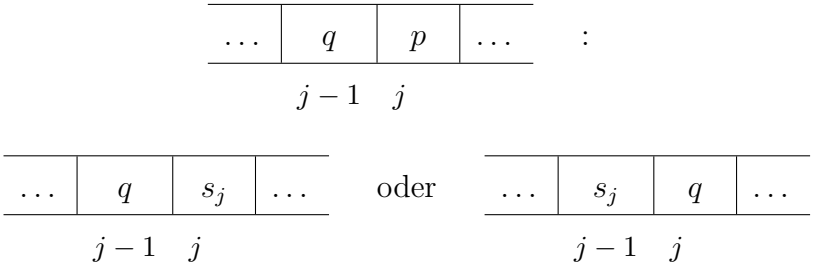
[7] Die Behandlung des letzten Falles ist auch zu vollziehen, wenn $\mathtt{A}[j]$ Ausläufer einer neuen Schicht ist. Der Punkt, der mit $\mathtt{A}[j]$ seine Position tauscht, ist bereits während der Überstreichung betrachtet worden. Seine Umpositionierung ist daher unproblematisch.

die für die Zählerkodierung notwendige Ordnung wiederherzustellen, sind eventuell $\mathtt{A}[j+1]$ und s_j zu vertauschen. Die aus diesem Fall resultierende Situation ist in der folgenden Skizze unten rechts abgebildet:

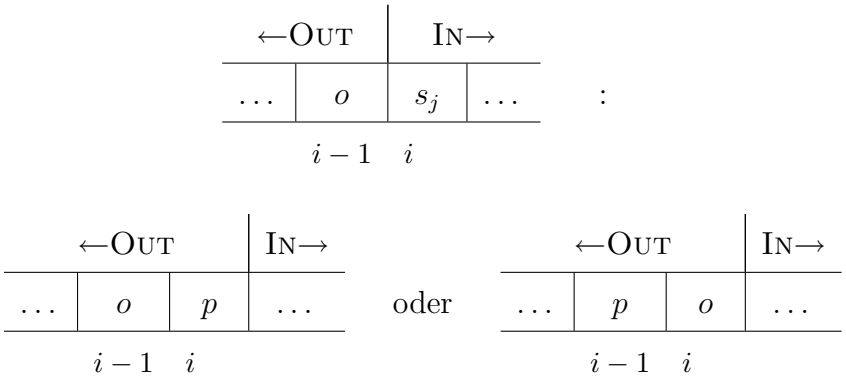


Die Vertauschung von $\mathtt{A}[j+1]$ und s_j betrifft das nächste zu überstreichende Element $\mathtt{A}[j+1]$. Die Information über dessen veränderte Position lässt sich jedoch in konstantem Speicher bis nach seiner Abarbeitung vorhalten.

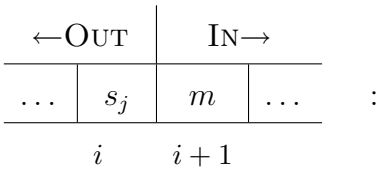
Folgend sei angenommen, dass $\mathtt{A}[j-1]$ und nicht $\mathtt{A}[j+1]$ der Bit-Nachbar des zu überstreichenden Punktes $\mathtt{A}[j]$ ist. In diesem Fall tritt die letztgenannte Problematik nicht auf, da das Vertauschen von $\mathtt{A}[j]$ mit s_j den nächsten zu übersteichenden Punkt (d. h. $\mathtt{A}[j+1]$ oder seinen Bit-Nachbarn $\mathtt{A}[j+2]$) nicht betrifft:



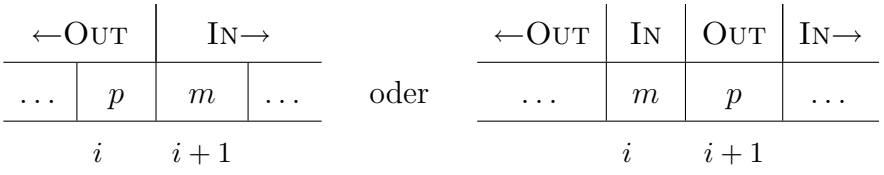
In ähnlicher Weise ist der Zähler zu behandeln, in dem der Punkt $\mathtt{A}[i]$ residiert, also das erste Element des In-Blocks. Wann immer der aktuell zu bearbeitende Punkt $p := \mathtt{A}[j]$ als Out-Punkt klassifiziert wird, ist er anschließend mit $s_j := \mathtt{A}[i]$ zu vertauschen. Diese Vertauschung wiederum kann den Zähler, zu dessen Kodierung $\mathtt{A}[i]$ diente, korrumpieren und in diesem Falle muss $\mathtt{A}[i]$ mit seinem Bit-Nachbarn vertauscht werden. Falls der Bit-Nachbar von $\mathtt{A}[i]$ sein Vorgänger $\mathtt{A}[i-1]$ ist, so ist dieser Bit-Nachbar ein Out-Punkt $o := \mathtt{A}[i-1]$ und die Vertauschung von $\mathtt{A}[i]$ und $\mathtt{A}[i-1]$ ändert zwar deren relative Ordnung, jedoch nicht den Index, der die Trennung von In- und Out-Block anzeigt.



Falls andererseits der Bit-Nachbar von $\mathtt{A}[i]$ sein Nachfolger $m := \mathtt{A}[i+1]$ ist, so führt die Vertauschung der Bit-Nachbarn zu einer „unscharfen“ Trennung zwischen In- und Out-Block, wie in der folgenden Skizze unten rechts dargestellt.

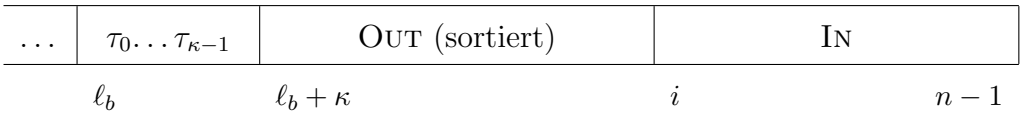


:



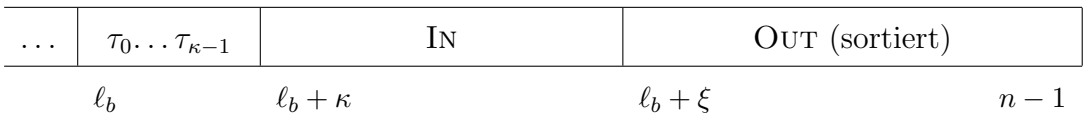
Im diesem Fall hat der nächste Punkt p', der als Out-Punkt klassifiziert wird, m zu ersetzen. Dadurch entsteht wieder eine eindeutige Trennung zwischen In- und Out-Block. Die nach einer Ersetzung von m eventuell notwendige Reparatur des Zählers, den m mit kodierte, ist wie oben ausgeführt durchzuführen und beeinflusst die wiederhergestellte Trennung der Blöcke nicht. Die Information über den temporären Sonderfall der unscharfen Trennung der beiden Blöcke kann wiederum mit konstantem Aufwand an Zeit und Speicherplatz vorgehalten werden. Dies erlaubt insbesondere auch, die klare Trennung nach Beendigen des Übersteichens wieder herzustellen, falls nach p kein weiterer Punkt als Out-Punkt klassifiziert wird.

Per Induktion ist somit nachgewiesen, dass die Extraktion der Punkte auf den κ Schichten nach Beenden des Überstreichens erfolgreich abgeschlossen ist, die Zähler ihre ursprünglich kodierten Werte darstellen und die folgende Struktur des Eingabefeldes resultiert:



Es ist noch die Sortier-Invariante, separat für In- und Out-Block, zu betrachten: Deren Wiederherstellung zum Zwecke der weiteren Gruppierung von Punkten in Schichten kann in der gewünschten Zeitkomplexität von $\mathcal{O}(n + \xi \cdot \log_2 n)$ geleistet werden: Hierfür ist zu beachten, dass die Sortierung aller $\Theta(\xi)$ In-Punkte durch heapsort nicht mehr als $\mathcal{O}(\xi \cdot \log_2 n)$ Zeit erfordert und die Out-Punkte nur lokal durch die (weiterhin vorzuhaltenden) Zähler permutiert sind. Letzteres ist darin begründet, dass alle als Out-Punkte klassifizierten Elemente stabil in den Out-Block verschoben werden. Da die Kodierung der Zähler die Elemente nur um maximal eine Position von ihrer bezüglich der absteigenden y-Ordnung korrekten Position verschiebt, ist diese Ordnung durch einen linearen Durchlauf wiederherzustellen.

Eine anschließende Vertauschung von Out-Block und IN-Block resultiert in folgender Gestalt des Eingabefeldes:



Die Vereinigung der Ausläufer in $\mathtt{A}[\ell_b, \ldots, \ell_b+\kappa-1]$ mit den ehemaligen Ausläufern der κ obersten Schichten enthält genau ξ Punkte (nach der Definition von ξ). Somit ist eine Sortierung des Teilfeldes $\mathtt{A}[\ell_b, \ldots, \ell_b + \xi - 1]$ in $\mathcal{O}(\xi \cdot \log_2 \xi) \subset \mathcal{O}(\xi \cdot \log_2 n)$ Zeit durchführbar und das Eingabefeld stellt sich anschließend wie folgt dar:

Die Extraktion der Punke, die die κ obersten Schichten konstituieren, ist damit abgeschlossen und die Ordnung sowohl auf den nun zu gruppierenden ξ Punkten als auch auf den in weiteren Iterationen zu betrachtenden Punkten (gemäß der Sortier-Invariante) hergestellt und wir erhalten das gewünschte Resultat:

Lemma 6.3.6 *Im Falle $\ell_b < \frac{1}{3}n$ können alle Punkte der obersten $min(\kappa, \kappa')$ Schichten* in-place *und in $\mathcal{O}(n + \xi \cdot \log_2 n)$ Zeit extrahiert werden, was die Kosten für die Wiederherstellung der Sortier-Invariante miteinschließt.*

Aufrechterhalten von Zähler-Kodierungen während des Zählens Obige Ausführungen und eine leicht modifizierte Form des vorgestellten Algorithmus liefern auch den Nachweis von Lemma 6.3.5. Auch während des — dem Extrahieren vorangehenden — Zählschrittes gilt es, durch ein Überstreichen einzelne Punkte ihren Schichten zuzuordnen und dabei kodierte Zähler konsistent vorzuhalten.

Die einzige zusätzliche Schwierigkeit, die sich während des Zählschrittes ergibt, ist, dass während des Überstreichens Zählerwerte nicht nur bewahrt, sondern auch aktualisiert werden müssen. Dies lässt sich jedoch einfach auf die im vergangenen Abschnitt behandelten Fälle zurückführen, indem generell *vor* der Verschiebung eines überstrichenen Punktes $\mathtt{A}[j] = \tau_h$ die Aktualisierung des Zeigers c_h vollzogen wird. Der Sonderfall, dass diese Aktualisierung des Zählers die Position des zu verschiebenden Punktes $\mathtt{A}[j] = \tau_h$ selbst verändert (also ihn mit seinem Bitnachbarn vertauscht), kann mit konstantem Zeit- und Speicheraufwand behandelt werden, da nur eine konstante Anzahl an Eingabeelementen involviert ist.

Zur Wiederherstellung der Sortier-Invariante ist nach dem Zählvorgang zunächst (analog zum Extraktionsschritt) der In-Block zu sortieren. Ist während des Zählvorgangs ein Wert κ' bestimmt worden, impliziert dies, dass der Extraktionsschritt nicht den kompletten In-Block, der während der Zählphase bestimmt wurde, zu extrahieren hat (da eine anschließende Gruppierung der Elemente dieses Blockes nicht genügend Platz für den notwendigen Zwischenspeicher ließe). Nur in diesem Fall ist eine nachfolgende Extraktionsphase notwendig. Für diese brauchen allerdings nur Punkte des In-Blockes betrachtet zu werden, da dieser eine Obermenge aller Punkte der obersten κ' Schichten enthält.[8]

Insgesamt ergibt sich somit die Aussage von Lemma 6.3.5. Die erhöhte Laufzeit für den Fall $\kappa' < \kappa$ (d. h. für den Fall, dass nicht die Mächtigkeiten aller

[8]Entsprechend existieren nach Beendigung der zweiten Extraktion zwei Out-Blöcke, die nach Ablauf der Iteration (in linearer Zeit, beispielsweise mit dem Algorithmus von Geffert *et al.* [GKP00]) zu verschmelzen sind, um die Sortier-Invariante wiederherzustellen.

$\kappa = \frac{1}{6}n/\log_2 n$ Schichten gezählt wurden) ergibt sich daraus, dass in diesem Fall mehr als ξ Punkte als IN-Punkte klassifiziert wurden. Dementsprechend fallen für die Sortierung des IN-Blockes erhöhte Laufzeitkosten an. Diese erhöhten Kosten sind jedoch akzeptabel, da im Falle $\kappa' < \kappa$ in der aktuellen Iteration $\mathcal{O}(n)$ Punkte ihren Schichten entsprechend gruppiert werden können, vergleiche die Analyse zum Abschluss dieses Abschnitts.

Arrangieren der Punkte auf den κ Schichten Für die Gruppierung von Punkten in die zu Schichten $\mathcal{L}_i$ korrespondierenden Teilfelder $\mathtt{A}_i$ ist für jedes dieser Teilfelder die Position vorzuhalten, an die der als nächstes gefundene Punkt dieser Schicht zu setzen ist. Diese Einfügepositionen werden durch die aus den Zählerwerten c_i berechneten Präfixsummen initialisiert, d. h. die in $\mathtt{A}[\frac{2}{3}n, \ldots, n-1]$ kodierten Zählerwerte werden wie folgt ersetzt und anschließend sukzessive inkrementiert: Setze $c_0 := 0$ und $c_j := \hat{c}_{j-1}$ für $0 < j \leq \kappa'$.

Wann immer ein neuer Ausläufer τ_i gefunden wird, ist der bisherige Ausläufer p in $\mathtt{A}[i]$ mit $\mathtt{A}[\ell_b + \xi' + c_i]$ zu vertauschen und der kodierte Index c_i um eins zu inkrementieren. Die Kosten für das Dekodieren und Inkrementieren von c_i können durch $\mathcal{O}(\log_2 n)$ abgeschätzt und dem Punkt p, der hiermit in das Teilfeld seiner Schicht verschoben wurde, zugeschlagen werden.

Folglich sind die globalen Kosten für das Arrangieren (d. h. Gruppieren nach Schichten) von ξ' Punkten in $\mathcal{O}(n + \xi' \log_2 n)$; dies beinhaltet auch die Kosten für das Resortieren des Zwischenspeichers $\mathtt{A}[\ell_b + \xi', \ldots, \ell_b + 2\xi' - 1]$ und das weitere Wiederherstellen der Sortierinvariante durch das Verschmelzen von $\mathtt{A}[\ell_b + \xi', \ldots, \ell_b + 2\xi' - 1]$ mit dem sortierten Teilfeld $\mathtt{A}[\ell_b + 2\xi', \ldots, n-1]$, vergleiche Abschnitt 6.3.3.

Liegt der Fall $\kappa' < \kappa$ vor, d. h. werden in der aktuellen Iteration einige, aber nicht alle der $\kappa = \frac{1}{6}n/\log_2 n$ Schichten arrangiert, so wird — als Nachbearbeitung der Iteration — zusätzlich der Algorithmus aus Abschnitt 6.2.1 zur Extraktion der Skyline der nächstfolgenden Schicht $\mathcal{L}_{\kappa'}$ (auf die Eingabe in $\mathtt{A}[\ell_b, \ldots, n-1]$) angewendet. Falls hingegen kein Index κ' vorliegt, so enthält die oberste Schicht $\mathcal{L}_0$ eventuell mehr Punkte, als im Zwischenspeicher sicher arrangiert werden können; genauer gilt $c_0 > \frac{1}{2}\left(\frac{2}{3}n - \ell_b\right) > \frac{1}{2}\left(\frac{2}{3}n - \frac{1}{3}n\right) \in \Theta(n)$. In diesem Fall wird in der anstehenden Iteration allein diese oberste Schicht extrahiert — wiederum mit dem Algorithmus aus Kapitel 6.2.1. Um nach erfolgter Iteration die Sortier-Invariante wiederherzustellen, ist in beiden Fällen nicht mehr als $\mathcal{O}(n \cdot \log_2 n)$ Zeit aufzuwenden.

Analyse Die folgende Analyse unterscheidet Iterationen danach, ob die ξ Punkte *aller* $\kappa = \frac{1}{6}n/\log_2 n$ obersten Schichten in den zugehörigen Teilfeldern arrangiert werden. In diesem Fall gilt $\xi \geq \frac{1}{6}n/\log_2 n$ and daher können nur logarithmisch viele solcher Iterationen durchgeführt werden. Der Zeitaufwand für eine solche Iteration ist jeweils in $\mathcal{O}(n+\xi\cdot\log_2 n)$. Jeder Iteration sei der lineare Anteil $\mathcal{O}(n)$ zugeschlagen; die restlichen Kosten seien zu je $\mathcal{O}(\log_2 n)$ den ξ in dieser Iteration arrangierten Punkten zugeordnet.

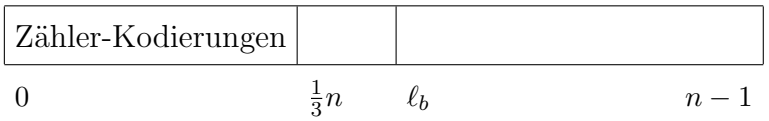
Wenn hingegen in einer Iteration weniger als κ Schichten arrangiert werden (was den Fall beinhaltet, dass κ' nicht existiert), so sind die Kosten der Iteration durch den Aufwand von $\mathcal{O}(n \cdot \log_2 n)$ für die Skyline-Berechnung der nächstfolgenden Schicht und die Wiederherstellung der Sortier-Invariante dominiert: Dies gilt sowohl für die

$\mathcal{O}(n + \xi \cdot \log_2 n)$ Kosten für das Zählen von ξ Punkten auf den obersten κ Schichten als auch für die Kosten von $\mathcal{O}(n + \xi' \log_2 n)$ für die anschließende Arrangierung der ξ' Punkte auf den obersten κ' Schichten.

Die Definition von κ' garantiert, dass in jeder Iteration letzteren Typs (nach der abschließenden Arrangierung einer einzelnen Skyline) mindestens $\frac{1}{2}\left(\frac{2}{3}n - \ell_b\right)$ Punkte entsprechend ihrer Schichten angeordnet und der Index ℓ_b des Abarbeitungsstatus entsprechend inkrementiert wurde. Somit folgt, dass die Anzahl der Iterationen letzteren Typs, für die $\ell_b < \frac{1}{3}n$ gilt, konstant ist und ihre summierten Kosten sich also durch $\mathcal{O}(n \cdot \log_2 n)$ abschätzen lassen.

Der Fall $\ell_b \geq \frac{1}{3}n$: die späteren Iterationen

Nachdem bereits ein Drittel der Punkte in Schichten arrangiert wurde, d. h. sobald $\ell_b \geq \frac{1}{3}n$ zum Ende einer Iteration gilt, werden die kodierten Zähler verschoben. Konkret wird zu ihrer Kodierung von nun an das Teilfeld $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$ verwendet. Das Kopieren bzw. Verschieben von kodierten Werten wurde bereits in Abschnitt 4.1.1 beschrieben und resultiert in diesem Falle in einem Zeitaufwand, der linear in der Anzahl der kodierenden Elemente, also in $\mathcal{O}(n)$ ist. Das Eingabefeld ist nach der Verschiebung der kodierten Informationen wie folgt strukturiert:



Für das Kodieren in $\mathtt{A}[0, \ldots, \frac{1}{3}n-1]$ ist zu beachten, dass in diesem Teilfeld bereits eine oder mehrere Maximaschichten (oder Teile hiervon) arrangiert sind. Das initiale Kodieren und das anschließende Aufrechthalten der Zähler c_i in $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$ zerstört daher eventuell die Ordnung nach Schichten sowie die $<_y$-Ordnung innerhalb einzelner Schichten. Diese zerstörenden Effekte sind jedoch nur lokal und müssen zudem erst nach Ablauf des Algorithmus zur Berechnung aller Maximaschichten rückgängig gemacht werden. Wie dies zu bewerkstelligen ist, wird noch näher ausgeführt.

Zählen der Punkte auf den obersten κ Schichten Der Algorithmus für das Zählen der Gesamtanzahl an Punkten der obersten κ Schichten arbeitet exakt wie für frühere Iterationen, d. h. für den Fall $\ell_b \geq \frac{1}{3}n$, beschrieben: Die Anzahl κ der Schichten, deren Elemente zu zählen sind, wird durch $\kappa := \frac{1}{6}n / \log_2 n$ festgelegt und die Zählungen selbst durch Inkrementierungen von κ kodierten Zählern vorgenommen, die in diesem Fall in $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$ residieren.

Der einzige weitere Unterschied zum bereits geschilderten Fall besteht darin, dass während der Präfixsummenberechnung der maximale Index κ' mit der Bedingung $\ell_b + 2\hat{c}_{\kappa'} < n$ (anstatt der Bedingung $\ell_b + 2\hat{c}_{\kappa'} < \frac{2}{3}n$) identifiziert wird.

Der Ablauf des Algorithmus zum Zählen von Punkten auf den obersten κ Schichten ist gegenüber dem bereits geschilderten Fall analog, gleichzeitig jedoch stark vereinfacht: Die Punkte, die die Zeiger kodieren, gehören nicht zu noch zu arrangierenden Schichten und müssen daher nicht (als Ausläufer τ_i einer solchen Schicht)

verschoben werden. Die Laufzeitkosten für das Zählen von Punkten in einer Iteration belaufen sich auf insgesamt $\mathcal{O}(n + \xi \cdot \log_2 n)$, was den Zeitaufwand für die Wiederherstellung der Sortier-Invariante mit einschließt.

Extrahieren und Arrangieren der Punkte auf den obersten κ Schichten Wie für den Fall früher Iterationen werden auch in einer späteren Iteration (d. h. für den Fall $\ell_b < \frac{1}{3}n$) entweder alle ξ Punkte der obersten κ Schichten in $\mathcal{O}(n+\xi\cdot\log_2 n)$ Zeit extrahiert und anschließend arrangiert oder dies wird wegen Platzknappheit für weniger als κ Schichten durchgeführt und abschließend eine einzelne Schicht durch den Skyline-Algorithmus aus Abschnitt 6.2.1 arrangiert, was einen zusätzlichen Zeitaufwand in $\mathcal{O}(\nu \log_2 \nu)$ mit $\nu := n - \ell_b$ mit sich bringt. In beiden Situationen dominieren diese Kosten den Aufwand für die Wiederherstellung der Sortier-Invariante.

Analyse Für die Analyse der Gesamtlaufzeit für den Fall $\ell_b > \frac{1}{3}n$ werden einzelne Iterationen erneut danach klassifiziert, ob alle obersten κ Schichten arrangiert werden konnten. Ist dies der Fall, so wurden $\xi \geq \kappa = \frac{1}{6}n/\log_2 n$ Punkte ihren Schichten zugeordnet und in die gewünschte Darstellung überführt. Somit kann jedem dieser Punkte $\mathcal{O}(\log_2 n)$ Zeitaufwand zugeordnet werden sowie der Iteration selbst Laufzeitkosten von $\mathcal{O}(n)$. Das Arrangieren von $\xi \geq \frac{1}{6}n/\log_2 n$ Punkten impliziert zudem, dass die Anzahl von Iterationen dieses Typs durch $\mathcal{O}(\log_2 n)$ beschränkt ist und daher die globalen Kosten, die Iterationen dieser Art zugeschlagen werden, in $\mathcal{O}(n \cdot \log_2 n)$ sind.

Iterationen, in denen nur $\kappa' < \kappa$ Schichten während eines Überstreichens arrangiert werden können, werden durch die Skyline-Berechnung der nächstfolgenden Schicht abgeschlossen. Nach dieser Berechnung sind (auf Grund der Definition von κ') zumindest $\frac{1}{2}(n - \ell_b)$ Punkte in dieser Iteration abgearbeitet, d. h. in Schichten arrangiert worden. Demzufolge ist ℓ_b um mindestens diese Anzahl inkrementiert worden und somit ist bei der nächsten Iteration diesen Typs das Feld der noch abzuarbeitenden Punkte höchstens halb so groß. Die Größe dieser Felder verhält sich daher wie eine geometrische Reihe; die Gesamtlaufzeit über alle Iterationen diesen Typs besitzt daher eine Komplexität von $\mathcal{O}(n \cdot \log_2 n)$ — die schon für die alleinige erste dieser Iterationen zu veranschlagen ist.[9]

Wiederherstellen der Ordnung nach Schichten in $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$

Nach der letzten Iteration des Algorithmus, wenn alle Eingabepunkte ihren Schichten zugeordnet wurden, bleibt noch auf den Schichten, die (gänzlich oder teilweise) im Teilfeld $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$ gespeichert wurden, die Ordnung wiederherzustellen, die durch die kodierten Zeiger lokal permutiert wurde.[10] Die Hauptschwierigkeit hierbei besteht darin, dass die Größe der Schichten und damit auch die Übergänge zwischen

[9] Ein formaler Nachweis ergibt sich aus folgender Ungleichungskette für die Summe $f(n)$ der Kosten über alle Iterationen des angegebenen Typs: $f(n) \leq \sum_{i=0}^{\log_2 n} n/2^i \cdot \log_2(n/2^i) = \sum_{i=0}^{\log_2 2n} 2^i \cdot \log_2(2^i) \leq \sum_{i=0}^{\log_2 2n} 2^n = \log_2 n \cdot \sum_{i=0}^{\log_2 2n} 2^i \leq 4 \cdot n \cdot \log_2 n \in \mathcal{O}(n \cdot \log_2 n)$

[10] Es ist zu beachten, dass zu Beginn jeder Iteration die kodierten Zählerwerte reinitialisiert (d. h. auf 0 gesetzt) werden können, ohne dass die eigentliche Ordnung der kodierenden Elemente nach Schichten hierfür bekannt zu sein braucht.

zwei Schichten nicht mehr bekannt sind und auch nicht in konstantem zusätzlichen Speicher vorgehalten werden könnten. Insbesondere können diese Punkte also nicht einfach absteigend bezüglich $<_y$ sortiert werden. Es kann jedoch die Lokalität der Permutationen, die die Kodierungen verursacht haben, ausgenutzt werden. Der im Folgenden vorgestellte Algorithmus zur Wiederherstellung der Ordnung nach Schichten sowie bezüglich der absteigenden $<_y$-Ordnung innerhalb einzelner Schichten iteriert daher über die in $\mathtt{A}[0, \ldots, \frac{1}{3}n - 1]$ befindlichen Paare von Bit-Nachbarn und hält dabei die folgende Invariante aufrecht:

Ordnungs-Invariante Erreicht die Betrachtung die Bit-Nachbarn $q := \mathtt{A}[2i]$ und $r := \mathtt{A}[2i + 1]$, so ist die korrekte Ordnung nach Schichten und bezüglich $<_y$ innerhalb einzelner Schichten bereits auf $\mathtt{A}[0, \ldots, 2i - 1]$ wiederhergestellt.

Diese Invariante lässt sich initial, d. h. für das Teilfeld $\mathtt{A}[0, \ldots, 2 - 1] = \mathtt{A}[0, 1]$, einfach herstellen: Die beiden Elemente müssen genau dann vertauscht werden, wenn $\mathtt{A}[0] <_y \mathtt{A}[1]$ gilt. Mit dem Induktionsprinzip sei daher $i \geq 1$ angenommen und die zu betrachtenden Punkte mit $q := \mathtt{A}[2i]$ und $r := \mathtt{A}[2i+1]$ bezeichnet. Für den Fall, dass q von r oder r von q dominiert wird, ergibt sich die korrekte Reihenfolge der Punkte aus den y-Koordinaten von q und r, da beide Punkte unterschiedlichen Schichten angehören und der dominierte der beiden Punkte dem dominierenden nachzufolgen hat, vergleiche den linken Teil der Abbildung 6.2, in der das Symbol "|" den Beginn einer neuen Schicht anzeigt.

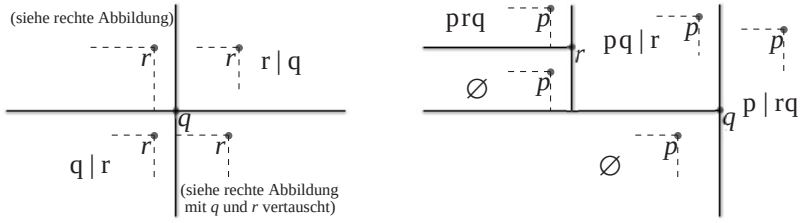
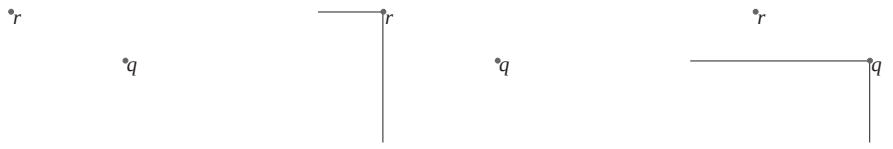


Abbildung 6.2: Wiederherstellung der Ordnung nach Schichten sowie bezüglich $<_y$ für q und r. Das Symbol "|" steht für den Beginn einer neuen Schicht.

Für den verbleibenden Fall, dass keiner der beiden Punkten den jeweils anderen dominiert, sei ohne Beschränkung der Allgemeinheit angenommen, dass r bezüglich der lexikografischen Ordnung nach absteigender y- und aufsteigender x-Koordinate kleiner als q ist. Demzufolge ist die relative Lage von q und r eindeutig, wie im linken Teil der Abbildung 6.2 oben links und in Abbildung 6.3(a) skizziert.

Nach Induktionsannahme ist $p := \mathtt{A}[2i - 1]$ in seiner korrekten Position bezüglich der zu etablierenden Ordnung. Folglich ist die korrekte Permutation von p, q und r entweder pqr oder prq. Für die Bestimmung der korrekten Permutation ist jedoch zusätzlich zu bestimmen, ob sich zwischen den Elementen dieser Permutation der Übergang zu einer neuen Maximaschicht befindet, also ob q oder r den Beginn einer neuen Schicht darstellen. Für die relative Ordnung von p, q und r sind demnach die folgenden Fälle denkbar.



(a) (b) (c)

Abbildung 6.3: Geometrische Konfigurationen zur Bestimmung der Ordnung von q and r nach Schichten.

(1)	pqr	(2)	$p\|qr$	(3)	$pq\|r$	(4)	$p\|q\|r$
(5)	prq	(6)	$p\|rq$	(7)	$pr\|q$	(8)	$p\|r\|q$

Die relative Lage von q und r, bzw. das Fehlen von Dominanz zwischen diesen beiden Punkten, schließt aus, dass r und q in der Reihenfolge qr Elemente derselben Schicht sind. Somit können die Situationen (1) und (2) in obiger Aufzählung nicht auftreten. Abbildung 6.3(b) veranschaulicht, warum die Reihenfolge rq nicht auftreten kann, wenn r der letzte Punkt seiner Schicht ist: Wenn r in seiner Schicht keinen Nachfolger besitzt, müsste diese Schicht sich von r aus vertikal nach $y = -\infty$ fortsetzen; da sich jedoch q rechts von r befindet, läge q damit „oberhalb“ der Schicht von r und die Reihenfolge rq bezüglich der wiederherzustellenden Sortierung wäre inkorrekt. Daher sind auch die Situationen (7) und (8) ausgeschlossen. In Abbildung 6.3(c) ist veranschaulicht, warum q keine eigene Schicht konstituieren kann. In diesem Fall läge die Situation (4) vor, da die Situation (8) bereits ausgeschlossen wurde. Die Situation (4) impliziert, dass die Schicht, der q angehört, sich von q aus horizontal nach $x = -\infty$ fortsetzt. Da r sich jedoch oberhalb dieser Fortsetzung befindet, ergibt sich ein Widerspruch dazu, dass r unterhalb der Schicht von q liegt. Somit ist auch Situation (4) auszuschließen.

Somit stellen nur die Situationen (3), (5) und (6), also nur die Konfigurationen $pq|r$, prq und $p|rq$ tatsächlich mögliche Konfigurationen der Punkte p, q und r bezüglich der wiederherzustellenden Ordnung dar. Diese drei Optionen für die korrekte Anordnung der drei Punkte sind allein an Hand ihrer $<_x$-Ordnung zu unterscheiden, was im rechten Teil der Abbildung 6.2 veranschaulicht ist und im Folgenden ausgeführt wird.

Wenn alle Punkte derselben Schicht angehören, also die Konfiguration prq vorliegt, ist auch die x-Ordnung dieser Punkte prq, da eine Maximaschicht monoton sowohl in absteigender y-Richtung als auch in aufsteigender x-Richtung ist.

In der Situation $p|rq$ ist p tatsächlich der letzte Punkt seiner Schicht, da nach Induktionsannahme p (sowie alle vor p im Eingabefeld befindlichen Punkte) sich an ihrer korrekten Position bezüglich der wiederherzustellenden Ordnung befinden. Lägen nun q oder r rechts von p, so wären sie ebenfalls Punkte der Schicht von p oder sogar Elemente einer höheren Schicht. Zudem liegt q rechts von r, da nach obiger Annahme nur Fälle betracht werden, in denen r bezüglich der lexikografischen Ordnung nach absteigender y- sowie aufsteigender x-Koordinate kleiner als q ist. Somit ergibt sich die x-Ordnung in dieser Situation eindeutig durch rqp.

In der verbleibenden Situation $pq|r$ liegt p oberhalb und rechts von q, da p und q Teil derselben Schicht sind. Des Weiteren befindet sich r rechts und oberhalb von q (wiederum wegen der getroffenen Annahme). Läge p links nicht nur von q, sondern auch von r, so wären p, q und r Teil derselben Schicht oder p wäre von r dominiert. In beiden Fällen ergibt sich ein Widerspruch zur betrachteten Situation, in welcher sich die Ordnung der Punkte bezüglich $<_x$ somit eindeutig zu rpq ergibt.

Somit ist nachgewiesen, dass die korrekte Reihenfolge der betrachteten Bit-Nachbarn q und r sich eindeutig und eventuell durch zusätzliche Betrachtung des Vorgängerpunktes p und durch Bestimmung der x-Ordnung dieser drei Punkte herleiten lässt. Die Kosten hierfür sind hinsichtlich Laufzeit und Speicherplatz jeweils konstant. Das Induktionsprinzip liefert das gewünscht Resultat:

Lemma 6.3.7 *Die lexikografische Sortierung einer n-elementigen Punktmenge nach Maximaschichten sowie bezüglich der y-Koordinate lässt sich* in-place *und in linearer Zeit wiederherstellen, sofern diese Ordnung nur durch Vertauschung von Bit-Nachbarn verletzt ist.*

Zusammenfassung Die Summation der Kosten über alle (frühen wie späteren) Iterationen des Algorithmus zur Berechnung der Maximaschichten sowie der Kosten für die Wiederherstellung der korrekten Ordnung der Ausgabe nach allen Iterationen ergibt sich zu $\mathcal{O}(n \cdot \log_2 n)$. Kombiniert mit der Analyse, dass jedem abzuarbeitenden Punkt zusätzlich $\mathcal{O}(\log_2 n)$ Kosten für die einzige Iteration, in der er entsprechend seiner Schicht gruppiert wurde, zuzuschlagen sind, ergibt sich das angestrebte Resultat:

Satz 6.3.8 *Es lassen sich alle Maximaschichten einer n-elementigen Menge zweidimensionaler Punkte* in-place *und in asymptotisch optimaler Laufzeit von $\mathcal{O}(n \cdot \log_2 n)$ bestimmen und die Eingabe nach Schichten und zudem die Elemente einzelner Schichten absteigend bezüglich ihrer y-Koordinate arrangieren.*

Kapitel 7

Zusammenfassung

7.1 Modellierung von und Algorithmik für Bewegungsdaten

Adäquate Repräsentation von Bewegungsdaten Im ersten Teil dieser Arbeit wurden zunächst Anforderungen an die Repräsentation der Bewegungen mobiler Objekte zum Zwecke der Datenhaltung und -verarbeitung beschrieben. Es wurde belegt, dass die in bestehenden Systemen verwendete Repräsentation durch stückweise lineare Funktionen nicht allen diesen Anforderungen gerecht wird, und alternative Repräsentationskonzepte wurden vorgestellt. Die Modellierung eines Rahmenwerks wurde präsentiert, das mehrere solcher Repräsentationskonzepte für Bewegungen bereitstellt, um auch verschieden charakterisierte Bewegungen jeweils realistisch darstellen zu können. Theoretische Laufzeitanalysen sowie eine erste Evaluation innerhalb des Rahmenwerks zeigen, dass die Performanzverluste beim Übergang von stückweise linearen Funktionen für die Bewegungsbeschreibung zu realistischeren Kurven, wie beispielsweise kubischen Splines, moderat sind. Die Abgeschlossenheit eines Typsystems für zeitvariante räumliche Daten gegen zeitvariante Verarbeitungen von Bewegungen kann hingegen nur in eingeschränkter Form erreicht werden, was in dieser Arbeit veranschaulicht und belegt wurde. Es wurde ein Lösungsvorschlag für dieses Problem präsentiert, der auf adäquater Näherung solcher Verarbeitungsergebnisse beruht und innerhalb des vorgestellten Rahmenwerkes realisiert ist.

Verarbeitung divers repräsentierter Bewegungen Die in Kapitel 3 beschriebene Erweiterung der im Rahmenwerk umgesetzten objektorientierten Modellierung mobiler Objekte ermöglicht, verschiedenste räumlich-temporale Anfragetypen und algorithmische Aufgaben zu realisieren, ohne dabei einen fixierten oder einheitlichen Typ für die Repräsentation der betrachteten Bewegungen vorauszusetzen. Kapselt man in „klassischen“ Algorithmen für komplexere Aufgabenstellungen und Anfragen die Aufrufe von algorithmischen Primitivopertionen, so können diese Algorithmen mit geringem Aufwand so adaptiert und in das vorgestellte Rahmenwerk integriert werden, dass sie für beliebige Bewegungsrepräsentationstypen anwendbar sind — auch wenn sie ursprünglich für einen spezifischen Typ von Bewegungsrepräsentatio-

nen, wie stückweise linearen Kurven, konzipiert wurden. Zur Realisierung besagter Primitivoperationen wurden Methoden präsentiert und in das Rahmenwerk integriert, die die Kenntnis über Restriktionen zu Bewegungen, wie Geschwindigkeits- und Beschleunigungsbeschränkungen, ausnutzen. Für mehrere solcher Primitivrealisierungen wurden ihre Korrektheit und ihre numerische Robustheit nachgewiesen sowie ihre Effizienz analysiert. Hierfür wurde beispielhaft die Aufgabenstellung der Kollisionserkennung näher betrachtet. Zusätzlich erlaubt die Modellierung des Rahmenwerkes auch die Verwendung klassischer numerischer und algebraischer Methoden zur Primitivrealisierung, wie sie in bereits vorgestellten Implementierungen von Bewegungen verarbeitenden Systemen eingesetzt werden.

Die Zielsetzung der generischen Operabilität des Rahmenwerks — unabhängig von den verwendeten Konzepten zur Bewegungsrepräsentation — ist unter anderem insofern erreicht worden, als dass die Integration neuer Typen von Bewegungsrepräsentationen möglich ist, ohne dass die Realisierung von Anfragetypen und weiteren Funktionalitäten angepasst werden muss, um auch Instanzen diesen neuen Typs zu verarbeiten. Die Umsetzbarkeit des in Kapitel 3 beschriebenen Ansatzes konnte durch eine funktionsfähige Implementierung einer entsprechenden Erweiterung des in Kapitel 2 vorgestellten Rahmenwerks belegt werden.

Ausblick Für die Fortführung der im ersten Teil dieser Arbeit geschilderten Forschung sind insbesondere die folgenden Bereiche von Interesse. Wie in Kapitel 3.2 geschildert, bildet die Kapselung von Primitiven die Grundlage auch bei der Entwicklung kinetischer Datenstrukturen. Hierfür wurde bereits von Guibas *et al.* [GKR04] eine Implementierung entwickelt, die allerdings von polynomialen Bewegungsrepräsentationen ausgeht und entsprechende algebraische und semi-algebraische Primitivrealisierungen anbietet. Das in dieser Arbeit modellierte Rahmenwerk kann bezüglich jenes Systems als eine Erweiterung angesehen werden, die die Behandlung von diversen und insbesondere realistischeren Bewegungsrepräsentationen unterstützt. Eine Realisierung einer solchen Erweiterung, d. h. eine Integration des hier vorgestellten Ansatzes mit der Implementierung zur Entwicklung kinetischer Datenstrukturen, scheint daher vielversprechend.

Des Weiteren wäre eine ausführlichere praktische Evaluation des in Kapitel 3 beschriebenen Ansatzes hilfreich, um Aufschluss darüber zu erhalten, ob ein Verlust an Laufzeiteffizienz in konkreten praktischen Szenarien für die generische Verarbeitung verschieden typisierter Bewegungsrepräsentationen in Kauf zu nehmen ist und falls ja, wie hoch dieser Verlust gegenüber Verarbeitungen ist, die auf einen bestimmten Typ von Bewegungsrepräsentation abgestimmt sind.

7.2 Speichereffiziente Algorithmen für den Einsatz in mobilen Umgebungen

Im zweiten Teil dieser Arbeit wurden mehrere Alaufzeiteffiziente Algorithmen vorgestellt, die nur eine konstante Anzahl an Wörtern zusätzlich zu dem die Eingabe enthaltenden Speicher verwenden. Es wurde ausgeführt, dass die Entwicklung speichereffizienter Algorithmen im Kontext mobiler Umgebungen wie mobiler Kom-

munikationssnetze oder Sensornetzwerke relevant ist, wie bereits von Agarwal *et al.* [AGE+02] und Roddick *et al.* [RHE+04] erkannt. Insbeondere wurde für diesen Kontext die Relevanz der in dieser Arbeit betrachteten Aufgabenstellungen des robusten Schätzens von Geraden sowie der Bestimmung von Maximamengen und -schichten aufgezeigt. Bezüglich dieser Aufgabenstellungen konnten die nachfolgend zusammengefassten algorithmischen Ergebnisse präsentiert werden.

Robustes Geradenschätzen Die Berechnung robuster Geradenschätzer ist *in-place*, d. h. unter Verwendung von zusätzlich zur Eingabe nur konstanten Anzahl an Speicherwörtern, möglich. Für die Berechnung des bezüglich seines *breakdown values* optimal robusten *repeated median*-Schätzer konnte ein Algorithmus mit einer erwarteten Laufzeit von $\mathcal{O}(n \cdot \log_2^3 n)$ angegeben werden. Diese Laufzeit weicht um einen zweifachen logarithmischen Faktor von der optimalen — und mit Verwendung weiteren Speichers erreichbaren — Laufzeit [MMN98] im erwarteten Fall ab. Für den ebenfalls sehr robusten Theil-Sen-Schätzer wurde ein *in-place*-Algorithmus mit optimaler erwarteter Laufzeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ präsentiert. Dieser Algorithmus ist auch Lösung des klassischen *slope selection*-Problems der Bestimmung einer bezüglich ihrer Steigung k-ten Geraden unter den Geraden, die durch jeweils zwei Elemente aus einer übergebenen Menge von Punkten bestimmt sind. Der Übergang in den Dualraum ermöglicht auch die im erwarteten Fall laufzeitoptimale *in-place*-Bestimmung des (bezüglich einer gewählten Ausrichtung) k-ten Schnittpunktes einer Menge von Geraden in der Ebene, welche ein Teilproblem in vielen Aufgabenstellungen der algorithmischen Geometrie darstellt.

Bestimmen von Maximamengen und -schichten Die Bestimmung der Maximamenge zu einer zwei- oder dreidimensionalen Punktmenge sowie die Gruppierung einer zweidimensionalen Punktmenge in Maximaschichten ist jeweils *in-place* und in optimaler Zeitkomplexität von $\mathcal{O}(n \cdot \log_2 n)$ möglich. Insbesondere das letztgenannte Ergebnis ist auch insofern von theoretischem Interesse, als dass der entsprechende Algorithmus der erste laufzeitoptimale ist, der *in-place* eine komplette Anordnung einer mehrdimensionalen Eingabe berechnet, wobei diese Anordnung anscheinend nicht in effizienter Weise allein mit Hilfe des divide & conquer-Prinzips oder mit Grahams Scan zu bestimmen ist.

Ausblick Eine naheliegende Richtung für die weitere Forschung in den im zweiten Teil dieser Arbeit betrachteten algorithmischen Gebieten ist die Erweiterung der in dieser Arbeit präsentierten Ergebnisse für höherdimensionale Eingaben. Sowohl für die betrachteten robusten Geradenschätzer als auch für die Bestimmung von Maximamengen und -schichten existieren laufzeiteffiziente Algorithmen, vergleiche [MN93] bzw. [KLP75, BG04], für die eine Realisierung von speichereffizienten Varianten nicht ausgeschlossen scheint. Gleiches gilt für Lösungen von Aufgabenstellungen, die sich auf *slope selection*-Algorithmen abstützen. Zudem fußen die vorgestellten Berechnungen robuster Schätzer auf der Technik der randomisierten Suche, die auch für weitere Problemstellungen dienlich sein kann und für die somit laufzeiteffiziente randomsierte *in-place*-Algorithmen realisierbar scheinen.

Des Weiteren wäre eine praktische Evaluation von speichereffizienten Algorithmen — insbesondere in konkreten mobilen Umgebungen — hilfreich, um die Laufzeitkosten, die aus der Verwendung einzelner speichereffizienter algorithmischer Techniken resultieren, abschätzen zu können. Mit Hilfe solcher Evaluationen ließe sich insbesondere ein Anpassen von Algorithmen — wie der hier beschriebenen — prüfen, so dass zwischen eingespartem Speicherplatz und den daraus im Konkreten resultierenden Laufzeitkosten abgewägt werden kann.

Literaturverzeichnis

[AAV01] Agarwal, P. K., L. A. Arge und J. Vahrenhold: *Time-Responsive External Data Structures for Moving Points.* In: *Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS '01)*, Band 2125 der Reihe *Lecture Notes in Computer Science*, Seiten 50–61. Springer, Berlin, 2001.

[ABC+03] Aslam, J., Z. Butler, F. Constantin, V. Crespi, G. Cybenko und D. Rus: *Tracking a Moving Object with a Binary Sensor Network.* In: *Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys '03)*, Seiten 150–161. ACM Press, 2003.

[ACGK94] Afrati, F., S. S. Cosmadakis, S. Grumbach und G. M. Kuper: *Linear vs. Polynomial Constraints in Database Query Languages.* In: Borning, A. (Herausgeber): *Principles and Practice of Constraint Programming*, Seiten 181–192. Springer, Berlin, Heidelberg, 1994.

[AG96] Arnold, K. und J. Gosling: *The Java Programming Language.* Addison-Wesley–Longman, Reading, MA, 1996.

[AGE+02] Agarwal, P. K., L. J. Guibas, H. Edelsbrunner, J. Erickson, M. Isard, S. Har-Peled, J. Hershberger, C. Jensen, L. Kavraki, P. Koehl, M. Lin, D. Manocha, D. Metaxas, B. Mirtich, D. Mount, S. Muthukrishnan, D. Pai, E. Sacks, J. Snoeyink, S. Suri und O. Wolefson: *Algorithmic Issues in Modeling Motion.* ACM Computing Surveys, 34(4):550–572, 2002.

[AHU82] Aho, A. V., J. E. Hopcroft und J. D. Ullman: *Data Structures and Algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley, 1982.

[AK00] Aurenhammer, F. und R. Klein: *Handbook of Computational Geometry*, Kapitel Voronoi diagrams, Seiten 201–290. Elsevier Science/North-Holland, Amsterdam, 2000.

[Arm88] Armstrong, M. P.: *Temporality in Spatial Databases.* In: *Proceedings of GIS/LIS '88: Accessing the World*, Band 2, Seiten 880–889. American Society for Photogrammetry and Remote Sensing, 1988.

[AS00] Agarwal, P. K. und M. Sharir: *Pipes, Cigars, and Kkreplach: the Union of Minkowski Sums in Three Dimensions.* Discrete & Computational Geometry, 24(4):143–153, 2000.

[Ash56] Ashby, W. R.: *An Introduction to Cybernetics.* Chapman & Hall, London, 1956.

[ASSC02] Akyildiz, I. F., W. Su, Y. Sankarasubramaniam und E. Cayirci: *A Survey on Sensor Networks.* IEEE Communications Magazine, 40(4):102–114, 2002.

[Bal95] Balaban, I. J.: *An Optimal Algorithm for Finding Segments Intersections.* In: *Proceedings of the 11th Annual ACM Symposium on Computational Geometry (SoCG '95)*, Seiten 211–219. ACM Press, 1995.

[Bas99] Basch, J.: *Kinetic Data Structures.* Doktorarbeit, Stanford University, Computer Science Deptartment, 353 Serra Mall, Stanford, CA 94305-9025, USA, 1999.

[BBB87] Bartels, R. H., J. C. Beatty und B. A. Barsky: *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, Los Altos, 1987.

[BBHV04] Becker, L., H. Blunck, K. H. Hinrichs und J. Vahrenhold: *A Framework for Moving Objects.* In: *Proceedings of the 15th International Conference on Database and Expert Systems Applications (DEXA '04)*, Band 3180 der Reihe *Lecture Notes in Computer Science*, Seiten 854–863. Springer, 2004.

[BBS83] Blum, L., M. Blum und M. Shub: *Comparison of Two Pseudorandom Number Generators.* In: *Proceedings of the 2nd Annual International Cryptology Conference (Crypto '82)*, Seiten 61–78, New York, 1983.

[BC98] Brönnimann, H. und B. Chazelle: *Optimal slope selection via cuttings.* Computational Geometry: Theory and Applications, 10:23–39, August 1998.

[BC04] Brönnimann, H. und T. M. Chan: *Space-efficient Algorithms for Computing the Convex Hull of a Simple Polygonal Line in Linear Time.* In: *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN '04)*, Seiten 162–171, Buenos Aires, Argentina, 2004.

[BCC04] Brönnimann, H., T. M. Chan und E. Y. Chen: *Towards in-place Geometric Algorithms and Data Structures.* In: *Proceedings of the 20th Annual ACM Symposium on Computational Geometry (SoCG '04)*, Seiten 239–246, New York, NY, USA, 2004. ACM Press.

[BCL90] Bentley, J. L., K. L. Clarkson und D. B. Levine: *Fast Linear Expected-Time Algorithms for Computing Maxima and Convex Hulls.* In: *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, Seiten 179–187, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[BEG+04] Basch, J., J. Erickson, L. J. Guibas, J. Hershberger und L. Zhang: *Kinetic Collision Detection between Two Simple Polygons.* Computational Geometry Theory & Applications, 27(3):211–235, 2004.

[Ben80] Bentley, J. L.: *Multidimensional divide-and-conquer.* Communications of the ACM, 23(4):214–229, 1980.

[Ben84] Bentley, J. L.: *Squeezing Space.* Communications of the ACM, 27(5):416–421, 1984.

[BFM04] Bender, M. A., M. Farach-Colton und M. Mosteiro: *Insertion Sort is O(n log n).* In: *Proceedings of the 3rd International Conference on Fun with Algorithms (FUN)*, Seiten 16–23, 2004.

[BG99] Basch, J. und L. J. Guibas: *Data Structures for Mobile Data.* Journal of Algorithms, 31(1):1–28, 1999.

[BG04] Buchsbaum, A. L. und M. T. Goodrich: *Three-Dimensional Layers of Maxima.* Algorithmica, 39(4):275–286, 2004.

[BH75] Bryson, A. E. und Y.-C. Ho: *Applied Optimal Control.* Hemisphere Pub. Corp., 1975.

[BHKR05] Bruce, R., M. Hoffmann, D. Krizan und R. Raman: *Efficient Update Strategies for Geometric Computing with Uncertainty.* Theory of Computing Systems, 38(4):411–423, 2005.

[BHPV04] Blunck, H., K. H. Hinrichs, I. Puke und J. Vahrenhold: *Verarbeitung von Trajektorien mobiler Objekte.* In: *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung. Beiträge zu den Münsteraner GI-Tagen 2004*, Band 22 der Reihe *IfGI prints*, Seiten 3–12, 2004.

[BHSV06] Blunck, H., K. H. Hinrichs, J. Sondern und J. Vahrenhold: *Modeling and Engineering Algorithms for Mobile Data.* In: *Progress in Spatial Data Handling: Proceedings of the 12th International Symposium on Spatial Data Handling (SDH '06)*, Seiten 61–77, 2006.

[BIK+04] Brönnimann, H., J. Iacono, J. Katajainen, P. Morin, J. Morrison und G. Toussaint: *Space-efficient Planar Convex Hull Algorithms.* Theoretical Computer Science, 321(1):25–40, 2004.

[BKS01] Börzsönyi, S., D. Kossmann und K. Stocker: *The Skyline Operator*. In: *Proceedings of the 17th International Conference on Data Engineering*, Seiten 421–430, Washington, DC, USA, 2001. IEEE Computer Society.

[Bli00] Blin, K.: *Stochastic Conflict Detection for Air Traffic Management*. EEC Note No. 5/2000, EUROCONTROL Experimental Centre, 2000.

[Blu02] Blunck, H.: *Repräsentation von Bewegung in Datenbanken*. Diplomarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2002.

[BMM+06] Bose, P., A. Maheshwari, P. Morin, J. Morrison, M. Smid und J. Vahrenhold: *Space-Efficient Geometric Divide-and-Conquer Algorithms*. Computational Geometry: Theory & Applications, 2006.

[BO79] Bentley, J. L. und T. Ottmann: *Algorithms for Reporting and Counting Geometric Intersections*. IEEE Transactions on Computing, C-28:643–647, 1979.

[Bob01] Bobko, P.: *Correlation and Regression: Applications for Industrial Organizational Psychology and Management*. SAGE Publications, 2. Auflage, 2001.

[Boe02] Boege, K.: *Qualitative Beurteilung verschiedener Realisierungen eines Modells zur Repräsentation zeitvarianter räumlicher Daten*. Diplomarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2002.

[Bre04] Breimann, C.: *Anfragebearbeitung in einem Datenbank-Kernsystem für Geo-Anwendungen unter Verwendung einer generischen Komponente zur Anfrageoptimierung*. Doktorarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2004.

[BS76] Bentley, J. L. und M. I. Shamos: *Divide-and-Conquer in Multidimensional Space*. In: *Proceedings of the 8th Annual ACM Symposium on Theory of Computing (STOC '76)*, Seiten 220–230. ACM Press, 1976.

[BS99] Boissonnat, J.-D. und J. Snoeyink: *Efficient Algorithms for Line and Curve Segment Intersection using Restricted Predicates*. In: *Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SoCG '99)*, Seiten 370–379. ACM Press, 1999.

[BSM99] Bronstein, I. N., K. A. Semendjajew und G. Musiol: *Taschenbuch der Mathematik*. Verlag Harri Deutsch, Frankfurt am Main, Thun, 4. Auflage, 1999.

[BV02] Boissonnat, J. und A. Vigneron: *Elementary Algorithms for Reporting Intersections of Curve Segments*. Computational Geometry Theory & Applications, 21:167–175, 2002.

[BV06a] Blunck, H. und J. Vahrenhold: *In-Place Algorithms for Computing (Layers of) Maxima.* In: *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT '06)*, Band 4059 der Reihe *Lecture Notes of Computer Science*, Seiten 363–374. Springer, Berlin, 2006.

[BV06b] Blunck, H. und J. Vahrenhold: *In-Place Randomized Slope Selection.* In: *Proceedings of the 6th Conference on Algorithms and Complexity (CIAC '06)*, Band 3998 der Reihe *Lecture Notes of Computer Science*, Seiten 30–41. Springer, Berlin, 2006.

[BVH96] Becker, L., A. Voigtmann und K. Hinrichs: *Temporal Support for Geo-Data in Object-Oriented Databases.* In: *Proceedings of the 7th International Conference on Database and Expert Systems Applications (DEXA '96)*, Band 1134 der Reihe *Lecture Notes in Computer Science*, Seiten 79–93. Springer, 1996.

[CEH+01] Cerpa, A., J. Elson, M. Hamilton, J. Zhao, D. Estrin und L. Girod: *Habitat Monitoring: Application Driver for Wireless Communications Technology.* In: *Workshop on Data communication in Latin America and the Caribbean (SIGCOMM LA '01)*, Seiten 20–41. ACM Press, 2001.

[Cha85] Chazelle, B.: *On the Convex Layers of a Planar Set.* IEEE Transactions on Information Theory, 31:509–517, 1985.

[Cha96] Chan, T. M.: *Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions.* Discrete & Computational Geometry, 16, 1996.

[Chu92] Chui, C. K.: *An Introduction to Wavelets.* Academic Press, London, UK, 1992.

[CKP04] Cheng, R., D. V. Kalashnikov und S. Prabhakar: *Querying Imprecise Data in Moving Object Environments.* IEEE Transactions on Knowledge and Data Engineering (TKDE), 16(9):1112–1127, 2004.

[Cla94] Clarkson, K. L.: *More Output-sensitive Geometric Algorithms.* In: *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS '94)*, Seiten 695–702, 1994.

[CLR01] Cormen, T. H., C. E. Leiserson und R. L. Rivest: *Introduction to Algorithms.* MIT Press, Cambridge, MA, 2. Auflage, 2001.

[CM03] Chen, E. Y. und T. M.Chan: *A Space-Efficient Algorithm for Segment Intersection.* In: *Proceedings of the 15th Canadian Conference on Computational Geometry (CCCG '03)*, 2003.

[CM05] Chen, E. Y. und T. M.Chan: *Space-efficient Algorithms for Klee's Measure Problem.* In: *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG '05)*, 2005.

[CR97] Chomicki, J. und P. Z. Revesz: *Constraint-Based Interoperability of Spatiotemporal Databases.* In: *Proceedings of the 5th International Symposium on Large Spatial Databases (SSD '97)*, Band 1262, Seiten 142–161, Berlin, 1997.

[CR99a] Chomicki, J. und P. Z. Revesz: *Constraint-Based Interoperability of Spatiotemporal Databases.* GeoInformatica, 3(3):211–243, 1999.

[CR99b] Chomicki, J. und P. Z. Revesz: *A General Framework for Specifying Spatiotemporal Objects.* In: *Proceedings of the 6th International Workshop on Temporal Representation and Reasoning (TIME '99)*, Seiten 41–46. IEEE Computer Society, 1999.

[CS95] Carlsson, S. und M. Sundström: *Linear-time In-place Selection in Less than 3n Comparisons.* In: *Proceedings of the 6th Annual International Symposium on Algorithms and Computation (ISAAC'02)*, Band 3827 der Reihe *Lecture Notes in Computer Science*, Seiten 244–253, Berlin, 1995. Springer.

[CSSS89] Cole, R., J. S. Salowe, W. L. Steiger und E. Szemerédi: *An Optimal-Time Algorithm for Slope Selection.* SIAM Journal on Computing, 18(4):792–810, August 1989.

[CWT03] Cao, H., O. Wolfson und G. Trajcevski: *Spatio-Temporal Data Reduction with Deterministic Error Bounds.* In: *Proceedings of the 2003 joint workshop on Foundations of mobile computing (DIAL-M '03)*, Seiten 33–42. ACM Press, 2003.

[Dan97] Daniel, H.: *Physik*, Band 1. Gruyter, 1997.

[dB78] Boor, C. de: *A Practical Guide To Splines.* Springer, New York, 1978.

[dBvKOS97] Berg, M. de, M. van Kreveld, M. Overmars und O. Schwarzkopf: *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1997.

[Dem02] Demaine, E. D.: *Cache-Oblivious Algorithms and Data Structures.* In: *Lecture Notes from the EEF Summer School on Massive Data Sets, BRICS*, Lecture Notes in Computer Science, University of Aarhus, Denmark, 2002. Springer, Berlin.

[DG04a] Ding, Z. und R. Güting: *Managing Moving Objects on Dynamic Transportation Networks.* In: *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*, 2004.

[DG04b] Ding, Z. und R. Güting: *Uncertainty Management for Network Constrained Moving Objects.* In: *Proceedings of the 15th International Conference on Database and Expert Systems Applications (DEXA '04)*, 2004.

[DMN92] Dillencourt, M. B., D. M. Mount und N. S. Netanyahu: *A Randomized Algorithm for Slope Selection*. International Journal of Computational Geometry and Applications, 2(1):1–27, März 1992.

[DNW05] Duckham, M., S. Nittel und M. Worboys: *Monitoring Dynamic Spatial Fields Using Responsive Geosensor Networks*. In: *Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems (ACM GIS '05)*, Seiten 51–60, New York, NY, USA, 2005. ACM Press.

[Dun05] Dunkelmann, A.: *Fehlerbetrachtungen für Trajektorien und Anfragen in räumlich-temporalen Datenbanken unter Berücksichtigung verschiedener räumlich-temporaler Maße*. Diplomarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2005.

[DZ04] Dai, H. und X. Zhang: *Improved Linear Expected-Time Algorithms for Computing Maxima*. In: *Proceedings ofthe 6th Latin American Symposium on Theoretical Informatics (LATIN '04)*, Band 2976 der Reihe *Lecture Notes in Computer Science*, Seiten 181 – 192. Springer Berlin, 2004.

[EGSV99] Erwig, M., R. H. Güting, M. Schneider und M. Vazirgiannis: *Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases*. GeoInformatica, 3(3):269–296, March 1999.

[EHM04] Erickson, J., S. Har-Peled und D. Mount: *On the Least Median Square Problem*. In: *Proceedings of the 20th Annual ACM Symposium on Computational Geometry (SoCG '04)*, Seiten 273–279. ACM Press, 2004.

[El-06] El-Rabbany, A.: *Introduction to GPS: The Global Positioning System*. Artech House, 2006.

[EM90] Edelsbrunner, H. und E. P. Mücke: *Simulation of Simplicity: a Technique to Cope with Degenerate Cases in Geometric Algorithms*. ACM Transactions on Graphics, 9(1):66–104, 1990.

[EN03] Elmasri, R. und S. B. Navathe: *Fundamentals of Database Systems*. Benjamin/Cummings, 4. Auflage, 2003.

[ES99] Erwig, M. und M. Schneider: *Developments in Spatio-Temporal Query Languages*. In: *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, Seiten 441–449. IEEE Computer Press, 1999.

[Eur99] European Organization for the Safety of Air Navigation: *Base of Aircraft Data*. http://www.eurocontrol.fr/projects/bada, 1999.

[FA97] Frühwirth, T. und S. Abdennadher: *Constraint-Programmierung*. Springer, 1997.

[FG03] Franceschini, G. und R. Grossi: *Optimal Worst-Case Operations for Implicit Cache-Oblivious Search Trees*. In: *Proceedings of the 8th International Workshop on Algorithms and Data Structures (WADS '03)*, Seiten 114–126, 2003.

[FG05] Franceschini, G. und V. Geffert: *An In-place Sorting with O(n log n) Comparisons and O(n) Moves*. Journal of the ACM, 52(4):515–537, 2005.

[FGK+00] Fabri, A., G.-J. Giezeman, L. Kettner, S. Schirra und S. Schönherr: *On the Design of CGAL: a Computational Geometry Algorithms Library*. Softw. Pract. Exper., 30(11):1167–1202, 2000.

[FGNS00] Forlizzi, L., R. H. Güting, E. Nardelli und M. Schneider-Lem: *A Data Model and Data Sructures for Moving Objects Databases*. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Seiten 319–330. ACM Press, 2000.

[Fis05] Fischer, G.: *Lineare Algebra*. Vieweg, Braunschweig / Wiesbaden, 15. Auflage, 2005.

[Flo64] Floyd, R. W.: *Algorithm 245: Treesort*. Communications of the ACM, 7(12):701, Dezember 1964.

[For99] Forster, O.: *Analysis 2*. Vieweg, Braunschweig, 5. Auflage, 1999.

[For04] Forster, O.: *Analysis 1*. Vieweg, Braunschweig, 7. Auflage, 2004.

[Fus00] Fussel, M. L.: *Double Dispatching in Java*. http://www.chimu.com/publications/short/javaDoubleDispatching.html, ChiMu Corporation, 2000.

[Füh01] Führer, L.: *Kubische Gleichungen und die widerwillige Entdeckung der komplexen Zahlen: Zwei Beispiele zur historisch-genetischen Methode*. Praktische Mathematik, 43(2):57–67, 2001.

[GBE+00] Güting, R. H., M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider und M. Vazirgiannis: *A Foundation for Representing and Quering Moving Objects*. ACM Transactions on Database Systems, 25(1):1–42, March 2000.

[Gee04] Geerts, F.: *Moving Objects and Their Equations of Motion*. In: *Proceedings of the 1st International Symposium on Applications of Constraint Databases (CDB'04)*, 2004.

[GH93] Graf, T. und K. Hinrichs: *A Plane-Sweep Algorithm for the All-Nearest-Neighbors Problem for a Set of Convex Planar Objects.* In: *Proceedings of the 3rd Workshop on Algorithms and Data Structures (WADS '93)*, Seiten 349–360. Springer-Verlag, 1993.

[GHJV95] Gamma, E., R. Helm, R. Johnson und J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, New York, 1995.

[GK99] Guibas, L. J. und M. I. Karavelas: *Interval Methods for Kinetic Simulations.* In: *Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SoCG '99)*, Seiten 255–264. ACM Press, 1999.

[GK01] Geffert, V. und J. Kollar: *Linear-Time In-Place Selection in $\epsilon \cdot n$ Element Moves.* Technischer Bericht, P. J. Safarik University, 2001.

[GKP00] Geffert, V., J. Katajainen und T. Pasanen: *Asymptotically Efficient In-Place Merging.* Theoretical Computer Science, 237(1–2):159–181, April 2000.

[GKR04] Guibas, L., M. I. Karavelas und D. Russel: *A computational framework for handling motion.* In: *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX '04)*, 2004.

[GLRS00] Grumbach, S., Z. Lacroix, P. Rigaux und L. Segoufin: *Optimization Techniques.* In: Kuper, G. M., L. Libkin und J. Paredaens (Herausgeber): *Constraint Databases*, Seiten 319–334. Springer, 2000.

[GM97] Gill, B. und B. Maddock: *PHARE: Prediction of Optimal 4D Trajectories in the Presence of Time.* European Organization for the Safety of Air Navigation, http://www.eurocontrol.int/phare/gallery/content/public/documents/97-70-09efms_4d_trajectory_prediction.pdf, 1997.

[GNRZ02] Guibas, L., A. Nguyen, D. Russel und L. Zhang: *Collision Detection for Deforming Necklaces.* In: *Proceedings of the 18th Annual ACM Symposium on Computational Geometry (SoCG '02)*, Seiten 33–42. ACM Press, 2002.

[GO05] Goodman, T. und B. Ong: *Shape Preserving Interpolation by Splines using Vector Subdivision.* Advances in Computational Mathematics, 22:49–77, 2005.

[Goo02] Goodman, T.: *Shape Preserving Interpolation by Curves.* In: *Algorithms for Approximation IV*, Seiten 24–35. University of Huddersfield, 2002.

[Gra72] Graham, R. L.: *An Effcient Algorithm for Determining the Convex Hull of a Finite Planar Set.* Information Processing Letters, 1:132–133, 1972.

[GRS98] Grumbach, S., P. Rigaux und L. Segoufin: *The DEDALE system for complex spatial queries.* In: *Proceedings of the 1998 ACM SIGMOD international Conference on Management of Data*, Seiten 213–224. ACM Press, 1998.

[GS95] Güting, R. H. und M. Schneider: *Realm-Based Spatial Data Types: The ROSE Algebra.* VLDB Journal, 4(2):243–286, 1995.

[Gui98] Guibas, L. J.: *Kinetic Data Structures: a State of the Art Report.* In: *Proceedings of the 3rd Workshop on the Algorithmic Foundations of Robotics*, Seiten 191–209. A. K. Peters, Ltd., 1998.

[Güt93] Güting, R. H.: *Second-Order Signature: A Tool for Specifying Data Models, Query Processing, and Optimization.* In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Seiten 277–286, Washington, May 1993.

[HAFG95] Hayward, V., S. Aubry, A. Foisy und Y. Ghallab: *Efficient Collision Prediction Among Many Moving Objects.* International Journal of Robotic Research, 14(10):129–143, 1995.

[HE00] Hornsby, K. und M. Egenhofer: *Identity-based Change: a Foundation for Spatio-Temporal Knowledge Representation.* International Journal of Geographical Information Science, 14(3):207–224, 2000.

[Hei30] Heisenberg, W.: *Die physikalischen Prinzipien der Quantentheorie.* Spektrum Akademischer Verlag, 1930.

[HHKR95] Hammer, R., M. Hocks, U. Kulisch und D. Ratz: *C++ Toolbox for Verified Computing I. Basic Numerical Problems. Theory, Algorithms, and Programs C++.* Springer, 1995.

[HL89] Hoschek, J. und D. Lasser: *Grundlagen der geometrischen Datenverarbeitung.* Teubner, Stuttgart, 1989.

[HL05] Hillier, F. S. und G. J. Lieberman: *Introduction to Operations Research.* Holden-Day, Inc., San Francisco, CA, USA, 8. Auflage, 2005.

[Hoa62] Hoare, C.: *Quicksort.* Computer Journal, 5(1):10–15, 1962.

[HTKR05] Höpfner, H., C. Türker und B. König-Ries: *Mobile Datenbanken und Informationssysteme.* dpunkt Verlag, 2005.

[JOW+02] Juang, P., H. Oki, Y. Wang, M. Martonosi, L. S. Peh und D. Rubenstein: *Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet.* In: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '02)*, Seiten 96–107. ACM Press, 2002.

[Kah91] Kahan, S.: *A Model for Data in Motion*. In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC '91)*, Seiten 265–277. ACM Press, 1991.

[Kap94] Kapoor, S.: *Dynamic Maintenance of Maxima of 2-d Point Sets*. In: *Proceedings of the 10th Annual ACM Symposium on Computational Geometry (SoCG '94)*, Seiten 140–149, New York, NY, USA, 1994. ACM Press.

[KDM05] Khemapech, I., I. Duncan, und A. Miller: *A Survey of Wireless Sensor Networks Technology*. In: *Proceedings of the 6th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting (PGNET'05)*, 2005.

[Kea96] Kearfott, R.: *Interval Computations: Introduction, Uses, and Resources*. Euromath Bulletin, 1(2):95–112, 1996.

[KKP00] Kuijpers, B., G. M. Kuper und J. Paredaens: *Euclidean Query Languages*. In: Kuper, G. M., L. Libkin und J. Paredaens (Herausgeber): *Constraint Databases*, Seiten 275–291. Springer, 2000.

[KKR90] Kanellakis, P. C., G. M. Kuper und P. Z. Revesz: *Constraint Query Languages*. In: *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '90)*, Seiten 299–313, 1990.

[KLP75] Kung, H. T., F. Luccio und F. P. Preparata: *On Finding the Maxima of a Set of Vectors*. Journal of the ACM, 22(4):469–476, 1975.

[KN89] Kamgar-Parsi, B. und N. Netanyahu: *A Nonparametric Method For Fitting A Straight Line To A Noisy Image*. IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI), 11:998–1001, 1989.

[Knu98a] Knuth, D. E.: *The Art of Computer Programming*, Band 1. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3. Auflage, 1998.

[Knu98b] Knuth, D. E.: *The Art of Computer Programming*, Band 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3. Auflage, 1998.

[Köt04] Kötterheinrich, D.: *Simplifizierung von Trajektorien sich bewegender Objekte*. Diplomarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2004.

[KP99] Katajainen, J. und T. A. Pasanen: *In-place Sorting with Fewer Moves*. Information Processing Letters, 70(1):31–37, 1999.

[KPL99] Kuper, G., J. Paredaens und L. Libkin (Herausgeber): *Constraint Databases*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[KRR02] Kossmann, D., F. Ramsak und S. Rost: *Shooting Stars in the Sky: An Online Algorithm for Skyline Queries.* In: *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02)*, 2002.

[KS93] Katz, M. J. und M. Sharir: *Optimal Slope Selection Via Expanders.* Information Processing Letters, 47(3):115–122, September 1993.

[KSS00] Kirkpatrick, D., J. Snoeyink und B. Speckmann: *Kinetic Collision Detection for Simple Polygons.* In: *Proceedings of the 16th Annual ACM Symposium on Computational Geometry (SoCG '00)*, Seiten 322–330. ACM Press, 2000.

[KT05] Kleinberg, J. und É. Tardos: *Algorithm Design.* Addison-Wesley, Boston, MA, 2005.

[KV03] Kung, H. T. und D. Vlah: *Efficient Location Tracking Using Sensor Networks.* In: *Proceeding of the 2003 IEEE Wireless Communications and Networking Conference*, 2003.

[KY00] Kuchar, J. und L. Yang: *A Review of Conflict Detection and Resolution Modeling Methods.* IEEE Transactions on Intelligent Transportation Systems, 16(4):179–189, 2000.

[Lee83] Lee, D.: *On Finding the Convex Hull of a Simple Polygon.* International Journal of Computing & Information Sciences, 12(2):87–98, 1983.

[Lee96] Lee, D. T.: *Computational Geometry.* ACM Computing Surveys, 28(1):27–31, 1996.

[LFG+03] Lema, J. A. C., L. Forlizzi, R. H. Güting, E. Nardelli und M. Schneider: *Algorithms for Moving Objects Databases.* The Computer Journal, 46(6):680–712, 2003.

[LHY+04] Lazaridis, I., Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. V. Kalashnikov und W. Yang: *QUASAR: Quality Aware Sensing Architecture.* ACM SIGMOD Record, 33(1):26–31, 2004.

[LLOW91] Lamb, C., G. Landis, J. Orenstein und D. Weinreb: *The ObjectStore Database System.* Communications of the ACM, 34(10):50–63, 1991.

[LM03] Lazaridis, I. und S. Mehrotra: *Capturing Sensor-Generated Time Series with Quality Guarantees.* In: *19th International Conference on Data Engineering (ICDE'03)*, Seiten 429– 440, 2003.

[LM04] Lin, M. C. und D. Manocha: *Collision and Proximity Queries.* In: Goodman, J. E. und J. O'Rourke (Herausgeber): *Handbook of Discrete and Computational Geometry*, Seiten 787–807. Chapman and Hall/CRC Press, New York, 2. Auflage, 2004.

[LMCG97] Lin, M. C., D. Manocha, J. Cohen und S. Gottschalk: *Collision Detection: Algorithms and Applications.* In: *Algorithms for Robotic Motion and Manipulation, WAFR II*, Natick, MA, 1997.

[LMR94] Louis, A. K., P. Maass und A. Rieder: *Wavelets: Theorie und Anwendungen.* Teubner, Stuttgart, 1994.

[Mat91a] Matoušek, J.: *Computing dominances in E_n (short communication).* Information Processing Letters, 38(5):277–278, 1991.

[Mat91b] Matoušek, J.: *Randomized Optimal Algorithm for Slope Selection.* Information Processing Letters, 39(4):183–187, 1991.

[MCP+02] Mainwaring, A., D. Culler, J. Polastre, R. Szewczyk und J. Anderson: *Wireless Sensor Networks for Habitat Monitoring.* In: *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '02)*, Seiten 88–97, New York, NY, USA, 2002. ACM Press.

[Md03] Meratnia, N. und R. A. de By: *Trajectory Representation in Location-Based Services: Problems and Solution.* In: *Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW '03)*, Seiten 18–24, 2003.

[Meg83] Megiddo, N.: *Applying Parallel Computation Algorithms in the Design of Serial Algorithms.* Journal of the ACM, 30(4):852–865, Oktober 1983.

[Mer05] Meratnia, N.: *Towards database support for moving object data.* Doktorarbeit, University of Twente, 2005.

[Mey93] Meyer, Y.: *Wavelets: Algorithms & Applications.* SIAM Press, Philadelphia, 1993.

[MMN98] Matoušek, J., D. M. Mount und N. S. Nethanyahu: *Efficient Randomized Algorithms for the Repeated Median Line Estimator.* Algorithmica, 20(2):136–150, Februar 1998.

[MN90] Mehlhorn, K. und S. Näher: *Dynamic Fractional Cascading.* Algorithmica, 5:215–241, 1990.

[MN91] Mount, D. und N. Netanyahu: *Computationally Efficient Algorithms for a Robust Line Estimator.* Technischer Bericht CS-TR-2816, Maryland University, College Park, 1991.

[MN93] Mount, D. und N. Netanyahu: *Computationally Efficient Algorithms for High-Dimensional Robust Estimators.* Graphical Models and Image Processing, 56(4):289–303, 1993.

[MNP+04] Mount, D. M., N. S. Netanyahu, C. D. Piatko, R. Silverman und A. Y. Wu: *A Computational Framework for Incremental Motion.* In: *Proceedings of the 20th Annual Symposium on Computational geometry (SoCG '04)*, Seiten 200–209. ACM Press, 2004.

[MR95] Motwani, R. und P. Raghavan: *Randomized Algorithms.* Cambridge University Press, New York, NY, USA, 1995.

[MSI02] Mokhtar, H., J. Su und O. Ibarra: *On Moving Object Queries.* In: *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '02)*, Seiten 188–198. ACM Press, 2002.

[MSS03] Meyer, U., P. Sanders und J. Sibeyn (Herausgeber): *Algorithms for Memory Hierarchies*, Band 2625. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[MU84] Mannila, H. und E. Ukkonen: *A Simple Linear-time Algorithm for In Situ Merging.* Information Processing Letters, 18(4):203–208, 1984.

[Mun86] Munro, J. I.: *An Implicit Data Structure Supporting Insertion, Deletion, and Search in O(log n) Time.* Journal of Computer and System Sciences, 33(1):66–74, 1986.

[NH93] Nievergelt, J. und K. Hinrichs: *Algorithms and Data Structures.* Prentice Hall Inc., 1993.

[NPI+05] Nuic, A., C. Poinsot, M.-G. Iagaru, F. A. N. Eduardo Gallo und C. Querejeta: *Advanced Aircraft Performance Modeling for ATM: Enhancements to the BADA model.* In: *Proceedings of the 24th Digital Avionics System Conference*, 2005.

[Nür78] Nürnberger, G.: *Approximation by Spline Functions.* Springer, New York, 1978.

[OSS01] Ottmann, T., S. Schuierer und S. Soundaralakshmi: *Enumerating Extreme Points in Higher Dimensions.* Nordic Journal of Computing, 8(2):179–192, 2001.

[OvL81] Overmars, M. und J. van Leeuwen: *Maintenance of Configurations in the Plane.* Journal of Computer and System Sciences, volume 23:166–204, 1981.

[Pau87] Paulson, L.: *Logic and Computation: Interactive Proof with Cambridge LCF.* Cambridge University Press, 1987.

[Peñ99] Peña, J. M. (Herausgeber): *Shape Preserving Representations in Computer-Aided Geometric Design*. Nova Science, 1999.

[PH02] Patterson, D. A. und J. L. Hennessy: *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3. Auflage, 2002.

[PJ99] Pfoser, D. und C. S. Jensen: *Capturing the Uncertainty of Moving-Object Representations*. In: *Proceedings of the 6th International Symposium on Advances in Spatial Databases (SSD)*, Band 1651 der Reihe *Lecture Notes in Computer Science*, Seiten 111–132. Springer, Berlin, 1999.

[PJ01] Pfoser, D. und C. S. Jensen: *Querying the Trajectories of On-Line Mobile Objects*. In: *Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE '01)*, Seiten 66–73. ACM Press, 2001.

[PS85] Preparata, F. P. und M. I. Shamos: *Computational Geometry—An Introduction*. Springer-Verlag, New York, 1985.

[PT01] Pfoser, D. und N. Tryfona: *Capturing Fuzziness and Uncertainty of Spatiotemporal Objects*. In: *Proceedings of the 5th East European Conference on Advances in Databases and Information Systems (ADBIS '01)*, Band 2151 der Reihe *Lecture Notes in Computer Science*, Seiten 112 – 126, 2001.

[PTFS05] Papadias, D., Y. Tao, G. Fu und B. Seeger: *Progressive Skyline Computation in Database Systems*. ACM Transactions on Database Systems, 30(1):41–82, 2005.

[Puk03] Puke, I.: *Algorithmen für zeitvariante räumliche Daten mit diskreten Repräsentationen*. Diplomarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2003.

[PW99] Phillips, S. und J. Westbrook: *On-Line Algorithms: Competitive Analysis and Beyond*. In: *Algorithms and Theory of Computation Handbook*, Kapitel 10. CRC Press, 1999.

[PXK+02] Prabhakar, S., Y. Xia, D. V. Kalashnikov, W. G. Aref und S. E. Hambrusch: *Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects*. IEEE Transactions on Computing, 51(10):1124–1140, 2002.

[RHE+04] Roddick, J. F., E. Hoel, M. J. Egenhofer, D. Papadias und B. Salzberg: *Spatial, Temporal and Spatio-Temporal Databases - Hot Issues and Directions for Phd Research*. SIGMOD Rec., 33(2):126–131, 2004.

[RL87] Rousseeuw, P. J. und A. M. Leroy: *Robust Regression and Outlier Detection*. John Wiley & Sons, Inc., New York, NY, USA, 1987.

[Rou84] Rousseeuw, P.: *Least Median of Squares Regression*. Journal of the American Statistical Association, 79:871–880, 1984.

[RSSG03] Rigaux, P., M. Scholl, L. Segoufin und S. Grumbach: *Building a Constraint-based Spatial Database System: Model, Languages, and Implementation*. Information Systems, 28(6):563–595, 2003.

[Sch97] Schwarz, H. R.: *Numerische Mathematik*. Teubner, 1997.

[Sch00] Schirra, S.: *Robustness and Precision Issues in Geometric Computation*. In: Sack, J. R. und J. Urrutia (Herausgeber): *Handbook of Computational Geometry*, Kapitel 14, Seiten 597–632. Elsevier Science Publishers, 2000.

[Sch04] Schmolke, T.: *Erhebung und Verarbeitung von Sensordaten*. Diplomarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2004.

[Sen68] Sen, P. K.: *Estimates of the Regression Coefficient Based on Kendall's tau*. Journal of the American Statistical Association, 63:1379–1389, 1968.

[Sie82] Siegel, A. F.: *Robust Regression Using Repeated Medians*. Biometrika, 69:242–244, 1982.

[SKS05] Silberschatz, A., H. F. Korth und S. Sudarshan: *Database System Concepts*. McGraw-Hill, Inc., New York, NY, USA, 5. Auflage, 2005.

[SM03] Suli, E. und D. F. Mayers.: *An Introduction to Numerical Analysis*. Cambridge University Press, New York, NY, 2003.

[Son05] Sondern, J.: *Nutzung von Bewegungsrestriktionen für Algorithmik in Moving-Objects-Datenbanken*. Diplomarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 2005.

[SOP+04] Szewczyk, R., E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring und D. Estrin: *Habitat Monitoring with Sensor Networks*. Communications of the ACM, 47(6):34–40, 2004.

[SS86] Steele, J. M. und W. L. Steiger: *Algorithms and Complexity for Least Median of Squares Regression*. Discrete Applied Mathematics, 14(1):93–100, 1986.

[SS87a] Salowe, J. S. und W. L. Steiger: *Stable Unmerging in Linear Time and Constant Space*. Information Processing Letters, 25(5):285–294, 1987.

[SS87b] Souvaine, D. L. und J. M. Steele: *Time- and Space-efficient Algorithms for Least Median of Squares Regression*. Journal of the American Statistical Association, 82(399):794–801, 1987.

[SS93] Shafer, L. und W. L. Steiger: *Randomizing Optimal Geometric Algorithms*. In: *Proceedings of the 5th Canadian Conference on Compututational Geometry (CCCG '93)*, Seiten 133–138, 1993.

[SS96] Sweldens, W. und P. Schröder: *Building Your Own Wavelets at Home*. In: *Wavelets in Computer Graphics*, Seiten 15–87. ACM SIGGRAPH Course notes, 1996.

[SS04] Sharifzadeh, M. und C. Shahabi: *Supporting Spatial Aggregation in Sensor Network Databases*. In: *Proceedings of the 12th Annual ACM International Workshop on Geographic information systems (ACM GIS '04)*, Seiten 166–175, New York, NY, USA, 2004. ACM Press.

[ST99] Salzberg, B. und V. J. Tsotras: *Comparison of Access Methods for Time-Evolving Data*. ACM Computing Surveys, 31(2):158–221, June 1999.

[Sto99] Stoer, J.: *Numerische Mathematik 1*. Springer, 1999.

[Str00] Stroustroup, B.: *The C++ Programming Language*. Addison Wesley, 3. Auflage, 2000.

[SW95] Sistla, A. P. und O. Wolfson: *Temporal Conditions and Integrity Constraints in Active Database Systems*. In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, Band 24.2 der Reihe *SIGMOD Record*, Seiten 269–280. ACM Press, June 1995.

[SWCD97] Sistla, A. P., O. Wolfson, S. Chamberlain und S. Dao: *Modeling and Querying Moving Objects*. In: *Proceedings of the 13th International Conference on Data Engineering Engineering (ICDE '97)*, Seiten 422–432. IEEE Computer Society, 1997.

[Swe94] Sweldens, W.: *The Construction and Applications of Wavelets in Numerical Analysis*. Doktorarbeit, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 1994.

[SXI01] Su, J., H. Xu und O. H. Ibarra: *Moving Objects: Logical Relationships and Queries*. In: *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD '01)*, Band 2121 der Reihe *Lecture Notes in Computer Science*. Springer, Berlin, 2001.

[TEO01] Tan, K.-L., P.-K. Eng und B. C. Ooi: *Efficient Progressive Skyline Computation*. In: *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*, Seiten 301–310, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[The50] Theil, H.: *A Rank-invariant Method of Linear and Polynomial Regression Analysis.* Nederlandse Akademie Wetenchappen Series, A(53):386–392, 521–525, 1397–1412, 1950.

[TWHC04] Trajcevski, G., O. Wolfson, K. Hinrichs und S. Chamberlain: *Managing Uncertainty in Moving Objects Databases.* ACM Transactions on Database Systems, 29(3):463–507, 2004.

[Vah05] Vahrenhold, J.: *Line-Segment Intersection Made In-Place.* In: *Proceedings of the 9th International Workshop on Algorithms and Data Structures (WADS '05)*, Band 3608 der Reihe *Lecture Notes in Computer Science*, Seiten 146–157, Berlin, 2005. Springer.

[Vah06] Vahrenhold, J.: *An In-place Algorithm for Klee's Measure Problem in Two Dimensions.* Manuskript, 2006.

[Val01] Valiela, I.: *Doing Science: Design, Analysis and Communication of Scientific Research.* Oxford University Press, 2001.

[VCdSdM03] Vieira, M., C. Coelho, D. da Silva und J. da Mata: *Survey on Wireless Sensor Network Devices.* In: *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA '03)*, Band 1, Seiten 537 – 544, 2003.

[Ven02] Venkataraman, P.: *Applied Optimization with MATLAB Programming.* John Wiley & Sons, Inc., 2. Auflage, 2002.

[Voi98] Voigtmann, A.: *An Object-Oriented Database Kernel for Spatio-Temporal Geo-Applications.* Doktorarbeit, Institut für Informatik, Westfälische Wilhelms-Universität Münster, 1998.

[Vui03] Vuilleumier, S.: *Dispersal Modelling.* Doktorarbeit, EPFL, Lausanne, 2003.

[VW01] Vazirgiannis, M. und O. Wolfson: *A Spatiotemporal Model and Language for Moving Objects on Road Networks.* In: *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD '01)*, Band 2121 der Reihe *Lecture Notes in Computer Science*, Seiten 20–35. Springer, Berlin, 2001.

[WCD+98] Wolfson, O., S. Chamberlain, S. Dao, L. Jiang und G. Mendez: *Cost and Imprecision in Modeling the Position of Moving Objects.* In: *Proceedings of the 14th International Conference on Data Engineering (ICDE '98)*, Seiten 588–596. IEEE Computer Society, 1998.

[Wer92] Werner, J.: *Numerische Mathematik.* Vieweg, Braunschweig, 1992.

[Wil64] Williams, J. W. J.: *Algorithm 232: Heapsort.* Communications of the ACM, 7(6):347–348, Juni 1964.

[WJS+99] Wolfson, O., L. Jiang, A. P. Sistla, S. Chamberlain, N. Rishe und M. Deng: *Databases for Tracking Mobile Units in Real Time*. In: *Proceedings of the 7th International Conference on Database Theory (ICDT '99)*, Band 1540 der Reihe *Lecture Notes in Computer Science*, Seiten 169–186. Springer, Berlin, 1999.

[Wor94] Worboys, M. F.: *A Unified Model for Spatial and Temporal Information*. The Computer Journal, 37(1):25–34, 1994.

[Wor95] Worboys, M. F.: *GIS: A Computing Perspective*. Taylor & Francis, 1995.

[WSX+99] Wolfson, O., A. P. Sistla, B. Xu, J. Zhou und S. Chamberlain: *DOMINO: Databases fOr MovINg Objects tracking*. In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, Band 28.2 der Reihe *SIGMOD Record*, Seiten 547–549. ACM Press, June 1999.

[WXCJ98] Wolfson, O., B. Xu, S. C. Chamberlain und L. Jiang: *Moving Objects Databases: Issues and Solutions*. In: *Proceedings of the 10th International Conference on Scientific and Statistical Database Management (SSDBM '98)*, Seiten 111–122. IEEE Computer Society, 1998.

[YdC95] Yeh, T. und B. de Cambray: *Modeling Highly Variable Spatio-Temporal Data*. In: *Proceedings of the 6th Australian Database Conference (ADC '95)*, Band 17.2 der Reihe *Australian Computer Science Communications*, Seiten 221–230, 1995.

Lebenslauf **September 2006**

Persönliche Daten

Name:	Henrik Blunck
Geburtstag:	31. Januar 1976
Geburtsort:	Lübeck, Deutschland
Nationalität:	deutsch
Familienstand:	ledig
Eltern:	Hans-Joachim Blunck und Beate Blunck, geb. von Lojewski

Ausbildung

seit 07/2002	WWU Münster: Studien zur Promotion in der Informatik und Beginn der Dissertation bei Prof. Dr. Klaus H. Hinrichs
11.07.2002	Diplom der Mathematik
03.09. 1999	Vordiplom der Mathematik
04/1997 – 07/2002	WWU Münster: Studium der Mathematik mit Anwendungsfach Informatik
09.06.1995	Abitur
08/1986 – 06/1995	Gymnasium Thomas-Mann-Schule, Lübeck
08/1982 – 06/1986	Wulfsdorfer Grundschule, Lübeck

Tätigkeiten

seit 08/2002	Institut für Informatik, WWU Münster: Wissenschaftlicher Mitarbeiter
08/2001 – 07/2002	Institute für Mathematik und angewandte Mathematik, WWU Münster: studentische Hilfskraft für Lehrtätigkeiten
10/1999 – 03/2001	Institut für Informatik, WWU Münster: studentische Hilfskraft für Lehrtätigkeiten
12/1995 – 12/1996	Institut für Kinderpsychosomatik, Universitätsklinikum Schleswig Holstein in Lübeck: Zivildienst