Dissertation

On Distance and Sorting of the Double Cut-and-Join and the Inversion-*indel* Model

Zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

an der Technischen Fakultät
der Universität Bielefeld

vorgelegt von

Eyla Willing

April 2018

Gedruckt auf alterungsbeständigem Papier ISO 9706

Zusammenfassung

In der vergleichenden Genomik werden zwei oder mehrere Genome hinsichtlich ihres Verwandtschaftsgrades verglichen. Das Ziel dieser Arbeit ist die Erforschung von mathematischen Modellen, die zum einen die evolutionäre *Distanz*, zum anderen die evolutionären Vorgänge zwischen zwei Genomen bestimmen können.

Neben Methoden, welche auf einer niedrigen Ebene, z. B. den Basen(paarungen), ansetzen, sind auch abstraktere Modelle, die auf einzelnen Genen oder noch größeren Abschnitten Genome vergleichen, etabliert. Handelt es sich auf niedrigerer Ebene um einzelne Basen, die eingefügt, gelöscht oder ersetzt werden, sind es auf höherer Ebene beispielsweise ganze Gene. Auf höherer Ebene können Ergebnisse sogenannter Umordnungsprozesse (*genome rearrangements*) beobachtet werden, welche in einem *Sortierszenario* beschrieben werden. Im Vergleich eines Genoms mit einem anderen können dies unter anderem Inversionen, Translokationen, aber auch Einfügungen oder Löschungen von großen Bereichen sein. Ein bekanntes Modell ist das *Inversionsmodell*, welches den Verwandtschaftsgrad zweier Genome ausschließlich durch Inversionen bestimmt. Ein weiteres ist das *double cut-and-join (DCJ)* Modell, welches neben Inversionen auch Translokationen, Chromosomenfusionen, bzw. -fissionen, sowie Integration und Extraktion von kleinen zirkulären Trägern erlaubt. Die Distanz ist hierbei die Anzahl Zwischenschritte eines Sortierszenarios von geringster Länge.

Diese Dissertation ist in zwei Teile gegliedert. Der erste Teil beschäftigt sich mit dem zufälligen Ziehen eines Sortierszenarios innerhalb des DCJ-Modells. Neben einigen naiven Ansätzen interessieren wir uns im Wesentlichen dafür, jedes Szenario mit gleicher Wahrscheinlichkeit, also uniform verteilt, zu ziehen. Hierfür wird nicht nur der gesamte Sortierraum betrachtet, sondern auch Maßnahmen zur effizienten Berechnung aufgezeigt. Der vorgestellte Algorithmus ist in einer Software-suite implementiert und wird hinsichtlich seiner Erzeugung von zufälligen Szenarien evaluiert.

Der zweite Teil der Arbeit beschäftigt sich mit dem *Inversions-indel* Modell. Dieses wenig erforschte Modell erlaubt Inversionen, sowie Einfügungen und Löschungen (kurz *indel*s). Dessen Distanz soll in Abhängigkeit von der DCJ- bzw. der DCJ-*indel*-Distanz wiedergegeben werden. Wir erweitern altbekannte Datenstrukturen des Inversionsmodells um Einfügungen und Löschungen repräsentieren zu können. Hierfür benutzen wir unter anderem Ansätze aus zwei anderen Modellen: Die Erweiterung des DCJ-Modells um indels, sowie die Ermittlung der Abhängigkeit von DCJ- und Inversionsmodell.

Um die minimale Anzahl an Inversionen, Einfügungen und Löschungen zu ermitteln muss beachtet werden, dass durch Inversionen zwei oder mehr getrennte Bereiche, die zur Löschung vorgesehen sind, verschmolzen werden. Diese können sodann in einem einzigen Schritt gelöscht werden. Ähnlich verhält es sich mit Einfügungen. Zunächst betrachten wir Instanzen in denen die DCJ-*indel*-Distanz und die Inversions-*indel*-Distanz identisch sind. Im Weiteren gehen wir dazu über, schwierige Instanzen, d.h. jene die mehr Schritte benötigen als die DCJ(-*indel*)-Distanz, zu berechnen. Zu diesen Zweck müssen die unterschiedlichen Eigenschaften der Instanzen und deren Auswirkungen ausgemacht werden. Durch geschickte Reduzierung des Lösungsraums gelangen wir zu einer Menge von Basisfällen, welche wir durch erschöpfende Aufzählung lösen können. Insgesamt bieten die unternommenen Schritte nicht nur die Lösung der Inversions-*indel* Distanz in Abhängigkeit zur DCJ-*indel* Distanz, sondern auch eine Möglichkeit des Sortierens.

Die Suche nach einer exakten Lösung für das Distanz- und das Sortierproblem im Inversions-*indel* Modell blieb lange unbeantwortet. Der Hauptbeitrag dieser Arbeit liegt darin diese zwei Fragen zu klären.

Abstract

In comparative genomics two or more genomes are compared with regard to their evolutionary relationship. The aim of this thesis is to study mathematical models that determine, for one, the evolutionary *distance* and, for another, the evolutionary events occurring between the divergence of two genomes.

Besides methods applied to a low level of abstraction, that, for instance, count insertions, deletions or substitutions of one or a few DNA bases, there are more abstract, well-established models that compare on gene level or consider even larger regions. In this context, genome rearrangement processes can be witnessed between two genomes. These large-scale modifications can involve, amongst others, inversions, translocations, as well as insertions and deletions (indels) of large regions. A *sorting scenario* starts with a source genome and shows which changes this genome undergoes until the target genome is obtained. The distance of two genomes is given by the length of a sorting scenario that has a minimum number of steps.

A well-known model is the *inversion* (or *reversal*) *model* that computes the minimum number of inversions between two genomes. Another well-established model is the *double cut-and-join (DCJ) model*, that, besides inversions also allows for translocations, chromosomal fission/fusion and extraction/integration of circular intermediates.

This thesis consists of two main parts. In the first part, the sampling of an optimal DCJ sorting scenario is studied. Apart from some naïve approaches we are mostly interested in sampling each scenario with equal probability, thus to sample *uniformly*. For this, we not only examine the sorting space but also devise measures for efficient computation. The proposed algorithm is integrated into an existing software suite and is evaluated concerning its drawing of random scenarios.

The second part of this thesis studies the *inversion-indel* model. This under-explored model allows for inversions, insertions and deletions. Its distance is given with respect to the DCJ(-*indel*) distance. We extend well-known data structures of the inversion model in order to represent insertions and deletions. We use approaches from two other models: the extension of the DCJ model by indels, as well as the determination of the difference between DCJ and inversion distance.

In order to determine the minimum number of inversions, insertions and deletions, it has to be considered that an inversion may fuse two or more parts that were destined to be deleted. Then this large region can be deleted in one step. The situation is similar for insertions. First, we study instances in which the DCJ-*indel* distance and the inversion-*indel* distance are identical. We proceed with difficult instances, i.e. those that need more steps than used under the DCJ(-*indel*) model. For this, the different properties of the instances and their impact have to be detected. By reduction of the solution space we derive a set of base cases that can be solved by exhaustive analysis. In total, the steps that are undertaken provide not only a solution to the inversion-*indel* distance with respect to the DCJ-*indel* distance, but also provide a way to sort two genomes.

The main contribution of this thesis is resolving two questions in the field of genome rearrangements that had remained unanswered for more than a decade: the inversion-*indel* distance and sorting.

Contents

1 Introduction **1**

1.1 Introduction to Genomes . . . 1

1.1.1 Organisation of Genetic Information . . . 2

1.1.2 Genome Nomenclature . . . 3

1.2 Comparative Genomics . . . 5

1.2.1 Genome Modifications . . . 6

1.2.2 Genome Modification Models . . . 10

1.3 Content and Structure of the Thesis . . . 13

2 Important Data Structures and Models **15**

2.1 Insertions and Deletions . . . 15

2.1.1 Core Genomes and Unique Markers . . . 16

2.1.2 A First Upper Bound to the Distance with Unique Markers . . 17

2.1.3 Extremities, Adjacencies and Labels . . . 19

2.1.4 The Indel Operation on $\mathcal{G}$-Adjacencies . . . 20

2.2 Graph Structures for Representing Genomic Relations . . . 21

2.2.1 Breakpoint Graph . . . 21

2.2.2 Adjacency Graph . . . 23

2.2.3 Master Graph . . . 24

2.2.4 Properties of Cycles . . . 25

2.2.5 A Note on Relations between the Types of Graphs . . . 25

2.3 The Double Cut-and-Join Model . . . 27

2.3.1 The DCJ Operation on Unlabelled $\mathcal{G}$-Adjacencies . . . 27

2.3.2 The Effect of a DCJ Operation on the Adjacency Graph . . . 28

2.3.3 Distance and Sorting . . . 29

3 Uniform Sampling of DCJ Sorting Scenarios 31
3.1 Sampling by Sorting Components Individually 35
3.1.1 Solution Space for DCJ Sorting without Recombinations 35
3.1.2 Bundling Cases with Identical Distance Values 37
3.1.3 Bundling Cases with Identical Change(s) in Distance Value . . 39
3.1.4 Sampling Weights for a Distance-Splitgroup-Pair 43
3.1.5 Uniform Sampling of an Optimal DCJ Operation 44
3.2 Implementation into `UniMoG` 47
3.2.1 Drawing a Huge Random Integer 49
3.2.2 Example Output 50
3.3 Evaluation 53
3.3.1 Overview of Sampling Methods 53
3.3.2 Real Data 53
3.3.3 Artificial Data 62
3.4 Discussion 66

4 DCJ-*indel* Model on Circular Genomes Via DCJ Distance 71
4.1 Generalising the DCJ Model and Distance 72
4.1.1 The DCJ Operation on Labelled $\mathcal{G}$-Adjacencies 73
4.2 DCJ and *indel* Operations on a Labelled Cycle 73
4.3 DCJ Operations on a Pair of Labelled Cycles 76
4.4 Distance 78
4.5 On Sorting With Indels 79
4.5.1 Introducing Insertions from a $\mathcal{B}$-run 80
4.5.2 Optimal Completion 80

5 Inversion-*indel* Distance Problems 87
5.1 Distance Relations 88
5.2 Preliminaries 89
5.2.1 Effect of an Inversion on Cycles 90
5.2.2 Component Groups 92
5.2.3 Component Group Relations 92
5.2.4 Effect of an Inversion on Component Groups 93
5.3 Resolving Unlabelled Good Components 94
5.4 Resolving Labelled Good Components 95
5.4.1 Finding Safe Integrations 95
5.4.2 The Inversion-*indel* Distance on Good Components 98

5.5 Handling $\mathcal{AB}$-Cycles and $\mathcal{AB}$-Component Groups 99
5.6 The Labelled Component Group Tree 100
5.6.1 Resolving Bad Unlabelled Component Groups 106
5.6.2 The Bad Component Group Tree $T_\circ$ 108
5.7 Chapter Summary 112

6 Optimal Tree Covers of $T_\circ$ 113
6.1 Covering Paths and Tree Covers 114
6.1.1 Cost of a Cover Bounded by Individual Subtrees 117
6.2 Properties of $T_\circ$ Influencing the Cost of Optimal Covers 118
6.2.1 Separation Types 119
6.2.2 Links of Subtrees 121
6.2.3 Short Paths 123
6.2.4 Property Set of $T_\circ$ 125
6.3 Heterogeneous Paths 126
6.4 The Residual Tree T_r 135
6.4.1 Residual Trees With One Type of Leaf Labelling 138
6.4.2 Residual Trees With Two Types of Leaf Labelling 139
6.4.3 Residual Trees With Three Types of Leaf Labelling 143
6.4.4 Residual Trees With Four Types of Leaf Labelling 145
6.4.5 Strategy for Reduction in the General Case 146
6.5 Chapter Summary 148

7 Inversion-*indel* Distance and Sorting 151
7.1 The General Inversion-*indel* Distance 151
7.2 Sorting with Inversions, Insertions and Deletions 156
7.2.1 The Residual Tree Cover as Precursor 156
7.2.2 Sorting with Inversions and Deletions 157
7.3 Model Limitations and Extensions 158
7.3.1 Multichromosomal Genomes 158
7.3.2 Co-tailed Genomes 158

8 Final Remarks 161
8.1 Summary 161
8.2 Prospects 162

Bibliography 169

Notations **179**

Appendix **181**

A Additional Examples . . . 181

A.1 Unsafe Covering Paths . . . 181

A.2 Non-Separated Labelled Tree . . . 182

A.3 Separations . . . 182

A.4 Short and Long ε-Branches . . . 185

A.5 Counter-Example for Reduction of $\mathcal{AB}$-Leaves . . . 186

A.6 The Number of Separating Vertices Matters . . . 186

B Bounds to an Optimal Tree Cover With Restraints . . . 188

C Cover Descriptions . . . 189

C.1 Nomenclature . . . 189

C.2 Trees with Two Leaf Types . . . 190

C.3 Trees with Three Leaf Types . . . 192

C.4 Trees with Four Leaf Types . . . 197

D Detailed Sampling Results for sty-stm Comparison of γ-Proteobacteria 202

List of Figures

1.1 The double helix structure of DNA. 2
1.2 Schematic view of an excerpt of the human X chromosome. 4
1.3 Illustration of the representation of marker order on a chromosome. . . 4
1.4 Synteny plot of two Rickettsia genomes along with marker arrangement. 6

2.1 Genome graph of genomes A and B with unique markers. 16
2.2 Genome graph of the corresponding core genomes of genomes A and B. 17
2.3 An optimal operation can group unique markers. 19
2.4 Breakpoint graph $\mathrm{BG}(A, B)$ of genomes A and B. 22
2.5 Breakpoint graph $\mathrm{BG}(B_a, A)$ of genomes B and A. 22
2.6 Adjacency graph $\mathrm{AG}(A, B)$ of genomes A and B. 23
2.7 Recombination of an AA- and a BB-path in the adjacency graph. . . . 28
2.8 Further DCJ operations on components of the adjacency graph. 28
2.9 Circularisation/linearisation in the adjacency graph. 28

3.1 Illustration of a linear time algorithm to sort the adjacency graph. . . 33
3.2 All optimal DCJ operations with probabilities for each next step. . . . 34
3.3 Actual probabilities for each step in the sorting scenario. 35
3.4 Sorted and unsorted components of an adjacency graph. 36
3.5 Possible next steps are weighted by the number of scenarios they leave. 36
3.6 The set of unsorted components partitioned into subsets. 37
3.7 Computing the edge weights for different subsets of $\mathcal{U}$. 38
3.8 Possible cut positions for DCJ operations acting on a single component. 39
3.9 Cut positions for a specific splitgroup in a BB-path. 42
3.10 Splitgroups and operations for one representative $\mathcal{U}_d$. 43
3.11 Sampling an optimal DCJ operation. 45
3.12 Screenshot of the graphical user interface of the software `UniMoG`. . . . 48
3.13 Drawing huge random variables. 49

3.14 Flow diagram of sampling a huge natural number. 50
3.15 Several different sorting scenarios for the same pair of genomes. 52
3.16 Compacted sorting space of the `sty-stm` adjacency graph. 55
3.17 Distribution of `sty-stm` scenarios for vertex sampling. 56
3.18 Distribution of `sty-stm` scenarios for edge sampling. 57
3.19 Distribution of `sty-stm` scenarios for split sampling. 58
3.20 Comparison of four different sampling methods for γ-proteobacteria. . 60
3.21 Boxplots of all pairwise comparisons of real genomes for all four methods. 61
3.22 Comparison of four different sampling methods for the `tiny` example. . 63
3.23 Boxplots of all four methods for the `tiny` example. 64
3.24 Comparison of four different sampling methods for the `interm` example. 65
3.25 Boxplots of all four methods for the `interm` example. 66

4.1 Possible double cut-and-join combinations for labelled adjacencies. . . 74
4.2 Accumulating all labels of an $\mathcal{A}$-cycle by optimal DCJ operations. . . . 74
4.3 Runs in a cycle of a relational diagram and a master graph. 75
4.4 Merging of runs reduces overall number of necessary indel operations. . 75
4.5 Merging of runs reduces overall number of necessary indel operations. . 76
4.6 Schematic sorting of genome A into B with rearrangements and indels. 79
4.7 Introducing an accumulated $\mathcal{B}$-run in genome B. 80
4.8 Two genomes with the singletons and positions for optimal integration. 84
4.9 Resulting genomes after an optimally integrated completion. 85

5.1 Overview of inversion-indel distance subproblems. 88
5.2 Assigned orientation of adjacency edges in the relational diagram. . . . 90
5.3 An optimal but not safe integration of a singleton to genome A. 96
5.4 Illustration of how to find a safe integration. 97
5.5 Circular display of a breakpoint graph and its component group tree. . 103
5.6 The relation of inversion, DCJ and indel operations. 107
5.7 Initial labelled component group tree of exemplar genomes. 108
5.8 Destroying bad component groups while using only bad nodes. 109
5.9 Destroying bad component groups while using good labelled nodes. . . 109
5.10 Example of how to obtain a bad labelled component group tree. 111

6.1 Trees with the same leaf composition yielding different optimal cover cost.115
6.2 Schematic display of some examples for subtrees and their interrelation. 120
6.3 Schematic display of (pairs of) separations and non-separations. 120
6.4 Three examples of links: Two $\mathcal{B}$-$\{\mathcal{A}, \varepsilon\}$-links and one $\mathcal{B}$-$\mathcal{A}$-link. 122

6.5 Counter-example for assigning all $\mathcal{AB}$-leaves to be $\mathcal{A}$-leaves. 123
6.6 Short ε-branches are not always safe. 124
6.7 Example of a solo ε-leaf. 124
6.8 Each instance of the same leaf composition has to be analysed individually. 127
6.9 Replacing heterogeneous $\mathcal{A}$-paths such that two instead of four are used. 129
6.10 Replacing heterogeneous $\mathcal{A}$-paths such that we use one instead of three. 130
6.11 Replacing heterogeneous ε-paths such that two instead of four are used. 132
6.12 Replacing heterogeneous ε-paths such that one instead of three is used. 133
6.13 Counter-example for using homogeneous $\mathcal{AB}$-paths. 133
6.14 Replacing heterogeneous $\mathcal{AB}$-paths until only three of this type are used. 134
6.15 The ε-subtree cannot be optimally reduced to a single leaf. 135

7.1 Visualisation of steps for the inversion-*indel* distance computation. . . 153
7.2 Visualisation of steps for computing the cost of an optimal tree cover. 154

A.1 A path produces a new bad leaf and is thus unsafe. 181
A.2 How a labelled subtree's separation is preserved. 182
A.3 Full separations for different numbers of leaf types. 183
A.4 Trees that have three or four subtrees of which not all are separated. . 183
A.5 Different trees whose ε- and $\{\mathcal{A}, \mathcal{B}\}$-subtrees are separated. 183
A.6 Three trees whose ε- and $\mathcal{A}$-subtrees but not $\mathcal{B}$-subtrees are separated. 184
A.7 Some schematic examples for four subtrees and their interrelation. . . 184
A.8 Three examples of trees with three ε-leaves on branches of various lengths. 185
A.9 A tree with three $\mathcal{AB}$-leaves has no optimal cover that can be reduced. 186
A.10 Examples of homo- and heterogeneous paths in covers for $\mathcal{L} = (2, 2, 0, 0)$. 187
A.11 Examples of homo- and heterogeneous paths in covers for $\mathcal{L} = (2, 0, 2, 0)$. 187
D.12 Adjacency graph for `sty-stm` comparison. 202
D.13 Sampling results for all four methods for the `sty-stm` comparison. . . . 202

List of Tables

1.1 An excerpt from the human X chromosome map. 3
1.2 Different models and implied genome modification. 13

2.1 Impact of different DCJ operations on the DCJ distance. 30

3.1 Selected species for a comparison of γ-proteobacteria. 54
3.2 Marker orders for the selected γ-proteobacteria species. 54
3.3 Adjacency graph characteristics of pairwise γ-proteobacteria comparisons. 54

4.1 Impact of the merging of different cycles on the DCJ-*indel* distance. . 77

5.1 Impact of the merging of different cycles on the DCJ-*indel* distance. . 91

6.1 List of costs of optimal covers of T_r by leaf composition. 149

A.2 A list of separations and non-separations of the subtrees of Figure A.7. 184

DNA is like a computer program but far, far more advanced than any software ever created.
Bill Gates

Chapter 1 Introduction

It has long been known that the macromolecule *deoxyribonucleic acid (DNA)* is the carrier of hereditary information. By analysing DNA, we wish to gain information about individuals, diseases or pedigrees. This information could then be used, for example, in producing vermin resistant agricultural crops, identifying the relation of species or individuals, determining origins or cures for certain diseases, or learning about the mechanisms changing genetic information.

In the following we give a brief summary of the history of DNA discovery, genome structure and comparative genomics.

1.1 Introduction to Genomes

The macromolecule that is known today as *DNA (deoxyribonucleic acid)* and is a carrier of genetic information was first noticed in 1869 by Friedrich Miescher who then published the findings in 1871 [73]. The composition of the macromolecule was further analysed, and it was found that there were nucleotides whose components are phosphate-sugar-bases. Furthermore, the sugars were determined to be 2-deoxyriboses, and the bases were identified as the purines *adenine* (`A`) and *guanine* (`G`) and the pyrimidines *cytosine* (`C`), and *thymine* (`T`). A detailed timeline on the subsequent discovery and isolation of parts of DNA is given in a review of the tetranucleotide theory in [56]. Amongst others, Phoebus Levene contributed fundamentally to this theory (e.g. [65,66]). Yet, the layout of the macromolecule and the purpose were not known.

Frederick Griffith first observed the exchange of genetic information between bacteria in 1928 without narrowing down which part of the cell debris served as carrier [51]. In 1944, Avery *et al.* [2] conducted subsequent experiments on different (classes of) molecules and found only the DNA molecule could effectuate the transformation observed by Griffith. From this result, scientists deduced that DNA is the carrier of genetic information [2] despite the alleged simplicity of the molecule.

Meanwhile, Chargaff analysed nucleic acids [30] and found that the molar amount of `A` was that of `T` and also the molar amount of `G` equalled the amount of `C`. The individual amounts were approximately the same across different tissue types but the ratio of `A+T` to `G+C` was shown to differ between species.

All doubt on the purpose of DNA was eradicated with the experiments of Hershey and Chase in 1952 [58] that proved that indeed the deoxyribonucleic acid is the genetic material.

Despite knowing individual parts of the macromolecule, the layout of DNA in space was not yet determined. Propositions existed for the organisation of the molecules of DNA in space, yet Watson and Crick [85] were the first to construct a model that has two chains forming a double helical structure by coiling around the same axis. Their model was based on prior research of Rosalind Franklin. They postulated the backbone of the helix to be sugar-phosphates joined by its 3',5'-linkages, and that the two chains run in opposite directions [85]. The bases `A`, `C`, `G` and `T` that are attached to the sugars of the backbone form hydrogen bonds with the opposing base of the other chain (in the pairing postulated by Chargaff) thus holding together the double helix as is schematically depicted in Figure 1.1 that shows a small fraction of the macromolecule.

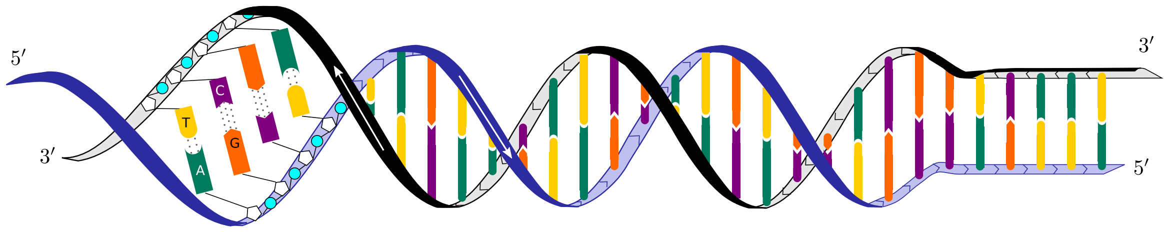


Figure 1.1: The double helix structure with sugar-phosphate backbone and base pairs `A-T` and `G-C` bonded by hydrogen bonds as postulated by Watson and Crick. More detailed view to the left and uncoiled layout towards the right-hand side. Arrows indicate the directions of the chains, white pentagons represent sugars and cyan circles represent phosphates.

1.1.1 Organisation of Genetic Information

A *genome* of a species is the entirety of the nucleic genetic/hereditary information. It is organised in one or multiple *chromosome(s)*. For example, for the human genome we consider a chromosome set of 23 chromosomes, while most bacteria have one or a few chromosomes. Each chromosome is present in a cell as a separate DNA double helix which can be either closed in a ring (circular chromosome) or not (linear chromosome). Usually, all chromosomes of one species are either linear, as in humans for example, or circular, as in most bacteria for example, but species in which both types of chromosomes co-exist are also known [29,84].

The bases of one strand are complemented by the bases in the other strand in the correct base pairing. Due to the nature of the double helix, when reading the double-stranded DNA, the *leading strand* refers to the strand from the $5'$-end towards the $3'$-end (in reading direction) and the *lagging strand* lies on the opposite side. Note that either of the strands can be the leading strand and the other the respective lagging strand, as both have a $3'$- as well as a $5'$-end. The leading strand constitutes the *reverse complement* of the lagging strand.

1.1.2 Genome Nomenclature

In this thesis, instead of at nucleotide level, we consider genomes at a more abstract level. This may be, for example, large stretches of bases that form functional units called *genes*. Genes comprise the information of one strand only and are thus considered to lay on either the leading or lagging strand. Known genes on chromosomes are annotated with position and strand, for example details of a fragment of the human X-chromosome (Annotation Release 106) taken from the NCBI Map Viewer[1] can be seen in Table 1.1.

Table 1.1: An excerpt from the human X chromosome[1]. The first and second column give the start and stop location of the gene, respectively. The symbol column gives the gene's name. The orientation (+ for leading, - for lagging strand) is given in column "O". The last column gives a brief description of the gene.

Start	Stop	Symbol	O ..	Description
:				
129980302	130058083	BCORL1	+	BCL6 corepressor-like 1
130064920	130110713	ELF4	-	E74-like factor 4 (ets domain trans. factor)
130129362	130165887	AIFM1	-	apoptosis-inducing factor, mitochondrion-assoc.,1
130171799	130184870	RAB33A	+	RAB33A, member RAS oncogene family
130202699	130268948	ZNF280C	-	zinc finger protein 280C
:				

A graphical representation of this section of the chromosome with its two strands is shown in Figure 1.2. It hints at the relative position and length of the genes.

[1] `http://www.ncbi.nlm.nih.gov/projects/mapview/maps.cgi?TAXID=9606&CHR=X&MAPS=genes[129980302.00%3A130268948.00]&CMD=TXT#1`, 26.03.2015

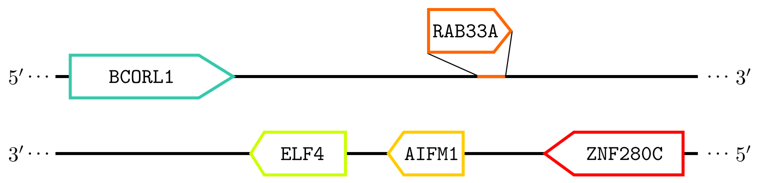


Figure 1.2: Schematic view of the excerpt from the human X chromosome given in Table 1.1. Genes `BCORL1` and `RAB33A` lie on one strand and genes `ELF4`, `AIFM1` and `ZNF280C` lie on the other strand.

Gene representation. Comparing two or more genomes may reveal that genes or even larger regions of a chromosome are common to more than one genome. These syntenic blocks will be called *markers* from now on, and each occurrence of a marker is indicated by the same identifier. The *copy number* of a marker then refers to the number of occurrences of that marker in a specific genome. The identifiers need to be unique names, for example $1, 2, 3, 4$ and 5. Furthermore, markers opposite of the reading direction of the leading strand (indicated by a minus "`-`" in column `O` of Table 1.1) are assigned a negative sign (markers $2, 3$ and 5). Plus-signs (in reading direction) are usually omitted (see Figure 1.3).

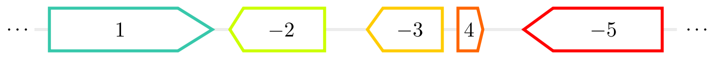


Figure 1.3: Illustration of the representation of marker order on a chromosome corresponding to the previous example (Figure 1.2). Markers have integers as unique identifiers, positive on the leading strand and negative on the lagging strand.

Along a stretch of DNA, functional units may be identified on either of the strands. Hence, a marker may be identified on one strand, and on the opposing strand overlapping it, another marker may be identified (there are many examples found in the human X-chromosome alone). How this is dealt with in a representation such as shown in Figure 1.3 depends on the processing of data conducted in each individual study.

Chromosome representation. A chromosome is represented as the tuple of markers in order as read from the chromosome and with respective signs enclosed by parentheses. For indicating the two ends of a linear chromosome (usually referred to as *telomeres*), we use the *cap*-symbol $\circ$ as auxiliary first and last symbol of the chromosome. The chromosome from Figure 1.3, for example, is represented by the tuple $(\circ, \dots, 1, -2, -3, 4, -5, \dots, \circ)$, where the dots represent the sections on the chromosome that are not shown. Since the direction of reading a chromosome is optional, as long as we take care of the respective signs, for the above example $(\circ, \dots, 5, -4, 3, 2, -1, \dots, \circ)$ is another equivalent representation (imagine the original Figure 1.2 turned by 180°).

Circular chromosomes are simply represented by a tuple of their markers, an example chromosome being $(1, -2, 3)$. This is because, in circular chromosomes, there is no restriction as to the starting point of reading and neither to the choice of reading direction as long as the reading continues in the same direction for the whole chromosome. Hence, $(-2, 3, 1)$, $(3, 1, -2)$, $(-3, 2, -1)$, $(2, -1, -3)$ and $(-1, -3, 2)$ are equivalent representations of the example chromosome.

Genome representation. A multichromosomal genome A, over a set of markers $\mathcal{G}_A$, is a collection of linear and/or circular chromosomes which are of arbitrary but finite length. In this thesis, we will usually assume that there are not multiple copies of the same marker, such that each marker $g \in \mathcal{G}_A$ occurs exactly once in A.

1.2 Comparative Genomics

In *comparative genomics*, two or more genomes, customarily from different species, are studied and compared with respect to their genome structure and/or function.

Traditionally-studied sequence-based mutations, e.g. insertions, deletions and substitutions affect single bases or small segments. However, it was shown that many organisms in the course of their evolution underwent large-scale mutations affecting large chromosomal regions [41,67]. More precisely, markers among closely related species are often similar (not subject to many small-scale mutations), but from one species to another, *modifications* occurred which affected the arrangement or the copy number of markers.

These large-scale mutations become apparent when taking a broader look at a whole chromosome of one genome with respect to another. Then, not only the differences in relative direction/orientation of the markers but also in their arrangement can be observed. As an example, we used the software `r2cat` [60] to compare the genome sequences of two Rickettsia bacteria (studied in [16]). Figure 1.4 shows the markers of the r2cat synteny plot (black) where we added the indication of large blocks of markers (coloured). It depicts the location and orientation of markers in one genome (*Rickettsia africae*[2]) with respect to the location and orientation of these markers in the other genome (*Rickettsia typhi*[3]).

For instance, markers 1, 2 and 3 in *R. typhi* and *R. africae* have the same order but marker 2 does not have the same orientation. Also we detect that markers 4, 5 and 6 are arranged in a different order and also different orientation in the two genomes.

[2] *R. africae* GenBank accession number: AAUY00000000, 01.12.2015
[3] *R. typhi* GenBank accession number: NC_006142, [70], 01.12.2015

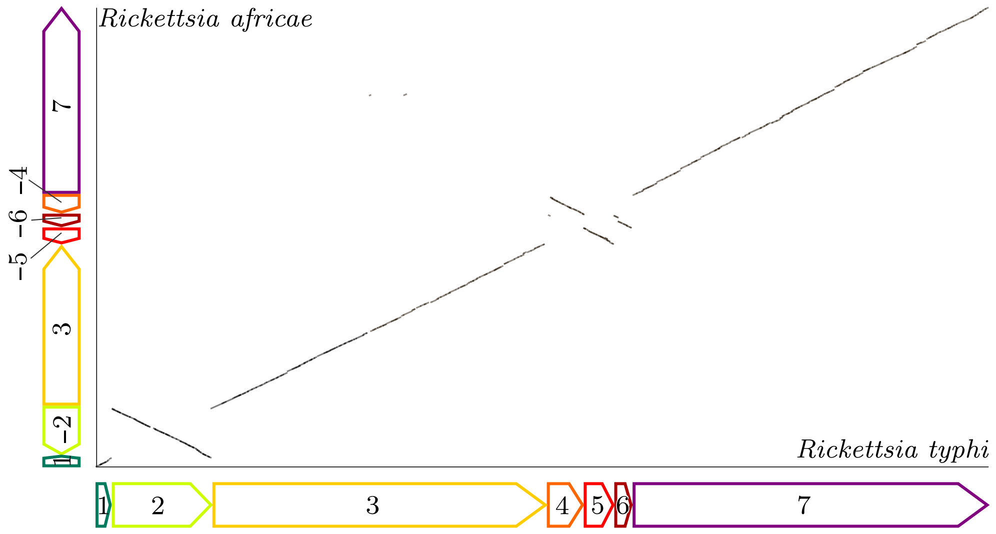


Figure 1.4: A synteny plot of *R. typhi* and *R. africae* produced via the software `r2cat` [60]. The plot was extended by arrows and marker identifiers that indicate a simplified arrangement of the corresponding markers in each genome.

Assuming a common ancestor for the *Rickettsia* genomes, at some point in the course of evolution the orientation of markers (for example marker 2) and arrangement (for example markers 4 and 6) were altered.

More types of changes a marker or sequence of markers can undergo are described below.

1.2.1 Genome Modifications

In the following, we will describe the different modifications that we consider. We distinguish between *arrangement* modifications and *content* modifications. (Griffith *et al.* [52] referred to *balanced* and *unbalanced* rearrangements, as the latter "can disrupt normal gene balance" [52] and the former can not). Both modifications can affect either single elements on one chromosome (*intrachromosomal*) or one or several whole chromosomes (*interchromosomal*), possibly changing the number of chromosomes.

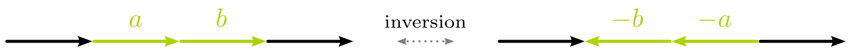
Arrangement Modifications

Here, we describe modifications which affect the arrangement of markers but not the copy number. Some of the modifications are limited to linear and some are limited to circular chromosomes. Also, the evidence of some modifications is observed more frequently than others [39].

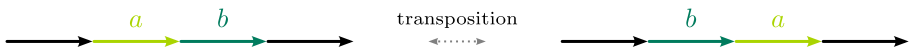
Please note that the schematic illustrations for each modification is kept simple, in that only few markers are displayed, though the modifications can also act on larger intervals that comprise arbitrarily many markers.

Intrachromosomal rearrangements. Genome modifications such as *inversions*, *transpositions* and *block-interchanges* change the arrangement of markers within the same chromosome and can act on circular or linear chromosomes.

Dobzhansky and Sturtevant [41] were among the first to study *inversions*, i.e. (blocks of) markers that are rotated by 180°, for example in the genus of *Drosophila*. An example for an inversion can be seen below, where the orientation of the elements is indicated by arrows and the sign of identifiers:

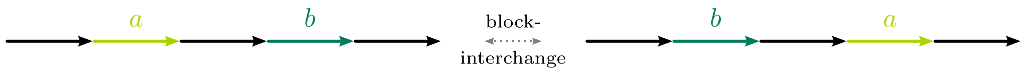


A (conservative) *transposition* moves a section of the genome to another position [4, 40], [28, *p. 403*], [52, *p. 433*]. Biologically, this could also be to another chromosome, however, in mathematical modelling, it is often assumed that the source and target chromosome is the same [6,40]. If the section is moved within the same chromosome, the result is swapping two adjacent intervals of markers [6]. Below, marker a is moved to a position after marker b, in essence exchanging the order of a and b.



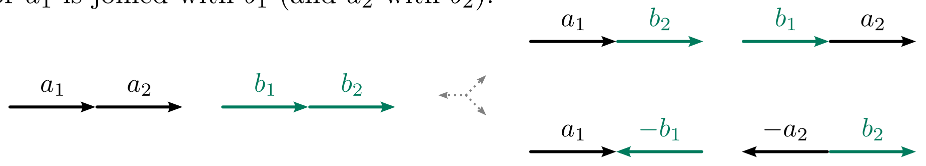
Note that in contrast to an inversion, the orientations of the markers remain unchanged. Some studies differentiate between transpositions and *inverted transposition* (in the latter, the transposed element is inserted in reversed order) [74] others do not distinguish [4].

Christie [31] first introduced *block-interchanges* as a generalisation of a transposition, in a sense that it exchanges two (non-overlapping) intervals on the same chromosome, where these intervals are not necessarily adjacent, as depicted below.



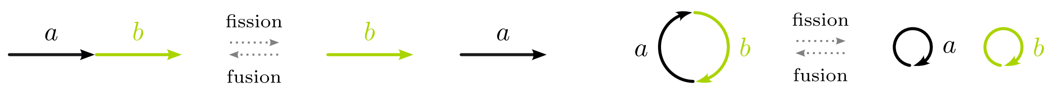
Interchromosomal rearrangements. Genome modifications affecting two chromosomes either change the number of chromosomes or exchange content between two chromosomes. As stated above, some models use transpositions (therefore also block-interchanges) in the biological sense and allow an interval of markers to be transferred to another chromosome [40].

The following graphic shows reciprocal *translocations* which exchange the ends of two linear chromosomes [52, *p. 496 ff.*]. For example, there are the two possible ways for chromosomes $(\circ, a_1, a_2, \circ)$ and $(\circ, b_1, b_2, \circ)$ to be re-joined such that the outcomes are two hybrid chromosomes: Either a_1 is joined with b_2 (and a_2 with b_1) or a_1 is joined with b_1 (and a_2 with b_2).



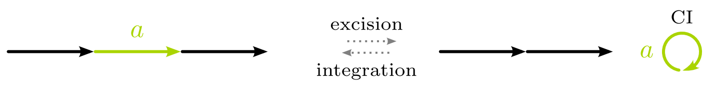
The direction of the chromosome parts needs to be taken care of, as the join happens at the cutting site. Recall that $(\circ, a_1, -b_1, \circ)$ is the same as $(\circ, b_1, -a_1, \circ)$.

A *fission* splits one chromosome into two. The reverse operation is called *fusion*, which concatenates two chromosomes and thus reduces the number of chromosomes by 1.



Depicted are the two modifications on linear chromosomes (left) and circular chromosomes (right). Fissions and fusions are common for example in bacteria [79].

The *excision* of an interval into an extra circular chromosome and its inverse, the *integration*, can be considered either as independent or as one continuous event. In the latter case, the excised interval remains temporarily as *circular intermediate (CI)* before its integration elsewhere. The two marker ends that are joined, in order to form the CI, are not necessarily the same that are cut for integrating the CI. This would result in a different marker order than before the excision. Both excision and integration of marker a are depicted below.



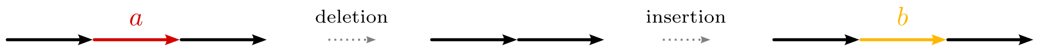
A linear chromosome can be circularised and vice versa. This can be regarded as some kind of excision/integration of a whole chromosome.

Content Modifications

Contrary to the previously mentioned modifications, *content modifications* do not rearrange the markers, but rather modify the content, which means markers (e.g. whole genes or even larger regions) can be inserted or lost.

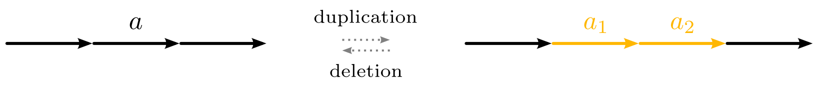
Intrachromosomal changes. Content modifications acting on a single chromosome impact the presence, absence or copy-number of one or several markers, resulting in copy number variations.

We call the gain of markers an *insertion* into the genome, and when we observe the opposite, a loss of markers, this is a *deletion* of the marker (depicted below).



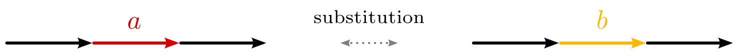
Depending on the source and the target genome these terms are symmetric, so we refer to them with the unifying term *indel*.

An insertion of a copy of a previously existing marker is called *duplication* and is outlined below. During a *replicative transposition* of a transposable element, the old marker is left behind and a copy is inserted at the new position [52, *p. 432f.*] (that is a duplication to another site, on the same or different chromosome). An example of a duplication is shown below.



It can be further specified as a *tandem* duplication, when –as in the above example– the copy lies directly next to the source marker. Note that the loss of one copy of a marker is also a deletion.

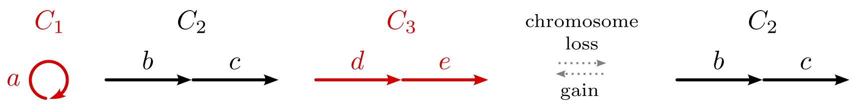
An insertion of marker b which replaces another marker a (thus a is deleted) is called *substitution* and is shown below.



We say marker a is *substituted* by marker b (or vice versa).

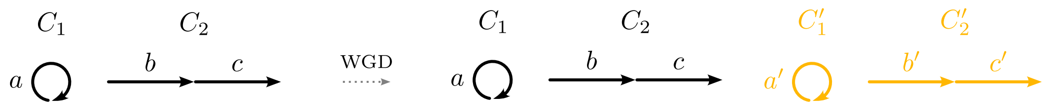
Interchromosomal changes. A whole chromosome may be gained, copied or lost (corresponding to chromosomal insertion, duplication or deletion) thus affecting the number of chromosomes.

Below we show an example of chromosomal loss (chromosomes C_1 and C_3 are lost).



The opposite, chromosomal gain, introduces one or even several whole chromosomes to a genome. These could be, for instance, bacterial plasmids, small extra circular chromosomes that are present in some species but not in other closely related species. Blanc *et al.* [16] studied several *Rickettsia* genomes of which only *Rickettsia felis* has a plasmid. They speculated that it occurred through a single insertion event.

A special case is *polyploidisation* [67], where the whole chromosome set is duplicated, which occurred due to a *whole genome duplication (WGD)* event as depicted below.



For example, the human genome is diploid, meaning each chromosome occurs twice. Higher copy numbers (of chromosomes) may happen in many bred plants, for example, some strawberries contain a k-fold chromosome set [83]: They may be diploid, tetraploid, hexaploid or even octoploid (i.e. they contain $k = 2, 4, 6$ or 8 copies of the chromosome set). Higher copy numbers than 2-fold can also occur in mammals [49].

1.2.2 Genome Modification Models

A typical task in comparative genomics is to quantify the differences between genomes. *Genome rearrangement* is a branch of comparative genomics that investigates the above mentioned different modifications that genomes can undergo. Despite the nomenclature, this field includes arrangement-modifying operations as well as content-modifying operations. In the following, we use *genome modification* for both arrangement as well as content modifications.

A measure of comparison used in genome rearrangements that is called *distance* counts the number of events that occurred between genomes. Generally, we assume parsimony, i.e., the minimal number of events is most likely the real evolutionary distance.

A *sorting sequence* of two genomes describes the sequence of events that, when applied to the first genome, transform it into the second genome. A sequence of shortest length is *optimal*. There may be more than one such sequence, and all of these are then considered as (co-)optimal. A *sorting scenario* shows the source genome, each intermediate genome that arises when the next element of the sorting sequence is applied, and the target genome.

We denote by $\mathcal{R} = \{\text{inversion, transposition, block-interchange, translocation, fission/ fusion, excision/integration}\}$ the set of rearrangements. Furthermore, let the set of content modifications be denoted by $\mathcal{I} = \{\textit{indel}, \textit{duplication}, \textit{substitution}\}$. We will now clarify some terms and their interpendence.

Modification: The observed change induced in the genome.
For example when integrating a circular intermediate into another chromosome we have two separate chromosomes first, but afterwards they form a single chromosome.

Operation: The way one modification is realised.
An example is cutting the genome in two positions and re-joining the loose ends in a different way. One operation can induce different modifications, depending on where/how they are applied.

Model: Definition of one or several operation(s) that induce(s) certain modifications.
One such operation can induce one or different types of modifications depending on how and where it is applied. Further restrictions, e.g., on the type of chromosomes, or succession of operations, may be imposed.

Step: A step in a sorting scenario that is equal to performing one operation under the specified model.
Several steps might be necessary to realise a modification, e.g. if block-interchanges cannot be directly induced with the operations of a model, then three inversions can produce the same resulting genome.

Weight: The weight assigned to an operation or modification w.r.t. a model.
For example deletions could be assigned less weight than inversions. Under unit cost, each operation is assigned the same weight.

In this thesis, we assume unit cost, such that the distance is the number of observed modifications between two genomes. Further restrictions on the types of modifications or types of operations lead to sorting scenarios which employ only specific types of events. More formally: let $\mathrm{R} \subseteq \mathcal{R}$ and $\mathrm{I} \subseteq \mathcal{I}$ be a selected set of arrangement and content modifications, respectively. We denote by $\mathcal{M} = \mathcal{R} \cup \mathcal{I}$ the set of different models and by $\mathrm{M} = \mathrm{R} \cup \mathrm{I}$ the set of modifications allowed under a specific model $\mathrm{M} \in \mathcal{M}$. Note that $\mathrm{M} = \{\} \cup \{\}$ or $\mathrm{M} = \{\} \cup \{\textit{indel}\}$ are legitimate models, but not worthwhile to study. In general, given a model comprising a certain set of modifications, the comparison of two genomes is measured as follows.

Definition 1 (Generalised Genome Modification Distance Problem): *Given two genomes A over $\mathcal{G}_A$ and B over $\mathcal{G}_B$ the distance $d_{\mathrm{M}}(A, B)$ of A and B under genome modification model* $\mathrm{M} \in \mathcal{M}$ *is the minimum number of steps required to sort A into B allowing only operations of* M.

The definition of a distance of two genomes immediately implies that two genomes A and B are *sorted* when $A = B$. Otherwise they are *unsorted*.

The choice of model has a huge influence on the modifications we can render possible. For example, if no interchromosomal operations are included, no content can be moved from one chromosome to another. It also plays a role in the number of steps we need, e.g., an inversion may be counted as a single operation in one model, or as four operations in another. In the latter case an inversion may be realised by cutting to the left and to the right of the marker and then sticking the ends together again in a different way, akin to inverting the marker.

Although there are many interesting problems to study in genome rearrangements, such as ancestral reconstruction or the median problem, the first and main intent, when studying genome modification models, is to determine the pairwise distance and compute an optimal sorting scenario. Many different models have been studied in the past, and algorithms for distance computation and sorting presented. Li *et al.* [67] give a detailed review of genome operations and models studied until 2006. Besides the history of publications for the different problems, the authors state which of them are proven to be NP-hard and also mention approximate solutions to some problem variants.

In this work we focus on the study of the *double cut-and-join (DCJ)*, the *DCJ-indel*, the *inversion* and the *inversion-indel models* and their interrelation. Some combinations related to the inversion or DCJ models are given in Table 1.2; it shows which modifications are realised in each model, sorted by intra-/interchromosomal rearrangements and content modifications. Closely related to the listed models are also the following models: Hannenhalli-Pevzner (HP) [14,53,61], single cut or join (SC/J) [47], single cut or join-*indel* (SC/J-*indel*) [46], single cut-and-join (SCJ) [9] and the 3-break model [1].

For some models there exist weighted solutions not operating under the unit cost scheme, for instance the DCJ-*indel* model with distinct operation cost [37]. Other models prove to be difficult in computing an exact solution for the general case (without restrictions) even under unit costs. For example, the exact computation of the *inversion-indel* distance, first introduced in 2000, has remained unsolved for the past 17 years. Two restrictive cases have been published: the inversion-*deletion* model by

Table 1.2: Different models and implied genome modifications from $\mathcal{R}$ and $\mathcal{I}$. Operations are in order: inversion, transposition, block-interchange, translocation, fission/fusion, excision/integration, *insertion*, *deletion*, *duplication*, *substitution*. Inclusion in model: direct (●), or number of steps. Υ means the excision of a CI must immediately be followed by its integration.

		$\mathcal{R}$						$\mathcal{I}$			
Model M		Inv	Trp	Bl-Int	Trl	F/F	Ex/In	*in*	*del*	*dup*	*sub*
INV (inversion)	[8,12,54,63]	●	3	3							
-*deletion*-*duplication*	[62]	●	3	3					●	●	
-*indel*	[42,89]	●	3	3				●	●		
DCJ (double cut-and-join)	[13,90]	●	2	2	●	●	●				
-*indel*	[25,26,33,35,37,91,92]	●	2	2	●	●	●	●	●		
-*indel*-*duplication*	[92]	●	2	2	●	●	●	●	●	●	
-*duplication*	[3,92]	●	2	2	●	●	●			●	
-*substitution*	[37]	●	2	2	●	●	●				●
rDCJ (restricted DCJ)	[64]	●	2	2	●	●	Υ				
rDCJ-*indel*	[24,36]	●	2	2	●	●	Υ	●	●		

Translocations, excisions/integrations involving two telomeres can be considered as fissions/fusions.

Nadia El-Mabrouk in 2000 [42,43] (whose distance computation is not exact) and the inversion-*indel* model for only certain "good" types of instances (Willing *et al.* [89]).

The main focus of this thesis is on giving an exact solution to the inversion-*indel* distance problem without restrictions. We seek to do so by computing the DCJ(-*indel*) distance.

1.3 Content and Structure of the Thesis

This thesis focuses on the double cut-and-join (DCJ) and the inversion model both with and without insertions and/or deletions.

In Chapter 2 the foundation is laid which comprises commonly used definitions, data structures unified to fit all covered models in this thesis, and the DCJ model. The contribution of this thesis is then split in two main parts.

The first part (Chapter 3) concentrates on the DCJ sorting problem (without insertions and deletions). In this part we study how to find one DCJ sorting scenario among all co-optimal DCJ sorting scenarios, assuming each co-optimal scenario is equally likely. The effect of uniform sampling over other sampling methods is evaluated by integrating the algorithm to derive such a uniformly sampled scenario into an existing software suite.

The second part of this thesis is spread over several chapters, ultimately solving the inversion-*indel* distance problem. It opens with a review of the DCJ-*indel* model in Chapter 4 that gives the distance as an offset to the DCJ distance as well as sorting procedures. We continue with elaborations on special cases of the inversion-*indel* model (Chapter 5). For this, we first elaborate on the data structures and their properties necessary to solve the well-known inversion distance problem (from 1995) in certain instances considered *good*. The previously known relation of the inversion to the DCJ distance is extended towards the general inversion distance that includes insertions and deletions. Among the so-called *bad* instances, we review the solution to the inversion distance. We solve special cases with insertions and deletions, and we present a generalised data structure for solving the inversion-*indel* distance problem.

After that, we study the solution to the general problem, that is, computing the distance allowing inversions, insertions and deletions when bad instances may be present. We are finally the first to present an exact solution for the distance problem with inversions and indels and offer a procedure to sort two genomes under this model.

Furthermore, due to the nature of our distance computation, we are also able to provide a sorting procedure.

The thesis concludes with an overview of genomic distance relations and prospects related to the theory presented in this work. More elaborate demonstration and information is given in the appendix.

Nature has a great simplicity and therefore a great beauty.
RICHARD P. FEYNMAN

CHAPTER 2

Important Data Structures and Models

Traditionally, different simplifications in genome modification models were studied. Depending on the model, that allows a distinct set of genome modifications, some restrictions are conventional while some are convenient. For example the restriction to a single chromosome or the restriction to circular chromosomes. Usually when a new model, more specifically a new combination of allowed rearrangements, is introduced, any type of content modifying operation is disallowed, making the data structures much simpler compared to the model with indels. However, we prefer to include indels right from the start along with generalised terminology.

In this chapter we first present the concept of indels and adjacencies before we characterise different graph structures and details of the basic genome modification model of this thesis, that is the double cut-and-join model.

2.1 Insertions and Deletions

We first take a look at the handling of genomes having unequal content. More precisely we have $\mathrm{I} = \{\mathit{insertions}, \mathit{deletions}\} \subseteq \mathcal{I}$ but duplicated markers are not allowed. This section establishes definitions of genomes that have equal or unequal content, definitions of insertions and deletions and the operation to perform such an insertion or deletion.

The sorting of two genomes can be considered as a directed process where operations are applied to the source genome in order to derive the target genome. Given two genomes A and B, a marker that occurs in A but not in B is witness of a *deletion* if the sorting is considered in the direction from A towards B. The same marker is witness of an *insertion* if the sorting is considered in the opposite direction. Under unit costs the cost of an insertion or deletion (*indel* for short) is the same as that of an inversion or DCJ operation.

2.1.1 Core Genomes and Unique Markers

Because we do not want to impose a direction on the sorting, rather than calling a specific marker a deletion or insertion, we speak of a *unique marker* if it occurs uniquely in A or uniquely in B. Otherwise, if the marker is common to both A and B, it is called *common marker.*

Definition 2 (Marker Sets): *Given a genome A over the set of markers $\mathcal{G}_A$ and a genome B over the set of markers $\mathcal{G}_B$, let $\mathcal{G} = \mathcal{G}_A \cap \mathcal{G}_B$ be the set of markers common to both genomes. We denote by*

$$\mathcal{A} = \mathcal{G}_A \backslash \mathcal{G} \quad \textit{and} \quad \mathcal{B} = \mathcal{G}_B \backslash \mathcal{G}$$

the sets of markers that occur uniquely in A and uniquely in B, respectively.

We consider the insertion of two or more adjacent markers as a single operation. These unique markers will not be split up in the course of the sorting process, since they need to occur in the other genome in the same order and direction. Splitting and rejoining them would not diminish the length of the scenario. In pairwise comparison, we can therefore replace them by a single unique marker. The same is done for deletions of consecutive markers.

Example 1: The graphs of unichromosomal circular genomes $A = \{(a, w, c, -d, y, e, -z, b, f, x, -h, -j, -i, g)\}$ and $B = \{(a, s, b, c, d, e, u, -v, f, g, h, i, r, j, t)\}$ are shown in Figure 2.1. The set of markers that are common to both genomes is: $\mathcal{G} = \{a, b, c, d, e, f, g, h, i, j\}$. The unique markers are indicated in colour and we have: $\mathcal{A} = \{w, x, y, z\}$ and $\mathcal{B} = \{r, s, t, u, v\}$. The markers from $\mathcal{A}$ each represent a deletion from genome A

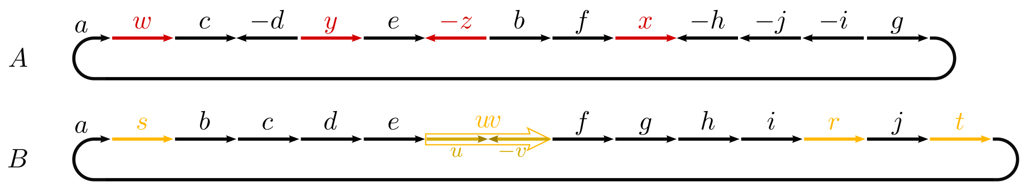


Figure 2.1: Genome graph of genomes $A = \{(a, w, c, -d, y, e, -z, b, f, x, -h, -j, -i, g)\}$ and $B = \{(a, s, b, c, d, e, u, -v, f, g, h, i, r, j, t)\}$ showing unique markers.

and the markers from $\mathcal{B}$ represent insertions (if A is sorted into B). Since u and $-v$ are adjacent unique markers, they are regarded as one, and assigned the identifier uv. The genomes and sets of markers are updated accordingly, e.g. $\mathcal{B} = \{r, s, t, uv\}$. ◇

Traditional genome rearrangement models such as the inversion [55] or the DCJ model [13,90] dealt only with genomes that have exactly one copy of a marker in each genome. They are in fact a special case of the corresponding generalised models with indels in such that A and B do not have (or ignore) unique markers. The following definition describes this analogy:

Definition 3 (Core Genome): *Given a genome A over $\mathcal{G}_A$ and a set of core markers $\mathcal{G} \subseteq \mathcal{G}_A$ then the* core genome *of A w.r.t. $\mathcal{G}$, denoted by $A|_{\mathcal{G}}$, is the genome derived by keeping only the markers from A that are present in $\mathcal{G}$ and removing all other markers.*

We consider only pairwise comparison such that the set of core markers $\mathcal{G}$ are the markers that genomes A and B have in common, i.e. $\mathcal{G} = \mathcal{G}_A \cap \mathcal{G}_B$.

Example 1 (continued): For the same marker sets as before, the core genomes of A and B are $A|_{\mathcal{G}} = \{(a, c, -d, e, b, f, -h, -j, -i, g)\}$ and respectively $B|_{\mathcal{G}} = \{(a, b, c, d, e, f, g, h, i, j)\}$. The corresponding genome graphs can be seen in Figure 2.2. $\diamond$

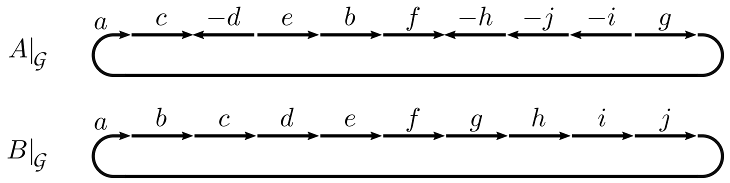


Figure 2.2: The two genome graphs of core genomes $A|_{\mathcal{G}}$ and $B|_{\mathcal{G}}$ where $A = \{(a, w, c, -d, y, e, -z, b, f, x, -h, -j, -i, g)\}$ and $B = \{(a, s, b, c, d, e, uv, f, g, h, i, r, j, t)\}$.

2.1.2 A First Upper Bound to the Distance with Unique Markers

Sometimes we are interested only in the number of R-operations that are used in a sorting scenario. For example, the first model that we study in Section 2.3 (the double cut-and-join model) allows no operations from I. For this, we ignore unique markers and define:

Definition 4 (Genome Rearrangement Distance): *Given two genomes A and B over $\mathcal{G}_A$ and $\mathcal{G}_B$, respectively, then the distance under genome modification model $\mathrm{M} \in \mathcal{M}$ using only arrangement modifications is given by*

$$d_{\mathrm{R}}(A, B) := d^{\mathrm{I}}_{\mathrm{R}}(A|_{\mathcal{G}}, B|_{\mathcal{G}}) = d_{\mathrm{M}}(A|_{\mathcal{G}}, B|_{\mathcal{G}}), \tag{2.1}$$

where $\mathcal{G} = \mathcal{G}_A \cap \mathcal{G}_B$ and $\mathrm{M} = \mathrm{R} \cup \mathrm{I}$. In other words $d_{\mathrm{R}}(A, B)$ is the distance of sorting all common markers in A and B under M without using content modifications.

Analogously, $d^{\mathrm{I}}(A,B)$ gives the minimum number of content modifications necessary to conform $\mathcal{G}_A$ and $\mathcal{G}_B$ without using arrangement modifications. Bear in mind that consecutive unique markers in a genome are replaced by a single unique marker, as for instance in Example 1. It is thus required to take into account the arrangement of markers in each genome.

For the distance of indels $d^{id}(A,B)$ that ignores arrangement modifications, we delete from $\mathcal{G}_A$ the content of $\mathcal{A}$ (remove deletions) and introduce insertions (the content of $\mathcal{B}$), yielding $\mathcal{G}_B$. Hence, the number of operations is equal to the number of elements in $\mathcal{A}$ and $\mathcal{B}$. This directly leads us to a first intuitive upper bound for general distances including indels:

Observation 1. Given genomes A over $\mathcal{G}_A$ and B over $\mathcal{G}_B$ without duplications, then

$$d^{id}_{\mathrm{R}}(A,B) \leq d_{\mathrm{R}}(A,B) + d^{id}(A,B) \leq d_{\mathrm{R}}(A,B) + |\mathcal{A}| + |\mathcal{B}|, \tag{2.2}$$

where $\mathcal{G} = \mathcal{G}_A \cap \mathcal{G}_B$, $\mathcal{A} = \mathcal{G}_A \backslash \mathcal{G}$ and $\mathcal{B} = \mathcal{G}_B \backslash \mathcal{G}$ and R is the set of allowed rearrangement operations (generalised from [25]) and $d^{id}(A,B)$ gives the differences in copynumbers of markers.

Note that, when $\mathrm{R} = \{\}$ (or generally $d_{\mathrm{R}}(A,B) = 0$, as given in Definition 4), Inequality (2.2) gives equality. It also does when $d^{id}(A,B) = 0$, since in both cases only one class of operations (I resp. R) has to be performed. However, when operations from both sets R and I are allowed (and also necessary), there sometimes are sorting scenarios that need fewer steps than this upper bound.

In the course of sorting one genome into another, two unique markers in one genome may become adjacent. Similar to the assumption above (the grouping of u and $-v$ in Example 1), we follow the parsimony principle and hence consider it biologically more plausible that those markers were inserted (or lost) at once rather than in separate steps. The following example outlines how larger blocks of unique markers can be obtained while sorting the two genomes, making apparent why the relation given by Observation 1 is in fact an inequality rather than an equality.

Example 2: Given two genomes $A = \{(\circ, a, y, -b, z, \circ)\}$ and $B = \{(\circ, a, b, \circ)\}$ we can sort A into B in three steps: by deleting y, deleting z and then performing an inversion of marker b. However, as shown in Figure 2.3, the sorting can also be done via one inversion and a single deletion: First, block $-b\,z$ is reversed, which makes marker z a neighbour of marker y. Then it is possible to delete the block $y\,-z$ in one step. Thus, in total we have two, instead of three operations. ◇

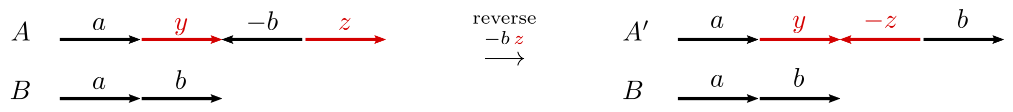


Figure 2.3: One operation that produces two consecutive unique markers while simultaneously sorting the common markers.

It is easy to see that the more indel operations can be *saved* by grouping unique markers during the (optimal) rearranging, the more the overall distance is decreased. Our goal is to minimise the number of steps in the sorting scenario. This can be approached by maximising the number of neighboured unique markers during optimal sorting.

In Chapter 4 we shall investigate the grouping of unique markers while using optimal DCJ operations which will also be used in Chapters 5-7. Also the balancing of indels with neutral or even counter-optimal operations is discussed.

2.1.3 Extremities, Adjacencies and Labels

We identify the orientation of a marker by distinguishing its two ends. For each marker we define:

Definition 5 (Extremities): *The ends of a marker g are called* extremities. *More precisely they are called* tail *and* head *and denoted by g^t and g^h, respectively.*

Analogous to unique resp. common markers we refer to their respective extremities as *unique* and *common extremities.*

Two markers that lie next to each other in the same chromosome are called *adjacent.* Their respective extremities form an *adjacency.* For example, in chromosome $(\circ, a, b, \circ)$ markers a and b are adjacent and form the adjacency $a^h b^t$. An adjacency such as $\circ\, a^t$ is also called *telomere*, since the cap symbol $\circ$ represents the end of a linear chromosome. In pairwise comparison, of all extremities of one genome only the common extremities have a counterpart in the other genome whereas unique extremities are unmatched. We therefore define a kind of adjacency that is formed by extremities of common markers (or caps) only.

Definition 6 ($\mathcal{G}$-Adjacency): *Given genome A over $\mathcal{G}_A$ and a set $\mathcal{G} \subseteq \mathcal{G}_A$, two extremities p and q from markers in $\mathcal{G}$ are $\mathcal{G}$-*adjacent *if in-between them in A there are no other extremities of markers from $\mathcal{G}$. The string of marker extremities between p and q (in this reading direction) in A is the* label *of the adjacency pq, denoted by $\ell(pq)$. Together they form a $\mathcal{G}$-*adjacency *denoted by $p\ell(pq)q$ that is* unlabelled *if $\ell(pq) = \varepsilon$. Otherwise its label is non-empty and contains only extremities of markers from $\mathcal{A}$.*

Example 2 (continued): In Figure 2.3 A' has unique markers y and $-z$ between markers a and b, the latter two are adjacent in $A'|_{\mathcal{G}}$. Then the label of the corresponding $\mathcal{G}$-adjacency is given by $\ell(a^hb^t) = y^ty^hz^hz^t$ and we use the notation $a^hy^ty^hz^hz^tb^t$ or alternative the short form $a^hy{-z}b^t$. $\diamond$

Note that if unique markers are reversed, so is the label of the corresponding $\mathcal{G}$-adjacency. When the reading direction changes, also the label needs to be read in the reverse direction. Indicating the reversed section by an overbar $\bar{\ }$, we have $\overline{p\ell(pq)q} = q\,\overline{\ell(pq)}\,p = q\ell(qp)p$, for example, $a^hy^ty^hz^hz^tb^t$ becomes $b^tz^tz^hy^hy^ta^h$ (or $a^hy\ {-z}b^t$ becomes $b^tz\ {-y}a^h$).

Moreover, apart from $\mathcal{G}$-adjacencies with no label, there can also be $\mathcal{G}$-adjacencies with no marker extremities. These are called *singletons* and represent a whole chromosome which is only composed of unique markers. There are two types of these: a labelled $\mathcal{G}$-adjacency $p\,\ell(pq)\,q$, where both p and q are caps, represents a whole linear chromosome that is only made of unique markers and is called a *linear singleton.* In the same way this applies to circular chromosomes that do not contain any common markers and also form a $\mathcal{G}$-adjacency that only contains a label. These are referred to as *circular singletons.* In contrast to all other $\mathcal{G}$-adjacencies the latter is the only one containing a circular string, the label [26]. A core genome naturally does not contain singletons, instead only chromosomes that have at least one common marker.

2.1.4 The Indel Operation on $\mathcal{G}$-Adjacencies

An indel operation can act only on unique markers, not on markers from $\mathcal{G}$. In this thesis we only deal with genomes that have no duplicated markers, thus each element of $\mathcal{G}$, $\mathcal{A}$, or $\mathcal{B}$ occurs at most once in the genomes. Therefore, another restriction is that an insertion cannot produce duplicate markers (in neither of the three sets $\mathcal{G}$, $\mathcal{A}$ or $\mathcal{B}$ [26]). As a consequence of this –if the sorting direction is from A to B– we can only delete markers from $\mathcal{A}$ and insert one copy for each marker in $\mathcal{B}$.

In terms of adjacencies, an indel operation acts only on the label of a single $\mathcal{G}$-adjacency, such that if it is a deletion, it replaces the label of a $\mathcal{G}$-adjacency in A by ε and the $\mathcal{G}$-adjacency in question becomes unlabelled. If the indel operation is an insertion, then the label of a $\mathcal{G}$-adjacency in B needs to be updated and the inserted unique markers either replace the empty label of the $\mathcal{G}$-adjacency or are placed before, in-between or after existing unique markers of that label.

Example 1 (continued): A $\mathcal{G}$-adjacency $a^hw^tw^hc^t$ to which a deletion of w is applied becomes a^hc^t with label $\ell(a^hc^t) = \varepsilon$. Similarly, a $\mathcal{G}$-adjacency e^hf^t into which the unique marker uv is inserted (such that the tail of uv follows the head of e) becomes $e^h uv^t uv^h f^t$. ◇

While a deletion can technically also act on only parts of the label of a $\mathcal{G}$-adjacency this would defy the parsimony principle. Therefore, we delete only complete labels. We will see later (in Chapter 4) what measures can be taken to group not only deletions but also insertions and how this will affect the point of insertion.

2.2 Graph Structures for Representing Genomic Relations

In order to find a parsimonious sequence of rearrangements (and indels) sorting one genome into the other, it is convenient to find some data structure that represents the relation between the two genomes.

During the study of different genome modification models, several data structures emerged. According to the needs or restrictions of each model, i.e. unichromosomal or multichromosomal but also the type of chromosomes (circular, linear or both types), data structures providing a most simple way to compute the corresponding distances or sorting scenarios were introduced. In the following we will subsume the different data structures. In order to provide easier access to the relation among them, we generalise the individual presentation of data structures, thus deviating from the customary notation.

2.2.1 Breakpoint Graph

The *breakpoint graph (BG)* proposed by Bafna and Pevzner [5] in 1993 was used for instance for the computation of the inversion distance [55] of two genomes. Well suited for unichromosomal genomes, its construction is particularly simple if one of the genomes is the identity permutation of the common markers, meaning all the markers of one genome are in strictly ascending order with only positive signs. The breakpoint graph was generalised to include labels for the computation of the inversion-*indel* distance by El-Mabrouk in 2001 [43]. These two distances are computed for one chromosome at a time only and we restrict the use of the breakpoint graph to unichromosomal circular genomes.

Construction. The labelled breakpoint graph of two unichromosomal circular genomes A and B is the graph $\mathrm{BG}(A, B)$ that has a vertex for each common extremity and has adjacency edges that connect these as follows. A-edges connect two vertices for which the corresponding common extremities form a $\mathcal{G}$-adjacency in A. Likewise,

B-edges connect two vertices whose common extremities form a $\mathcal{G}$-adjacency in B. The label of each $\mathcal{G}$-adjacency becomes the label of the corresponding adjacency edge (where the order of extremities in the vertex label corresponds to that of the corresponding $\mathcal{G}$-adjacency label). An example is given in Figure 2.4.

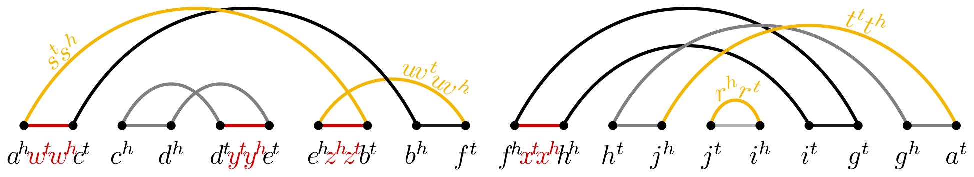


Figure 2.4: Breakpoint graph $\mathrm{BG}(A, B)$ of genomes $A = \{(a, w, c, -d, y, e, -z, b, f, x, -h, -j, -i, g)\}$ and $B = \{(a, s, b, c, d, e, uv, f, g, h, i, r, j, t)\}$, which are the genomes from Example 1. Vertices arranged as read from genome A. A-edges are straight while B-edges are arcs. Labelled A-edges are drawn in red and labelled B-edges are drawn in yellow.

Connected Components. We assume that the chromosomes are not singletons. The adjacency edges connect the vertices in such a way that each vertex is the endpoint of one A-edge and one B-edge. The breakpoint graph therefore consists of a collection of cycles. The length of a cycle is its number of adjacency edges, thus a multiple of two for each connected component.

Diagram Layout of the Breakpoint Graph. Some problems in genome rearrangement require to analyse the interplay of connected components. For this, a view of the breakpoint graph is fixated. The vertices of the diagram $\mathrm{BG}(A_g, B)$ are then arranged in the same order in which the corresponding markers are read from genome A, starting with the head of g and due to circularity, ending with the tail of g, all drawn in a horizontal arrangement. However, when a different start marker is chosen, for example $-g$, which changes the reading direction, the construction yields a different layout. The same happens when the two genomes are swapped and the reading direction of the other genome is taken for the distribution of vertices. This can be observed in Figure 2.5 which fixates a different view by distributing the vertices according to genome B, thus depicting $\mathrm{BG}(B_a, A)$ while Figure 2.4 shows $\mathrm{BG}(A_a, B)$. The break-

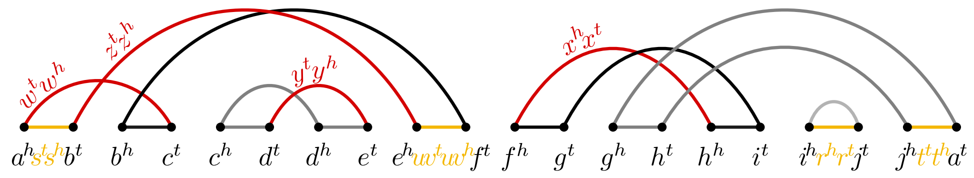


Figure 2.5: $\mathrm{BG}(B_a, A)$ for the same genomes as in Figure 2.4.

point graph constructed here is therefore rather a diagram than a graph. Most often when referring to the breakpoint *graph* we refer to the fixated view.

2.2.2 Adjacency Graph

Along with the study of the DCJ model [13,90] the *adjacency graph (AG)* was introduced [13] which allows for a symmetric display of the relation of the two genomes under consideration. For the computation of the DCJ-*indel* distance [26,92] the adjacency graph was generalised to represent $\mathcal{G}$-adjacencies [26]. It can be used for uni- or multichromosomal genomes with circular and/or linear chromosomes.

Construction. The labelled adjacency graph AG(A, B) of two genomes A and B is a graph that has one vertex for each $\mathcal{G}$-adjacency of A and one vertex for each $\mathcal{G}$-adjacency of B. The label of the vertex corresponds to the label of the adjacency (where the order of extremities in the vertex label corresponds to that of the corresponding $\mathcal{G}$-adjacency label). Furthermore, it has two extremity edges for each common marker $g \in \mathcal{G}$: One edge connecting the vertex in A and the vertex in B that contain g^h and one edge connecting the two vertices in A and B that contain g^t. The adjacency graph of the two genomes from Example 1 can be seen in Figure 2.6.

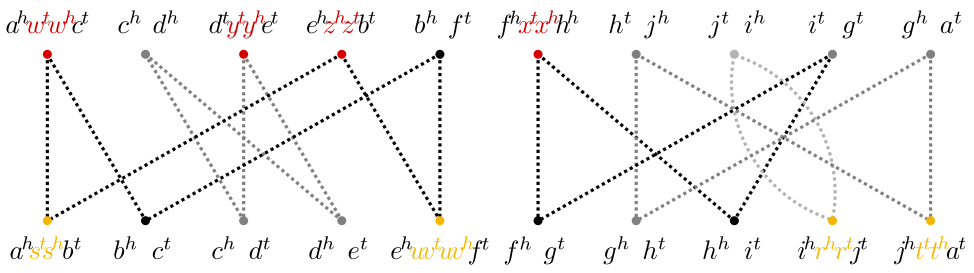


Figure 2.6: Adjacency graph AG(A, B) of genomes $A = \{(a, w, c, -d, y, e, -z, b, f, x, -h, -j, -i, g)\}$ and $B = \{(a, s, b, c, d, e, uv, f, g, h, i, r, j, t)\}$ which are the genomes from Example 1.

Connected Components. The vertices are connected by extremity edges in such a way that we have a collection of connected components (or simply components). A component can either be a circular singleton, a path or a cycle. We consider the *length* of a component to be its number of extremity edges. Each vertex can have a degree of 0, 1 or 2 analogous to the number of common extremities in the concerned $\mathcal{G}$-adjacency. If a component connects only vertices with degree 2 it is a cycle and of even length. Vertices with no common extremity consist of unique markers only and represent linear singletons if they contain two caps, or they represent circular singletons if they have no caps. Vertices with only one common extremity represent telomeres and as such an end of a linear chromosome. If at least one vertex in a specific component has a degree of less than two, it is a path. Paths are distinguished according to the genomes its endpoints belong to, thus forming *AA*-, *AB*- and *BB*-paths. Note that a linear singleton corresponds to a path of length 0, more precisely an *AA*-path, if the singleton is in genome A, or a *BB*-path, if the singleton is in genome B. Further use of the different types of connected components in the adjacency graph will be made first in Section 2.3 and then in Chapter 3.

2.2.3 Master Graph

The *master graph (MG)* [48] visualises a unification of the breakpoint graph and the adjacency graph. Since for the theory presented in this thesis the master graph is not used elsewise, we will restrict its use to unichromosomal circular genomes only.

Construction. The labelled master graph of two genomes A and B is the graph $\mathrm{MG}(A, B)$ that inherits from $\mathrm{AG}(A, B)$ the fact that we have two sets of vertices and also the extremity edges. We take all the vertices from $\mathrm{BG}(A, B)$ and $\mathrm{BG}(B, A)$ as well as the corresponding adjacency edges.

Connected Components. As each vertex represents exactly one common extremity, it is connected to exactly one extremity edge and either an A- or a B-edge. Since we assume the genomes to be unichromosomal and circular, the graph is a collection of cycles. Each cycle has a multiple of four edges (two extremity edges, one A-edge and one B-edge).

Relational Diagram

The *relational diagram ($\mathcal{R}$)* [19] is a specific view of the master graph and we adopt the term *diagram* (similar to the *reality and desire diagram* by [82]).

Given two unichromosomal circular genomes A and B, their relational diagram $\mathcal{R} = \mathrm{MG}(A_g, B_f)$ shows the vertices and A-edges of $\mathrm{BG}(A_g, B)$ in an upper horizontal line and those of $\mathrm{BG}(B_f, A)$ in a lower horizontal line. The extremity edges are constructed in the same way as before. As with the master graph, the relational diagram is a collection of cycles with the number of edges being a multiple of 4. An example that shows how different the layout of a component in the mastergraph or relational diagram can be, is given in Figure 4.3 on page 75.

2.2.4 Properties of Cycles

Moreover, we consider the labelling of edges and cycles. The label of an A- resp. B-edge is called $\mathscr{A}$- resp. *$\mathscr{B}$-label*, otherwise (if there is no unique marker between the corresponding extremities) the A- or B-edge is said to be *unlabelled*. Likewise, any cycle with only unlabelled edges is called *unlabelled*, otherwise it is *labelled*. A cycle that has only $\mathscr{A}$-labels is called an $\mathscr{A}$-cycle (resp. $\mathscr{B}$-cycle if it has only $\mathscr{B}$-labels). All other labelled cycles are called $\mathscr{AB}$-cycles.

Although the relational diagram fixates a view, still, for any choice of g or f as start markers, we have the same connected components as the following proposition shows.

Proposition 1: *Let A and B be two unichromosomal circular genomes with possibly unequal marker content but without duplications. Then, for any circular rotation and/or change of reading direction of genome A and/or B, the components of the resulting relational diagram are identical.*

Proof. It is necessary to show that all components have the same edges and labellings and that they are present in any $\text{MG}(A_g, B_f)$, that for any fixation of the master graph derived when starting to read genome A in $g \in \mathcal{G}$ and genome B in $f \in \mathcal{G}$.

As both A and B have exactly one circular chromosome each, the change in start marker or the change of reading direction do not impose any change on the neighbouring of any markers and their extremities, thus neither on the A- or B-edges or their labels. Obviously, as neither $\mathcal{G}, \mathscr{A}, \mathscr{B}$ or the composition of adjacencies was altered, the vertices and extremity edges are intact. For the same reason, we know that no components are removed or added. At the same time, the edges and vertices are connected in a way that they construct the same cycles that have the same labelling for any rotation of the genomes. □

We observe that all connected components are equal also in the adjacency graph and breakpoint graph.

For models that rely only on the information such as length and labelling of single components and not on the relation of components to one another or other properties of the component's edges, the graph data structures are sufficient to use.

2.2.5 A Note on Relations between the Types of Graphs

By construction, $\text{BG}(A, B)$ can be obtained from $\text{MG}(A, B)$ by collating all extremity edges and their two end points into a single vertex. Analogously, $\text{AG}(A, B)$ can be obtained from $\text{MG}(A, B)$ by fusing each adjacency edge and its two end points from MG into a single vertex, such that the vertex represents the corresponding $\mathcal{G}$-adjacency with its label.

Let us assume that the two genomes we intend to study are unichromosomal circular genomes and have identical core genomes, thus the same common extremities form a $\mathcal{G}$-adjacency. All graphs have one cycle per $\mathcal{G}$-adjacency, which is a *trivial* cycle. The master graph has two extremity edges and one A- and one B-edge per cycle. Then the breakpoint graph has exactly one A- and one B-edge per cycle and the adjacency graph has two extremity edges per cycle.

While using, for instance, the adjacency graph, we may allow for further types of genomes. For two genomes with identical core genomes the graph has one vertex for each linear or circular singleton and two paths that each have exactly one extremity edge for every linear chromosome that is not a singleton (a path of length 1 is also a trivial component).

One disadvantage of $\mathrm{BG}(A, B)$ is that while changes are applied to its elements it is hard to visually follow the changes that are induced in the genome(s). This is easier for the other graphs as the two genomes can be drawn to appear separated. However, only the relational diagram and the diagram view of the breakpoint graph fixate a view with two distinct vertex sets, strictly following the marker order, while the adjacency graph and the master graph could have the vertices scattered arbitrarily.

On the other side, the breakpoint graph does have one advantage: if it is drawn as in the examples (with arcs of different height) it is easier to analyse how the cycles are related to each other, as it shows plainly which cycles have intersecting edges and which cycles are embedded or completely independent of other cycles (see for example Figure 2.4). We will come back to this in Chapter 5.

The presented graph structures are used to compute certain genome modification distances and sorting scenarios accordingly. Usually the operations are applied as changes to the data structures in order to obtain a graph with only trivial components. In the following section we present one of these models that serves as the basis for further theory studied in this thesis.

2.3 The Double Cut-and-Join Model

The concept behind a *double cut-and-join (DCJ)* operation is to perform two cuts in the genome, yielding four blunt ends, and performing two joins gluing the four ends back together in a different way than before. In the DCJ model a series of DCJ operations is performed and one genome is transformed into the other. In contrast to other genome modification models, the DCJ model can handle so-called *circular intermediates* (CI), small circular chromosomes that arise through excision and are integrated later on. Under the DCJ model the set of rearrangements $\mathrm{R} \in \mathcal{R}$ for multichromosomal genomes comprises:

- inversion
- translocation
- fission / fusion
- excision / integration of a circular chromosome.

Transpositions and block-interchanges can be modelled by the excision of a circular intermediate and its integration elsewhere (two DCJ operations). The DCJ model can handle linear as well as circular chromosomes but accepts only genomes with the same marker content and no duplications. We assume these preconditions unless stated otherwise. Formally, we have $\mathrm{I} = \{\,\}$ and therefore $\mathrm{DCJ} \equiv \mathrm{R}$ as enumerated above.

The DCJ model was first introduced by Yancopoulos, Attie and Friedberg in 2005 [90]. In 2006, Bergeron, Mixtacki and Stoye [13] refined the DCJ model and introduced the *adjacency graph* which we reviewed in Subsection 2.2.2 and that is unlabelled for this model. The authors provided a simple distance formula and also a linear time algorithm for sorting two genomes.

In this section we will briefly cover the basics necessary for Chapter 3 and the DCJ-*indel* model.

2.3.1 The DCJ Operation on Unlabelled $\mathcal{G}$-Adjacencies

The double cut-and-join operation that acts on adjacencies of a genome is formally defined as follows.

Definition 7 (Double Cut-and-Join Operation [13]): *Given two unlabelled adjacencies $u_1 = p\,q$ and $u_2 = r s$ where p, q, r and s are common extremities, one cut separates the two extremities of u_1 and the other cut separates the two extremities of u_2. There are two ways to join the four extremities such that two new adjacencies are created:*

(*v*) $v_1 = p\,r$ *and* $v_2 = q\,s$ *or*

(*w*) $w_1 = p\,s$ *and* $w_2 = r\,q$.

The cases where either one adjacency of u_1 *and* u_2 *or both adjacencies are telomeres can be derived from above by replacing one or more extremities among* p, q, r *or* s *by cap symbols.*

2.3.2 The Effect of a DCJ Operation on the Adjacency Graph

By definition, a double cut-and-join operation acts on two adjacencies of the same genome. In the adjacency graph these can be vertices of the same or of different components. Figures 2.7, 2.8 and 2.9 show some examples for DCJ operations and their effects on the components of the adjacency graph.

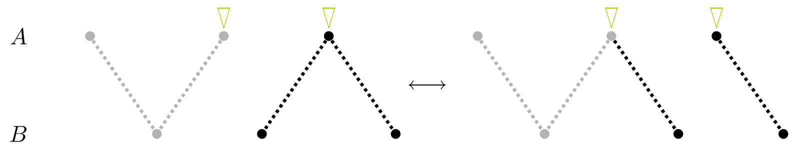


Figure 2.7: Recombination of paths of the adjacency graph: An *AA*-path and a *BB*-path become two *AB*-paths. The *AA*-path could also be an empty chromosome, resulting in simply splitting the *BB*-path.

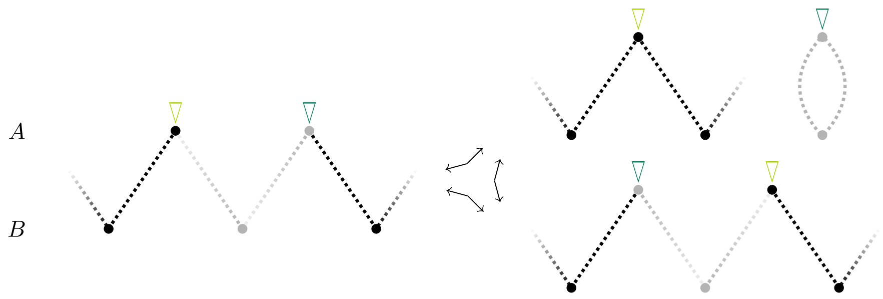


Figure 2.8: Extracting a (trivial) cycle, integrating a (trivial) cycle or reversing a section (of length 2) in the adjacency graph. The component can be any path or cycle.

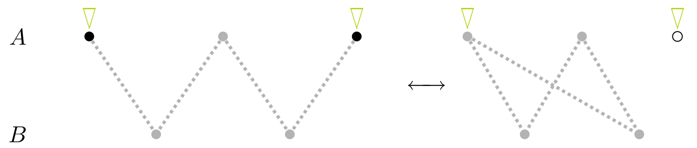


Figure 2.9: Extracting a cycle of length 4 from an *AA*-path, leaving an empty chromosome behind (circularisation). The inverse operation is the linearisation of a chromosome.

Bergeron *et al.* [13] showed that one DCJ operation can change the number of cycles by at most 1 or the number of *AB*-paths by at most 2.

2.3.3 Distance and Sorting

As already outlined in Subsection 1.2.2, we are interested in the number of evolutionary events that separate two genomes, in this case in the number of DCJ operations.

Definition 8 (DCJ Distance Problem): *Given two genomes A and B over $\mathcal{G}$ without duplications, the* DCJ distance *is the minimum number of steps required to sort A into B using only* DCJ operations.

We observed in Subsection 2.2.5 that the adjacency graph of two identical core genomes has only trivial components. Along with the observations on the change in components induced by a DCJ this leads to the findings below.

Theorem 1 (DCJ Distance [13]): *Given two genomes A and B over the same set of markers $\mathcal{G}$ and without duplications, the* DCJ distance $d_{\text{DCJ}}(A, B)$ *is given by:*

$$d_{\text{DCJ}}(A, B) = |\mathcal{G}| - \left(c + \frac{p_{AB}}{2}\right), \tag{2.3}$$

where c and p_{AB} are the number of cycles and AB-paths in $\text{AG}(A, B)$*, respectively.*

Definition 9 (DCJ-sorted): *Two genomes A and B are* DCJ-sorted *if $d_{\text{DCJ}}(A, B) = 0$, or, more generally, if $d_{\text{DCJ}}(A|_{\mathcal{G}}, B|_{\mathcal{G}}) = 0$. In the adjacency graph this means there are only trivial components. Otherwise, the pair of genomes, respectively the adjacency graph, is* DCJ-unsorted.

DCJ operations can be classified according to their impact on the DCJ distance. The impact varies, as there are two ways to rejoin four extremities such that two new adjacencies are created. Under unit cost, a DCJ operation ρ acting on genome A resulting in A' yields:

$$\Delta d_{\text{DCJ}}(A, B, \rho) = d_{\text{DCJ}}(A', B) - d_{\text{DCJ}}(A, B) + 1. \tag{2.4}$$

This reflects the change in distance between A and A' while consuming one step in the sorting scenario. An *optimal* operation reduces the distance between the two given genomes by 1 and thus has $\Delta d_{\text{DCJ}} = 0$. A *neutral* operation does not change the number of cycles or AB-paths and, as the distance is not changed, yields $\Delta d_{\text{DCJ}} = 1$. A *counter-optimal* DCJ operation increases the distance between two genomes, by reducing the number of cycles or AB-paths, and at the same time also consuming a step during the sorting process, hence has $\Delta d_{\text{DCJ}} = 2$.

In Table 2.1 all possible DCJ operations acting on different operands and yielding different resultants are grouped according to their impact on the overall DCJ distance. Some of those examples were already shown in more detail in Figures 2.7, 2.8 and 2.9.

Table 2.1: DCJ operations acting on different component types maintaining (optimal) or increasing (neutral, counter-optimal) the distance. X is a component of arbitrary type. (AA) and (X) mean the component can have length 0.

	operands		→	resultants		Δd_{DCJ}
optimal	X			(X)	cycle	0
	BB	(AA)		AB	AB	
neutral	AA	AA		AA	(AA)	+1
	AA	AB		AB	(AA)	
	AB	BB		AB	BB	
	BB	BB		BB	BB	
	AB	AB		AB	AB	
	AA			AA	AA	
	AB			AB	AA	
	X			X		
counter-optimal	AB	AB		BB	(AA)	+2
	cycle	(X)		X		

Moreover, it was shown in [22] that all components of the adjacency graph can be sorted individually using only optimal DCJ operations, hence that,

$$d_{\mathrm{DCJ}}(A, B) = \sum_{C \in \mathrm{AG}(A,B)} d_{\mathrm{DCJ}}(C), \tag{2.5}$$

where $d_{\mathrm{DCJ}}(C)$ is the minimum number of steps required to sort component C. We will go into more details on sorting with DCJ operations in the ensuing chapter.

Because the DCJ model comprises many modifications, its distance is often smaller than those of others (for example compared to the SC/J distance [47] or the inversion distance [12]). In this thesis, the DCJ model will serve as a basis for the DCJ-*indel* model, the inversion model and the inversion-*indel* model in a way that we seek to compute the number of steps that are necessary for each model in addition to the length of a DCJ sorting scenario.

The purpose of computing is insight, not numbers.
RICHARD HAMMING, 1961

CHAPTER 3

Uniform Sampling of DCJ Sorting Scenarios

In this chapter we concentrate on the *sorting* process in the double cut-and-join (DCJ) model whose operations and distance calculation have been introduced in the previous chapter. As no content modifications are allowed under this model, we disallow unique and duplicated markers. Sorting two genomes then means changing the arrangement of markers in the source genome until it is equal to the target genome.

In 2005, Yancopoulos *et al.* [90] introduced the DCJ operation as well as a sorting algorithm based on the breakpoint graph. The algorithm first performs all translocations (including fissions and fusions), then all inversions are performed and finally all block-interchanges are handled. The latter requires the use of circular intermediates. In 2006, Bergeron *et al.* [13] presented the adjacency graph and a linear time algorithm optimally sorting the two genomes by extracting trivial cycles and in the end splitting any remaining *BB*-path into two trivial paths. For pairs of genomes that are sorted in one step, obviously there is only one optimal sorting scenario. For all other pairs of genomes (with a larger distance) there are several (co-)optimal sorting scenarios.

Picking one random scenario among all possible optimal scenarios reflects the variety of sorting scenarios more adequately. For clarification we call a *sorting sequence* a sequence of events which when applied to the source genome transform it into the target genome while the *sorting scenario* gives the source and target genome as well as all intermediate genomes. We will discover later that, depending on how to describe the steps in a sorting sequence, there may be two or more sequences that yield the same scenario.

In this chapter we present different approaches to find one among the many co-optimal scenarios randomly. In particular, we want each optimal scenario to be equally likely, thus to perform *uniform sampling*. For this, we study previous results on the concept of sorting and the number of ways of sorting [13,22,23,77] and present an algorithm to compute a sorting scenario sampled uniformly among all co-optimal sorting scenarios (without recombinations of *AA*- and *BB*-paths). We present details of obtaining such a scenario using the software `UniMoG` and evaluate our different sampling methods. Subsequently, we discuss further evaluation as well as sampling scenarios with recombinations of *AA*- and *BB*-paths and an alternative approach using Markov chains which was presented in [75,76]. Our combinatoric approach and implementation was published as abstract and presented as a poster [88].

Sorting and Sorting Space

We now study ways of computing an optimal DCJ scenario and propose different randomisation mechanisms.

Constructing Sorted Adjacencies. A simple linear time algorithm for sorting one genome into another by DCJ operations was provided by Bergeron *et al.* in [13]. Given the adjacency graph $\mathrm{AG}(A, B)$, the algorithm walks along each full adjacency of genome B (in the order of input), forming the respective adjacency also in the other genome. In effect, we are extracting trivial cycles from larger components until the only unsorted components are *BB*-paths of length 2. Each of these *BB*-paths has a pair of telomeric adjacencies in genome B which, in the second phase of the algorithm, are split into two trivial paths each. In the end, we have a pair of sorted genomes. Example 3 illustrates this procedure.

Example 3 (Bergeron sorting): Considering an adjacency graph consisting of one *BB*-path of distance 4, the sorting procedure with its two phases is depicted in Figure 3.1. ◇

The sorting in this manner always provides the same sorting sequence, as the processing of B-adjacencies follows the order of input. A first attempt to achieve a different optimal sorting sequence in each run could be to alter the order of visiting the B-vertices. In Section 3.3 we show the results of this method (referred to as vertex-approach) in comparison to other sampling methods. Still, any new component this attempt produces is a trivial cycle or a trivial path, i.e. scenarios that include producing two non-trivial components are not considered. Obviously, this naïve way of sampling, that even omits certain scenarios, is far from uniform.

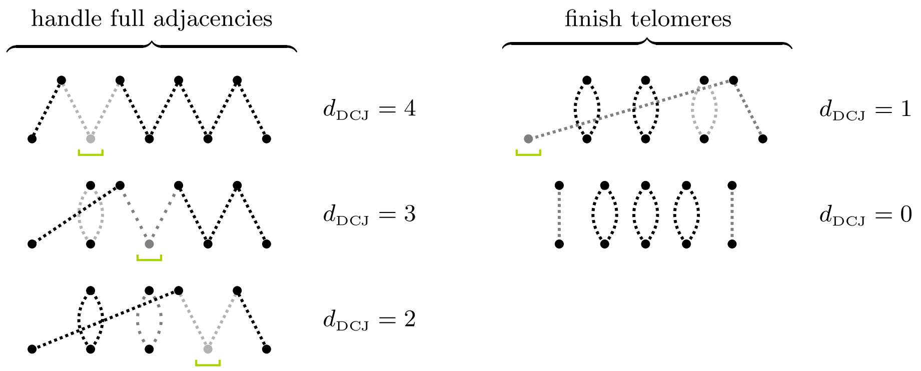


Figure 3.1: The sorting procedure of Bergeron *et al.* [13] consecutively targets adjacencies from B to be produced by DCJ operations that act on vertices of the source genome. Afterwards, the procedure splits all remaining BB-paths, yielding a linear runtime.

We study another method of randomising, which is to sample uniformly among all optimal DCJ operations possible in the next step. This way, also extracting larger cycles or splitting larger paths are included in the sorting and each optimal DCJ operation is chosen with equal probability. This approach is henceforth referred to as edge sampling and is evaluated amongst others in Section 3.3 where differences in the results are pointed out and explained. The following example shows how sampling the next step in a scenario influences the overall probability of a specific scenario.

Example 4 (Edge sampling): We consider an adjacency graph with two BB-paths of lengths 2, resp. 4 amounting to an overall distance of 3. We can apply three optimal DCJ operations to the 4-path in the first step. Components with distance 1 are sorted in only one possible way. Figure 3.2 shows the sorting space, thus all possible sorting scenarios. In the edge approach the probability for the next step is $^1/_{\#\,\mathrm{DCJ\ ops}}$ w.r.t. each (intermediate) adjacency graph. As a result, some scenarios have probability $^1/_4 \cdot {}^1/_3 = {}^1/_{12} = 0.08\bar{3}$ (for instance the highlighted dark path) while others have $^1/_4 \cdot {}^1/_2 = {}^1/_8 = 0.125$ (for instance the highlighted light path) of Figure 3.2. $\diamond$

Looking at the outgoing edges of one sorting step only does not guarantee that each scenario is chosen with equal probability. When unravelling the whole sorting space, thus taking into account also the follow-up operations that choosing this specific edge renders possible, the total number of optimal sorting scenarios becomes apparent. In the example above, nine different scenarios (not sorting sequences) exist for sorting genome A into genome B. Sampling uniformly should then produce each scenario with probability $^1/_9 = 0.\bar{1}$.

Instead of choosing each edge in one step with equal probability, we should therefore choose the next DCJ operation according to the number of different paths that are

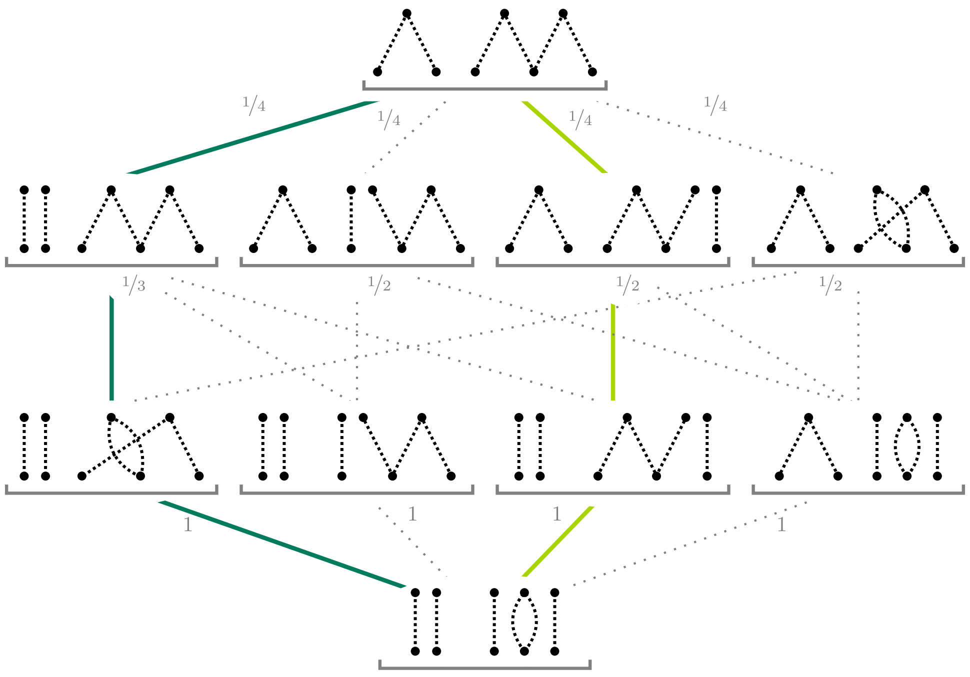


Figure 3.2: All optimal DCJ operations with probabilities for each next step.

possible after this step, i.e. (possibilities of step $k + 1$) /(possibilities of step k). This can be achieved for example by moving bottom-up in the sorting space and adding up the number of edges that lead to one entry. The analysis of the same adjacency graph as before, but for uniform sampling, is given in the following example.

Example 5 (Uniform Sampling): For each node of Figure 3.2, the corresponding node in Figure 3.3 shows in red the number of ways to sort the adjacency graph at this node. Sorting the left component first (dark solid paths), gives three possibilities for the following operations, whereas after performing one of the optimal DCJ operations acting on the other component there are only two possibilities left (light solid path). In the first step, we henceforth choose the dark solid operation with probability $^3\!/_9$ and the light solid operation with probability $^2\!/_9$, instead of $^1\!/_4$ each. Clearly, each of the scenarios now has overall probability $^1\!/_9$. $\diamond$

For further determination of the correct probabilities for choosing a specific operation, we first need to take a closer look at what the whole graph and its vertices look like. From [22,23] (given in Equation 2.5 on page 30) we know that sorting each component individually still gives an optimal sorting scenario but it must be noted that optimal sorting scenarios that recombine an *AA*- and a *BB*-path cannot be produced in this way. Hence, the graph would be incomplete and the probabilities incorrect.

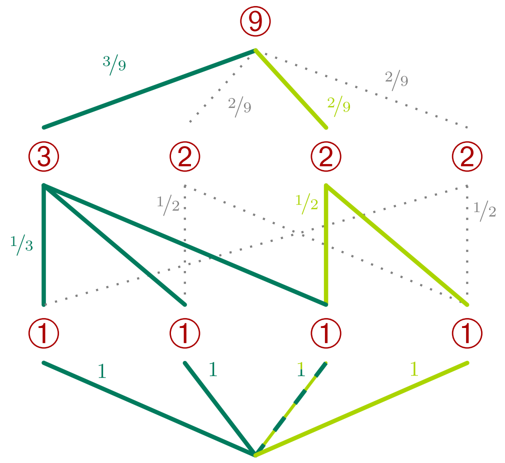


Figure 3.3: The number of possible scenarios in the follow-up (red), determines the actual probabilities for each edge. The colouring of the edges is to highlight specific sorting scenarios.

For a start, we assume that no recombination of even paths is possible. This means, there is either no *AA*-path or no *BB*-path. Otherwise, if both exist, we sort strictly individually. Examples for genomes that have no recombinations of adjacency paths are circular genomes or co-tailed genomes. A discussion on this topic will follow in the last section of this chapter.

3.1 Sampling by Sorting Components Individually

In order to choose the correct probabilities for performing a specific DCJ operation in the first step, we need to know the total number of scenarios as well as the number of possible scenarios for each of the child nodes. Since this can only be determined if we have the information from all recursive child nodes we need to take a look at the sorting space of each adjacency graph.

3.1.1 Solution Space for DCJ Sorting without Recombinations

Given a single unsorted component C of DCJ distance d_C, Braga and Stoye [22] as well as Ouangraoua and Bergeron [77] showed that

$$\mathsf{s}_C = (d_C + 1)^{d_C - 1}$$

gives the number of different ways to sort C. Furthermore, it was also shown that the number of different optimal DCJ sorting scenarios, obtained by sorting each component indvidually, is given by:

$$\mathsf{s} \;=\; d_{\mathcal{U}}! \prod_{C \in \mathcal{U}} \frac{\mathsf{s}_C}{d_C!} \;=\; d_{\mathcal{U}}! \prod_{C \in \mathcal{U}} \frac{(d_C+1)^{d_C-1}}{d_C!}, \tag{3.1}$$

where the set of all unsorted components is denoted by $\mathcal{U}$ and its distance is $d_{\mathcal{U}}$ [22,77].

Example 6: Given an adjacency graph as shown in Figure 3.4, we observe that it has twelve components of which nine are in the set $\mathcal{U}$ of unsorted components (indicated by the dashed line). The total distance is $d_{\mathcal{U}} = 19$. The number of different ways to

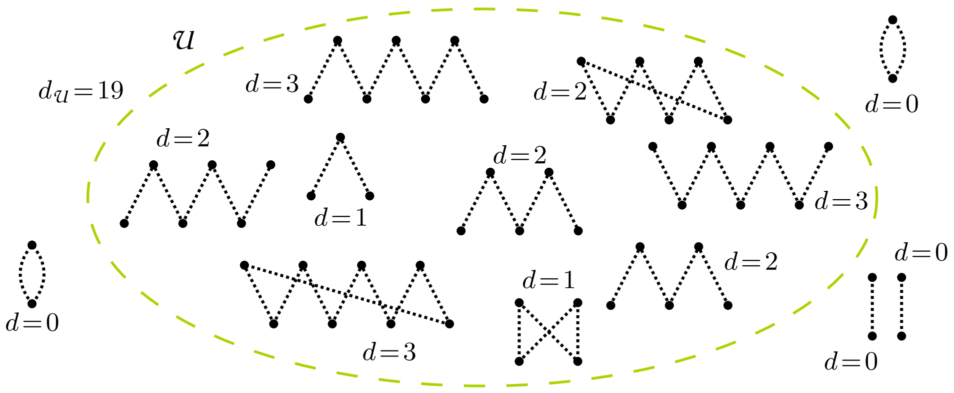


Figure 3.4: An adjacency graph consisting of nine unsorted components (indicated by the dashed line) and three sorted components.

sort this adjacency graph amounts to $\mathsf{s} = 11\,677\,929\,639\,247\,872\,000$. ◇

In order to get the correct edge weights as for example given in Figure 3.3, we need to take into account all operations possible for all intermediate steps. Counting the number of scenarios/edges for every node in the sorting space prior to sampling is very tedious. However, if we know what the intermediate adjacency graphs look like, we can compute s directly. Figure 3.5 shows schematically one node of the sorting

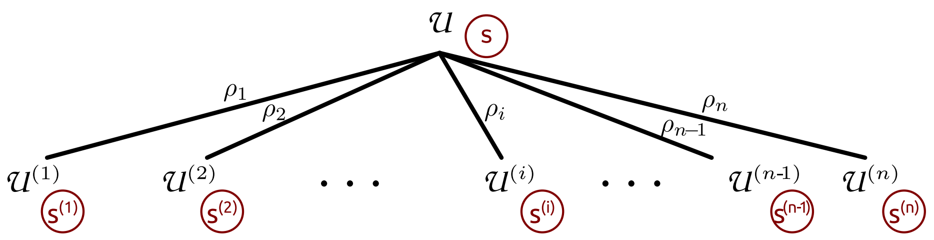


Figure 3.5: Possible optimal DCJ operations $\rho_1, \ldots, \rho_n$ acting on $\mathcal{U}$ produce subsets $\mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(n)}$ that have $\mathsf{s}^{(1)}, \ldots, \mathsf{s}^{(n)}$ scenarios left over, respectively. Then ρ_i has probability $\mathsf{s}^{(i)}/\mathsf{s}$.

space that has the set of unsorted components $\mathcal{U}$ and n optimal DCJ operations that can be applied in the next step of sorting. Let for each DCJ operation ρ_i acting on a component of $\mathcal{U}$ the resulting set of unsorted components be denoted by $\mathcal{U}^{(i)}$. Furthermore, let $\mathsf{s}^{(i)}$ denote the number of sorting scenarios left over for $\mathcal{U}^{(i)}$. Each edge ρ_i has thus probability ${}^{\mathsf{s}^{(i)}}/_{\mathsf{s}}$. But we still need to find out how many and which optimal DCJ operations are possible on components of $\mathcal{U}$.

3.1.2 Bundling Cases with Identical Distance Values

Before we continue, let it be noted that Equation (3.1) immediately gives rise to the following observation which allows us to narrow down the number of distinct edges for each set of unsorted components that is considered.

Observation 2. The type of each individual adjacency graph component is irrelevant, only the distance contributes to the calculation of the number of optimal scenarios. It is therefore sufficient to bundle distance values and consider only one representative for every distinct distance value.

Let $\mathcal{D}$ be the set of distinct distances of $\mathcal{U}$, then we bundle all components with the same distance $d \in \mathcal{D}$ into the set $\mathcal{U}_d \subseteq \mathcal{U}$. Hence, instead of $\mathsf{s}_C = (d_C + 1)^{d_C - 1}$ we can simply write: $\mathsf{s}_d = (d+1)^{d-1}$. With this bundling of components into component subsets with representatives, Equation (3.1) can be re-written in the following way:

$$\mathsf{s} \;=\; d_{\mathcal{U}}! \prod_{d \in \mathcal{D}} \left(\frac{(d+1)^{d-1}}{d!} \right)^{|\mathcal{U}_d|}. \tag{3.2}$$

Example 6 (continued): The components from $\mathcal{U}$ have three distinct distances: $\mathcal{D} = \{1, 2, 3\}$. Thus, we organise the components into subsets $\mathcal{U}_d$ according to their

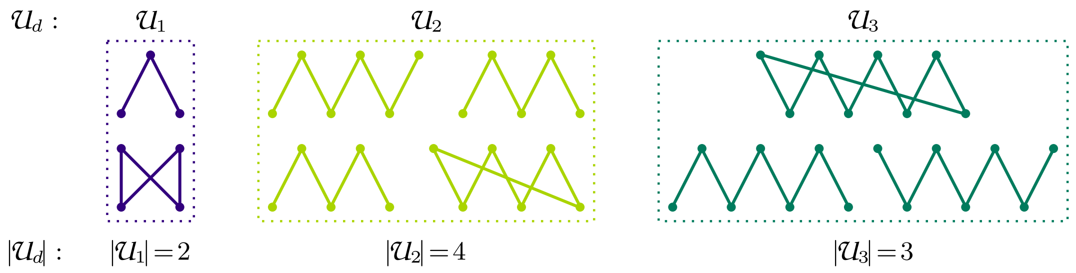


Figure 3.6: The set of unsorted components $\mathcal{U}$ partitioned into subsets $\mathcal{U}_1$, $\mathcal{U}_2$ and $\mathcal{U}_3$.

distance $d \in \mathcal{D}$. As can be seen in Figure 3.6 the sets $\mathcal{U}_1$, $\mathcal{U}_2$ and $\mathcal{U}_3$ have two, four and three elements, respectively. Obviously $\mathcal{U}_1 \cup \mathcal{U}_2 \cup \mathcal{U}_3 = \mathcal{U}$ and we have $\sum_{d \in \mathcal{D}} |\mathcal{U}_d| \cdot d = 2 \cdot 1 + 4 \cdot 2 + 3 \cdot 3 = 19$ which is the value of $d_{\mathcal{U}}$. Furthermore we compute s according to Equation (3.2):

$$\mathsf{s} = 19! \frac{\left((1+1)^{1-1}\right)^2 \cdot \left((2+1)^{2-1}\right)^4 \cdot \left((3+1)^{3-1}\right)^3}{(1!)^2 \cdot (2!)^4 \cdot (3!)^3} = 19! \cdot 96,$$

which is the same as the value calculated using Equation (3.1). ◇

Bundling the components also has an advantage when looking at the changes induced by a single DCJ operation. For instance, let $C \in \mathcal{U}$ be an unsorted component with distance d_C. Extracting a trivial cycle from C produces two new components; a trivial cycle and a smaller component C' with distance $d_C - 1$. This is valid for all elements of $\mathcal{U}_{d_C}$. We thus can reduce the number of times that we need to calculate the number of scenarios for a given node.

Still, computing s for any given set of unsorted components $\mathcal{U}$ of the whole sorting space prior to sampling is very tedious. It also is often redundant, as the change induced by one edge affects only one initial component. Instead, we compute s for each next step of the current adjacency graph, thus for the direct children only. For this, we need to know how many children a node has, and what the resulting set of unsorted components looks like.

For further calculations we will use one representative for each $d \in \mathcal{D}$. Starting with $\mathcal{U}$ we perform all possible optimal DCJ operations separately for the first step on each representative. Each optimal DCJ operation ρ acting on a representative $d \in \mathcal{D}$ results in a set $\mathcal{U}^{(\rho)}$. The number of possible optimal sorting scenarios left after this step is given by $\mathsf{s}^{(\rho)}$ and is weighted by the number of elements that $\mathcal{U}_d$ represents. Figure 3.7 shows schematically the different next steps of the sorting scenario bundled

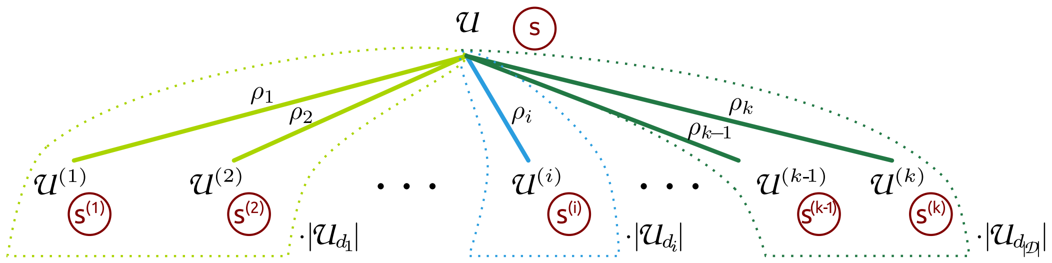


Figure 3.7: Computing the edge weights for different subsets of $\mathcal{U}$.

by distance value, where k is the total number of optimal DCJ operations on representatives. Clearly, the sum of all edge weights is equal to s. This means we sample ρ acting on any element of $\mathcal{U}_d$ according to the probability $(|\mathcal{U}_d| \cdot \mathsf{s}^{(\rho)})/\mathsf{s}$.

3.1.3 Bundling Cases with Identical Change(s) in Distance Value

In the following we will describe how we can not only bundle DCJ operations for a distance representative, but also DCJ operations that result in the same pair of new distance values. We first analyse which DCJ operations that are possible are optimal.

An optimal DCJ operation decreases the DCJ distance by increasing the number of cycles or *AB*-paths. This can be achieved by extracting cycles from components or by splitting a *BB*-path into two *AB*-paths (see also Table 2.1). In any way, a component C of distance d to which a DCJ operation is applied, gets replaced by components C_a and C_b of distance d_a and d_b, respectively.

Performing a double cut-and-join operation on a single component means to cut two A-vertices of the adjacency graph and re-join the four ends. We consider all possible combinations of two cut positions within a single component of distance d. Hence, we take into account all $d + 1$ vertices of that component that belong to genome A (indexed $0, \dots, d$ following the edges in order of traversal, shown in Figure 3.8).

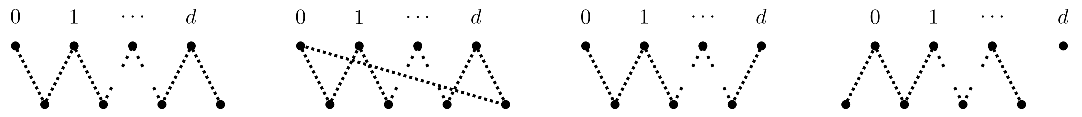


Figure 3.8: Possible cut positions for DCJ operations acting on a single component. In a *BB*-path performing the cut in vertex d corresponds to splitting it into two *AB*-paths.

Splitting a *BB*-path into two *AB*-paths can be modelled by performing the second cut on an empty chromosome, as shown in the rightmost case of Figure 3.8. For an *AA*-path, it is possible to close the complete path into a cycle leaving behind an empty chromosome. It could be shown that, for any pair of cuts in A-vertices that belong to the same component, there is exactly one re-join that is optimal [22].

Observation 3. Let a component C of distance d be split into two components C_a and C_b by an optimal DCJ operation resulting in distances d_a and d_b, respectively. As already observed, the type of component is irrelevant, hence also the component types of C_a and C_b are irrelevant, and only the change in distance value matters.

We bundle possible DCJ operations that produce the same outcomes for $\{d_a, d_b\}$ into one *splitgroup*. For simplicity and without loss of generality we assume that $d_a \geq d_b$.

Proposition 2: *Let C be a single unsorted component of distance d that is split into two components C_a and C_b by an optimal DCJ operation, then there are $\lceil \frac{d}{2} \rceil$ different sets of distances $\{d_a, d_b\}$ of the resulting components.*

Proof. An optimal DCJ operation that splits C into two components C_a and C_b satisfies $d-1 = d_a + d_b$. If d is even, the possible resulting distances of $\{d_a, d_b\}$ are $\{d-1, 0\}$, $\{d-2, 1\}, \ldots \{\frac{d}{2}, \frac{d}{2}-1\}$, yielding $\frac{d}{2}$ different sets. If d is odd, $d_a + d_b$ must be even and an additional case where $d_a = d_b$ can occur, yielding $\frac{d+1}{2}$ different value pairs. $\square$

We use j as the *splitgroup identifier* in order to distinguish the different splitgroups. The splitgroup identifier j that determines the resulting distances $\{d_a, d_b\}$ as $\{d - j, j-1\}$ thus can take $\lceil \frac{d}{2} \rceil$ different values. Furthermore, we need to determine which cut positions belong to which splitgroup. We consider two distinct A-vertices for cut positions.

Observation 4. Given a single unsorted component C of distance d and a vertex v_1 of C, we can have d possible second cut positions (where $v_2 \neq v_1$). Let the resulting components be C_a and C_b with distances d_a and d_b, respectively. Since (v_1, v_2) produces the same result for d_a and d_b as (v_2, v_1), we only consider cut positions (v_1, v_2) where $v_1 < v_2$ (i.e. only $\binom{d+1}{2}$ positions).

For the same value of j multiple choices of cut positions are possible. Although the shape of C_a and C_b may alter, choosing $(0, 1)$, for example, will give the same distances for the resulting components as choosing $(0, d), (1, 2), (2, 3)$ and so on. We obtain two components of distances $d_a = d - j$ and $d_b = j - 1$ by either of the following cases: (i) extracting a cycle of distance d_b, leaving behind a component of distance d_a, (ii) extracting a cycle of distance d_a, leaving behind a component of distance d_b, or (iii) splitting a *BB*-path into two *AB*-paths of lengths d_a and d_b. For cases (i) and (ii) the vertices are v_1 and $v_2 = v_1 + j$. If we have a *BB*-path, cutting the last vertex (index d) and any other vertex splits the path into two *AB*-paths. For this, the second cut is made in vertex $v_2 = j - 1$ or $v_2 = (d-1) - (j-1) = d - j$.

We now determine how many different pairs of cut positions there are for each of the $\lceil \frac{d}{2} \rceil$ splitgroups.

Proposition 3: *Given a single unsorted component C of distance d, there are $\lceil \frac{d}{2} \rceil$ splitgroups with $d+1$ elements each. If d is odd, the last splitgroup $j = \lceil \frac{d}{2} \rceil$ represents the case in which $d_a = d_b$ and has only $1/2 \cdot (d+1)$ elements.*

Proof. Component C is split into two components C_a and C_b with distances d_a and d_b, respectively. We have $d_a + d_b = d - 1$ and $j \in \{1, \ldots, \lceil \frac{d}{2} \rceil\}$.
If d is even, it means $d_a + d_b$ is odd and we distinguish between *BB*-paths and other components (cycles, *AA*- and *AB*-paths).

*C is **not** a BB-path.* Given j, the resulting components have distances $d_a = d - j$ and $d_b = j - 1$ which are obtained by extracting either (i) a cycle C_b from C or (ii) a cycle C_a from C. In case (i), we extract into a cycle the path that is between cut positions $(0, 0+j), (1, 1+j), \ldots, (d-j, d)$, i.e. we have $d - j + 1$ cut positions for extracting a cycle of distance $j - 1$ and leaving behind the rest of the component with distance $d - j$. In case (ii) we derive the same values of d_a and d_b for different component layouts by extracting a cycle with distance $(d-j)$ and leaving the original component at distance $j-1$. Obviously, this can be done $j - 1 + 1 = j$ times.
The number of different DCJ operations splitting component C of distance d into components of distances d_a and d_b using j is given by (i) + (ii), which is always $d - j + 1 + j = d + 1$.

C is a BB-path. The component has A-vertices labelled from $0, \ldots, d-1$. Cases (i) and (ii) each have one possibility less (the cut that includes the last vertex), but instead we now have a third case. In case (iii), the vertex with index d is for splitting the component into two paths with exactly two possibilities: (j, d) and $(d-j, d)$. In total, we have the same number of splitgroup elements as the other components.

If d is odd, it means $d_a + d_b$ is even and again we distinguish between different types of components.

*C is **not** a BB-path.* This case is analogous to the case in which d is even, but the last splitgroup splits s.t. $d_a = d_b$. For this splitgroup, the cut positions (and also the performed operations) from (i) and (ii) are identical and can be counted only once. Hence the splitgroup $j = \lceil \frac{d}{2} \rceil$ has $1/2 \cdot (d+1)$ elements.

C is a BB-path. This case is analogous to the case in wich d is even, but as the last splitgroup splits s.t. $d_a = d_b$, cases (i) and (ii) of the last splitgroup are identical. Also the two cut positions in (iii) are identical and can be counted only once. Hence, the splitgroup $j = \lceil \frac{d}{2} \rceil$ has $1/2 \cdot (d+1)$ elements.

For any type of component each splitgroup has $d + 1$ elements, unless d is odd, in which case the last splitgroup (that is $j = \lceil \frac{d}{2} \rceil$) has only $1/2 \cdot (d+1)$ elements. □

Example 7 (Splitgroup cuts): For a *BB*-path with distance d and for splitgroup $j = 2$, we see in Figure 3.9 the three types of possibilities to derive resulting components with distances $\{d - j, j - 1\} = \{d - 2, 1\}$. For cases (i) and (ii) from the proof of

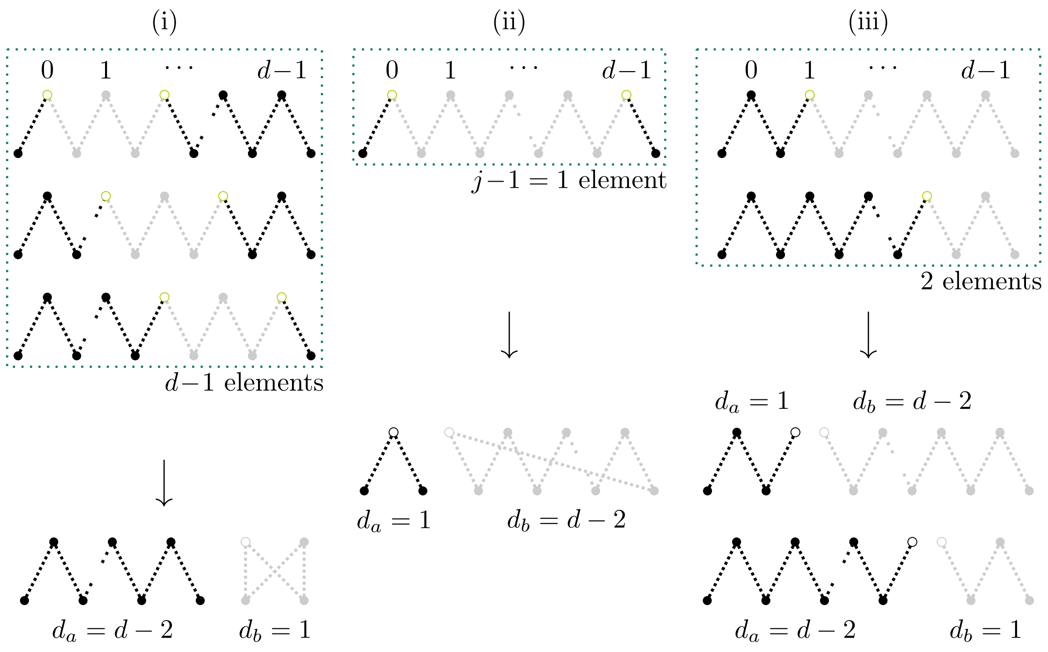


Figure 3.9: Cut positions for splitgroup $j = 2$ in a *BB*-path of distance d. The resulting pair of components all have distances 1 and $d-2$.

Proposition 3, the grey parts are extracted into a cycle, where in (i) the extracted cycle has distance $j-1$ and in (ii) it has distance $d-j$. The size of this splitgroup is (i) $d-2$ + (ii) 1 + (iii) $2 = d+1$. ◇

Adaptation to the Formula

After performing one optimal DCJ operation ρ on a single component C, its distance d is changed and the set of unsorted components is also altered, let it be called $\mathcal{U}^{(\rho)}$. The overall distance $d_{\mathcal{U}}$ is reduced by 1. In order to see in which way s is changed, we need to replace C by C_a and C_b and adapt Equation (3.1) accordingly:

$$\overbrace{\mathsf{s}}^{\mathsf{s}^{(\rho)}} = (\overbrace{d_{\mathcal{U}}}^{d_{\mathcal{U}}-1})! \frac{(d_1+1)^{d_1-1}\cdots \overbrace{(d+1)^{d-1}}^{(d_a+1)^{d_a-1}\cdot(d_b+1)^{d_b-1}} \cdot (d_{|\mathcal{U}|}+1)^{d_{|\mathcal{U}|}-1}}{d_1!\cdots \underbrace{d!}_{d_a!d_b!} \cdot d_{|\mathcal{U}|}!}.$$

We can re-use the previously computed value of s, and we know that $d_{\mathcal{U}}$ is diminished by 1. After performing ρ the number of optimal sorting scenarios that remain is:

$$\mathsf{s}^{(\rho)} = \mathsf{s} \cdot \frac{1}{d_{\mathcal{U}}} \cdot \frac{d!}{d_a!d_b!} \cdot \frac{(d_a+1)^{d_a-1}\cdot(d_b+1)^{d_b-1}}{(d+1)^{d-1}}.$$

Moreover, since an optimal DCJ operation ρ acting on a single component of distance d produces two components with distances $d_a = d - j$ and $d_b = j - 1$, we re-write $\mathsf{s}^{(\rho)}$ as a function of j and d:

$$\begin{aligned} \mathsf{s}_d^j &= \mathsf{s} \cdot \frac{1}{d_{\mathcal{U}}} \cdot \frac{d!}{(d-j)!(j-1)!} \cdot \frac{(d-j+1)^{d-j-1} \cdot (j)^{j-2}}{(d+1)^{d-1}} \\ &= \mathsf{s} \cdot \frac{1}{d_{\mathcal{U}}} \cdot \frac{d \cdot (d-1) \cdots (d-j+1)}{(j-1)!} \cdot \frac{(d-j+1)^{d-j-1} \cdot (j)^{j-2}}{(d+1)^{d-1}}. \end{aligned} \tag{3.3}$$

Obviously, this is done only once per splitgroup and we can further bundle the edges such that not only all components of one set $\mathcal{U}_d$ are bundled but also all DCJ operations of a certain splitgroup of one component. Note that for $d_a = 0$ or $d_b = 0$ the corresponding component is not added to $\mathcal{U}^{(\rho)}$ and thus does not occur in Equation (3.3).

3.1.4 Sampling Weights for a Distance-Splitgroup-Pair

From Equation (3.1) we know how to compute the total number of scenarios s for a given collection of components $\mathcal{U}$ and in Equation (3.3) we see what happens if we change component C of distance d using splitgroup j.

In order to sample uniformly among all scenarios, we have to look at the impact of each possible DCJ operation. For each distance value d and for all possible values of j we have to check the number of optimal scenarios, denoted by s_d^j. We bundle the value of s_d^j for all $c \in \mathcal{U}_d$ and all members of splitgroup j. Hence we compute the weight (representing a *bucket* with a specific j and d) by $w_d^j = \mathsf{s}_d^j \cdot |\mathcal{U}_d| \cdot (d+1)$ (times $1/2$ if d is odd). Figure 3.10 illustrates this for one step for one set $\mathcal{U}_d$.

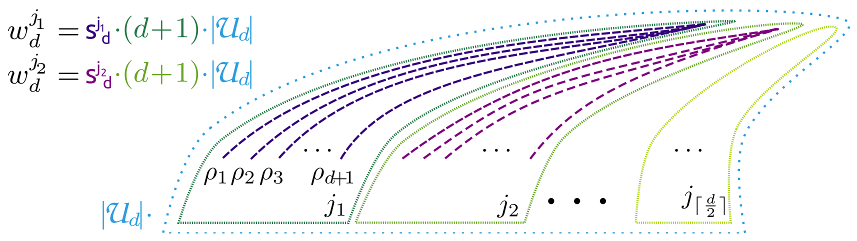

$w_d^{j_1} = \mathsf{s}_d^{j_1} \cdot (d+1) \cdot |\mathcal{U}_d|$
$w_d^{j_2} = \mathsf{s}_d^{j_2} \cdot (d+1) \cdot |\mathcal{U}_d|$
$\rho_1\ \rho_2\ \rho_3\ \ldots\ \rho_{d+1}$
j_1
j_2
$\ldots$
$j_{\lceil \frac{d}{2} \rceil}$
$|\mathcal{U}_d| \cdot$

Figure 3.10: Splitgroups and operations for one representative $\mathcal{U}_d$. The number of splitgroups and their number of elements determine the weight.

All computed weights add up to s and, finally, we can sample a bucket with probability w_d^j/s and derive the values for j and d. Figure 3.11 illustrates this in its first step. Since $\mathsf{s} \cdot 1/d_{\mathcal{U}}$ from Equation (3.3) is common to all possible s_d^j in this step, the crucial factor is the multiplier.

3.1.5 Uniform Sampling of an Optimal DCJ Operation of Size j on a Component of Distance d

Since we operate under a unit cost model, we consider each DCJ operation equally likely. However, when performing one DCJ operation we have to keep in mind that the avalanche it triggers might be of different impact. Assume we are provided with values for j and d. We thus know the size of the resulting components and how many vertices lie in-between the two cut positions that we are to sample. If the first cut position is $x \in \{0, \dots, d\}$ then the second vertex can be either j vertices to the left (in genome A) or j vertices to the right. For the mathematical modulo[1] we choose as possible second cut positions one of the following:

$$\begin{aligned} & y_1 = && (x + j) \mod (d + 1) \\ \text{or} \quad & y_2 = && (d + 1 + x - j) \mod (d + 1). \end{aligned}$$

We consider the two cut positions (x, y_1) and (x, y_2) as equal, since they produce the same distances for the resulting components, and we choose either of them with probability $^1\!/_2$.

Example 8 (Cut positions): Given a *BB*-path of distance $d = 5$ and a first cut position $x = 1$. If we use splitgroup $j = 2$ then $y_1 = ((1 + 2) \bmod (5 + 1) = 3$ and $y_2 = ((5 + 1 + 1 - 2) \bmod (5 + 1) = 5$, thus we can have either $(1, 3)$ or $(1, 5)$ as cut positions, yielding components of distances $\{d - j, j - 1\} = \{1, 3\}$. $\diamond$

At each step of the sampling, we have the value of s (for the current set of unsorted components $\mathcal{U}$) and the individual buckets. After we sampled a bucket, we also have to "backtrace" the bundling. The following example shows how this is done.

Example 6 (continued from p.36): The adjacency graph has unsorted components with distances $\{1, 1, 2, 2, 2, 2, 3, 3, 3\}$, thus one splitgroup for d_1 and d_2 and two for d_3 yielding buckets $\mathsf{s}_1^1, \mathsf{s}_2^1, \mathsf{s}_3^2$ and s_3^2. Here, $\mathsf{s} = 11\ 677\ 929\ 639\ 247\ 872\ 000$. Figure 3.11 illustrates how from s we iteratively derive two sampled cut positions for the sampled component. When sampling operation ρ^*, there are still $\mathsf{s}_2^1 = 409\ 751\ 917\ 166\ 592\ 000$ different optimal ways of sorting the remaining components, or, in other words: we choose ρ^* with probability $^{\mathsf{s}_2^1}\!/_{\mathsf{s}} = 0.035088$. $\diamond$

We developed an algorithm for adjacency graphs which contain either no *AA*-paths

[1] Programming libraries may use the *symmetric* modulo instead, which differs where negative numbers are concerned: $x \bmod y = -(|x| \bmod y)$, for $x < 0$. Hence, adding $d + 1$ in the equation of y_2 circumvents confusion.

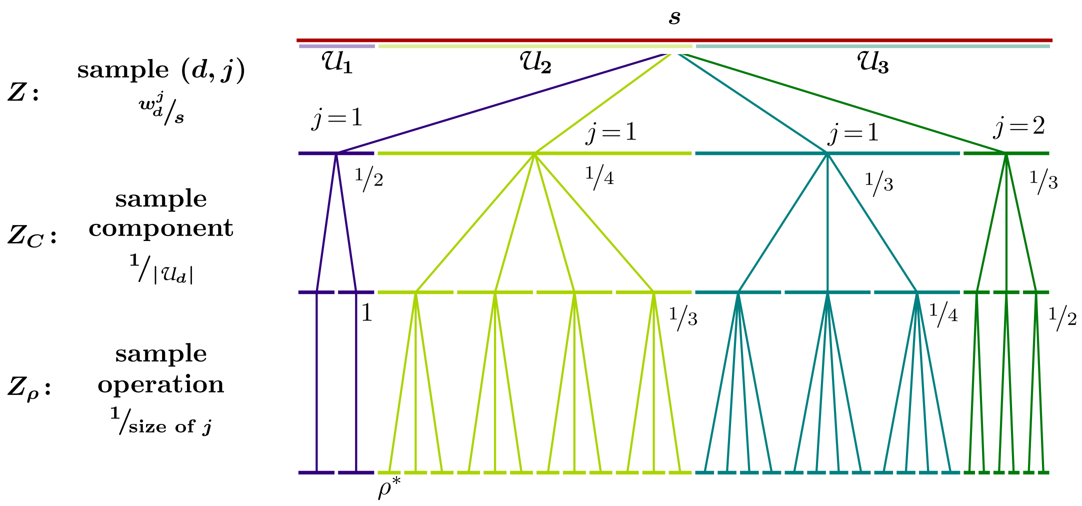


Figure 3.11: Sampling steps that lead to Z_ρ. Step (1): sample bucket that gives values for d and j. Step (2): sample component of distance d. Step (3): sample an optimal DCJ operation of size j.

or no BB-paths or neither of them. Algorithm 1 samples a sorting scenario uniformly from all possible sorting scenarios.

Complexity. Obviously, we need $d_{\mathcal{U}}$ steps to complete the scenario (line 6 of Algorithm 1). First we construct the buckets, which is done once for each distance representative and for each splitgroup. This depends on the number of distinct distance values $|\mathcal{D}|$ (line 8) and the distance value itself (line 9). For one sorting step we compute at most $|\mathcal{D}| \cdot \lceil \frac{\max \mathcal{D}}{2} \rceil$ buckets, where $\mathcal{D}$ is updated after each sorting step (and ultimately decreases).

The worst case for line 8 occurs when $|\mathcal{D}| = |\mathcal{U}|$, i.e. each component of $\mathcal{U}$ has a different distance, or, in other words, each $\mathcal{U}_d$ has exactly one element. Given $d_{\mathcal{U}}$, we would achieve a maximum of $|\mathcal{D}|$ if the smallest distinct distance values occur. The worst case for line 9 would be when d has a high value. It is easy to see that d is maximal when $d = d_{\mathcal{U}}$, thus $|\mathcal{D}| = 1$ (this annuls the worst case of line 8). Combining the worst cases of each loop yields $d_{\mathcal{U}} \cdot |\mathcal{U}| \cdot d_{\mathcal{U}}$ steps, which in practice is always an overestimation.

Algorithm 1 Sampling uniformly among all optimal DCJ scenarios that sort components individually.

```
Input  : The adjacency graph AG$(A, B)$ of two genomes $A$ and $B$.
Output: The sampled optimal DCJ sorting scenario.
 1: $\mathcal{U} \leftarrow$ set of unsorted components of AG$(A, B)$
 2: $\mathcal{D} \leftarrow$ set of distinct distance values of $\mathcal{U}$
 3: $\mathcal{U}_d \subseteq \mathcal{U} \leftarrow$ set of unsorted components of AG$(A, B)$ categorised by distances
 4: $d_{\mathcal{U}} \leftarrow \sum_{C \in \mathcal{U}} d_C$
 5: $\mathsf{s} \leftarrow d_{\mathcal{U}}! \prod_{C \in \mathcal{U}} \frac{(d_C + 1)^{d_C - 1}}{(d_C!)}$
 6: while $d_{\mathcal{U}} > 0$ do
 7:     print current genome $A$
 8:     for all $d \in \mathcal{D}$ do                                  //bucket construction
 9:         for $j \leftarrow 1$ to $\lceil \frac{d}{2} \rceil$ do                     //ordinary splitgroups
10:             $\mathsf{s}_d^j \leftarrow$ Equation (3.3)
11:             $w_d^j \leftarrow |\mathcal{U}_d| \cdot (d + 1) \cdot \mathsf{s}_d^j$     //for each member of $\mathcal{U}_d$ and each split
12:         if $d$ is odd then
13:             last $w_d^j \leftarrow$ (last $w_d^j$) / 2
14:     $(Z, Z_C, Z_\rho) \leftarrow$ sampleOP                        //sample next step, see below
15:     perform $Z_\rho$ on $Z_C$
16:     //$Z_C$ is replaced by $C_a$, $C_b$ with distances $d_a$, $d_b$ and yields $d_{\mathcal{U}} - 1$
17:     update $\mathcal{D}$, $\mathcal{U}_{d_Z}$, $\mathcal{U}_{d_a}$, $\mathcal{U}_{d_b}$, accordingly
18:     $\mathsf{s} \leftarrow \mathsf{s}_d^j$
19: print genome $B$
20: return

    sampleOP:
    $Z \leftarrow$ sample bucket $B$ with probability $w_B/\mathsf{s}$          (distance $d_Z$, splitgroup id $j_Z$)
    $Z_C \leftarrow$ sample component $C$ from $\mathcal{U}_{d_Z}$ with uniform probability
    $Z_\rho \leftarrow$ sample operation on $Z_C$ satisfying $j_Z$
```

3.2 Implementation into UniMoG

The software framework UniMoG[2] [59] provides an easy-to-use manner to compute genomic distances and sorting scenarios for several genome rearrangement models. It comes along with a graphical user interface (GUI) but can also be used as a command line tool. The user can choose among the provided genome rearrangement models:

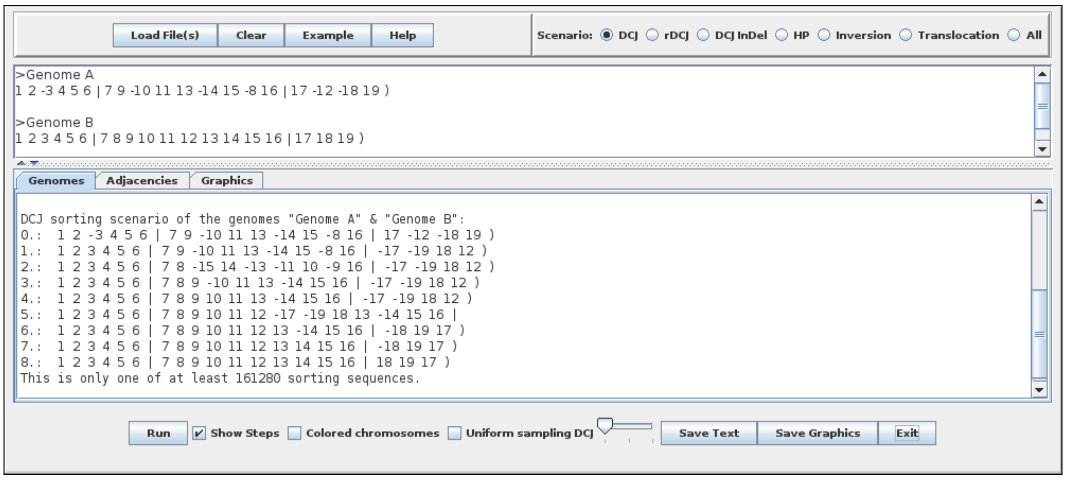
- DCJ
- DCJ-*indel*
- restricted DCJ
- Inversion (via DCJ)
- Translocation (via DCJ)
- HP (via DCJ)

UniMoG accepts two or more genomes in an appropriate format with marker identifiers as input (more details can be found in its help pages). When several genomes are chosen, pairwise genomic distances are computed and output in a table. By default, the software will adjust the genomic content to the chosen model, e.g. remove duplicated or unique markers where necessary. In case the user is interested not only in the distance but also in a sorting scenario of the pairwise comparisons, the check box Show Steps can be ticked ☑. The implementation in UniMoG uses the most efficient existing algorithms for both distance and sorting of each model [59].

The theory of this chapter is embedded in UniMoG. If the check box Uniform Sampling DCJ remains unchecked, the Bergeron sorting algorithm always returns the same sorting scenario. Otherwise, if the option is checked, a scenario sampled uniformly from the sorting space (that has no recombination of *AA*- and *BB*-paths) is output. This is, of course, only considered for the standard DCJ model. Figure 3.12 shows a screenshot where the two check boxes can be seen at the bottom.

For instances where recombinations are not possible, the exact solution of uniform sampling has been integrated into UniMoG. This is the case for any adjacency graph that has no pair of *AA*- and *BB*-paths. Circular genomes as well as all co-tailed genomes meet this condition. Otherwise, if the adjacency graph has at least one *AA*- and at least one *BB*-path, the user is notified that the sampling is no longer uniform among all possible sorting scenarios, rather only uniformly among all non-recombining scenarios and is given the lower bound on the number of scenarios (see at the bottom of the text field in Figure 3.12).

[2] http://bibiserv.cebitec.uni-bielefeld.de/dcj/welcome.html



Scenario: [✓ DCJ] rDCJ DCJ InDel HP Inversion Translocation All

```
>Genome A
1 2 -3 4 5 6 | 7 9 -10 11 13 -14 15 -8 16 | 17 -12 -18 19 )

>Genome B
1 2 3 4 5 6 | 7 8 9 10 11 12 13 14 15 16 | 17 18 19 )
```

```
DCJ sorting scenario of the genomes "Genome A" & "Genome B":
0.:  1 2 -3 4 5 6 | 7 9 -10 11 13 -14 15 -8 16 | 17 -12 -18 19 )
1.:  1 2 3 4 5 6 | 7 9 -10 11 13 -14 15 -8 16 | -17 -19 18 12 )
2.:  1 2 3 4 5 6 | 7 8 -15 14 -13 -11 10 -9 16 | -17 -19 18 12 )
3.:  1 2 3 4 5 6 | 7 8 9 -10 11 13 -14 15 16 | -17 -19 18 12 )
4.:  1 2 3 4 5 6 | 7 8 9 10 11 13 -14 15 16 | -17 -19 18 12 )
5.:  1 2 3 4 5 6 | 7 8 9 10 11 12 -17 -19 18 13 -14 15 16 |
6.:  1 2 3 4 5 6 | 7 8 9 10 11 12 13 -14 15 16 | -18 19 17 )
7.:  1 2 3 4 5 6 | 7 8 9 10 11 12 13 14 15 16 | -18 19 17 )
8.:  1 2 3 4 5 6 | 7 8 9 10 11 12 13 14 15 16 | 18 19 17 )
This is only one of at least 161280 sorting sequences.
```

[✓ Show Steps] Colored chromosomes Uniform sampling DCJ

Figure 3.12: Screenshot of the graphical user interface of UniMoG. Here, the user chose by ☑ to display a sorting scenario but by not ticking the respective box, uniform sampling is not performed.

The drawing of huge random integers in programming has to be considered carefully, as Example 6 shows. In that example for genomes that have a distance of 19, there are a total of 11 677 929 639 247 872 000 non-recombining scenarios and we know there is one sampling bucket that has $s_2^1 = 409\ 751\ 917\ 166\ 592\ 000$ scenarios, which exceeds the java data types `int` (2 147 483 647) or `unsigned int` (4 294 967 295). Looking at real data, for example human and mouse, an optimal sorting sequence has almost thirteen times the distance than our "small" example, namely 246 steps [81]. UniMoG gives a lower bound (the adjacency graph has both *AA*- and *BB*-paths) to the number of optimal sorting scenarios as:

```
 1149558699621369598502808293424335339496547447512690727762325125045023737924
21365104406101297156008757069808685655944388203513652184769096164486777697833
03807166638918618151356626449284707617488539477244399838242362074546906600637
52708561192062245263623515292576610353877119579132172547782425158456791255360
40526569557869511001711095716747311354421096519710115603022118018183910495007
28630756378606453102269500548971468121328632644144846385927901341973609139974
9938184192000000000000000000000000000000000000000000000000000000000000000000
```

(which is a number with 538 digits). However, not only the possibility to store such large numbers but also the availability of a uniform sampler is required. We hence chose the data structure `Xint`[1] to store large numbers.

Naturally, as the numbers get higher and higher, the individual probabilities get lower and lower, thus almost impossible to sample. The upcoming section shows how we can deal with sampling of this magnitude.

[1] Available from: `https://github.com/PeterLuschny/Fast-Factorial-Functions/blob/master/JavaFactorial/src/de/luschny/math/arithmetic/Xint.java`.

3.2.1 Drawing a Huge Random Integer

We want to sample an operation with weight w_d^j/s, where s can be arbitrarily high. The idea is to draw a random integer RV that lies in the interval $]0, \mathsf{s}]$, which is all buckets of w_d^j strung after another. When we draw RV such that $0 \leq RV - 1 < \mathsf{s}$ for some arbitrarily high natural number s and then compare RV to the interval boundaries of added weights w_d^j, we derive the bucket.

Assume we have a (pseudo) random number generator for natural numbers up to nine decimal digits, i.e. in the interval $[0, 10^9[$. Let z be the drawn random variable with $0 \leq z < 10^9$. Then, for a small s, we can draw the number directly from $[0, \mathsf{s}[$ and use $RV = z + 1$ as the final random integer. However, for larger s this is not easily possible, since the computational methods are limited in size.

Here, we employ a rejection method: Blocks of digits are sampled individually, then concatenated to a full number and compared to X. More precisely, let the upper bound of the interval be $X = X[0]X[1]X[2] \cdots X[n]$, consisting of $n + 1$ digits. Our random variable z has the same number of digits as X and is composed of k blocks of nine digits and, if applicable, one block that has r digits. This can be seen in Figure 3.13. The random variable z is then obtained by drawing the nine-digit blocks

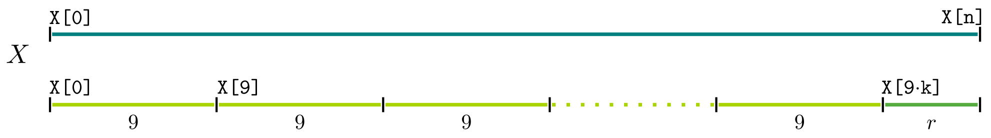


Figure 3.13: An integer X of length $n + 1$, divided into k buckets of length nine and one bucket of length r.

$z_1, \ldots, z_k$ from $[0, 10^9[$ and the r-digit blocks from $[0, 10^r[$ and concatenating it to $z = z_1 +\!\!+ z_1 +\!\!+ \ldots +\!\!+ z_k +\!\!+ z_r$. Finally we derive $RV = z + 1$. Clearly, we have $k = \left\lfloor \frac{n+1}{9} \right\rfloor$ and $r = (n + 1) \bmod 9$. For the huge number, given for the human-mouse-example above, we have $k = 59$ and $r = 7$.

If $RV > X$ we reject the outcome and resample. We do not reject individual values z_i, but all of the values of RV sampled in this attempt. Figure 3.14 shows this (simple) process of drawing a huge random variable RV in more detail.

However, checking for acceptance/rejection can be done while sampling so that we do not continue sampling the huge number when we already know we will have to reject it. That is why we start sampling with the leftmost digits, thus with $B := X[0] \ldots X[8]$ which serves as the first bound for rejecting/accepting. We sample z_1 for the leftmost bucket from $[0, 10^9[$.

$\boldsymbol{z_1 > B}$ We know that even if we continue with the other buckets, in the end we will

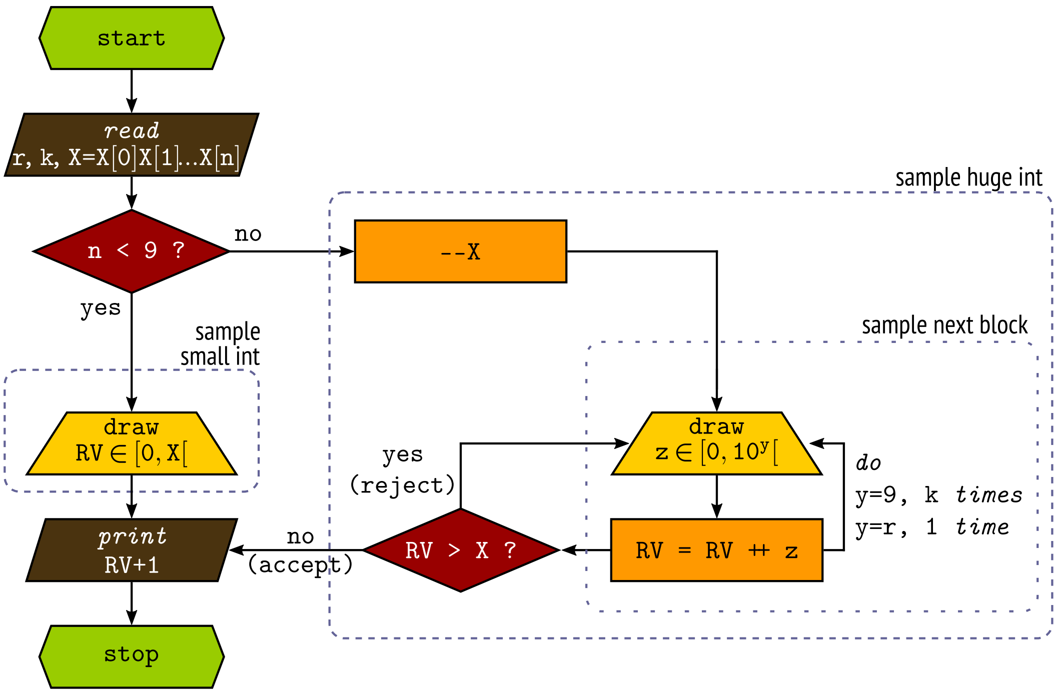


Figure 3.14: Flow diagram of sampling a (huge) natural number RV from $[1, X]$.

have $RV > X$, thus we can already stop here, reject z_1 and start anew.

$\boldsymbol{z_1 < B}$ In this case we can accept all outcomes of $z_2, \ldots, z_k, z_r$ since $RV < X$ will always hold.

$\boldsymbol{z_1 = B}$ If z_1 hits the upper bound, we need to check for the next outcome of z_2. For each of the following buckets, we record a B_i as an acceptance bound. When $z_i > B_i$ we reject everything and start anew. If we hit the upper bound $z_i = B_i$, we continue to check the next sampled bucket, until we can accept or reject the outcomes, or reach $z_r \leq B_r$.

For example, in the first bucket we draw from $[0, 999\,999\,999]$ and since $B = 114\,955\,869$ the probability of z_1 being rejected is roughly 0.885.

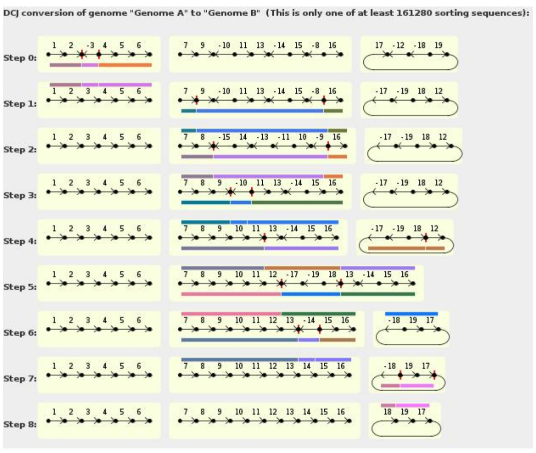
Note. In principle, this procedure could run forever, that is when the drawn RV is forever bigger than X, which in practice will not happen.

3.2.2 Example Output

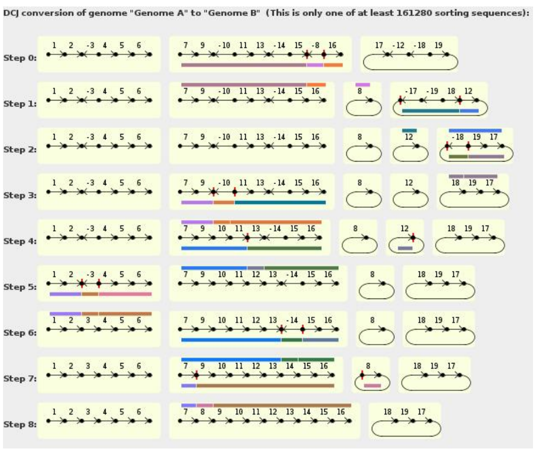
Besides a textual output shown in Figure 3.12 (the same as given when using the command-line), `UniMoG` also provides a visual scenario for the operations. Here, the linear or circular chromosomes are depicted with their markers. A cut is indicated by a vertical red bar. The section of the chromosome that will be affected by this operation is coloured below the chromosome and in the next step, when the new arrangement is shown, the same regions are colour-coded above the chromosome.

For the two given genomes $A = \{(\circ, 1, 2, -3, 4, 5, 6, \circ), (\circ, 7, 9, -10, 11, 13, -14, 15, -8, 16, \circ), (17, -12, -18, 19)\}$ and $B = \{(\circ, 1, 2, 3, 4, 5, 6, \circ), (\circ, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, \circ), (17, 18, 19)\}$ from Figure 3.12 the `UniMoG` output gives a scenario with distance 8 as well as the lower bound of 161 280 optimal sorting scenarios. Six of these are depicted in Figure 3.15. Clearly it can be seen that the number of chromosomes can vary, e.g. scenarios (ii) and (iii) have at one time two extra circular intermediates (five chromosomes in total), while scenarios (i) and (vi) incorporate one chromosome into another thus having only two chromosomes temporarily. Also one can see from chromosome 1 that it needs only one DCJ operation to be sorted and the step in which this single operation can occur also varies.

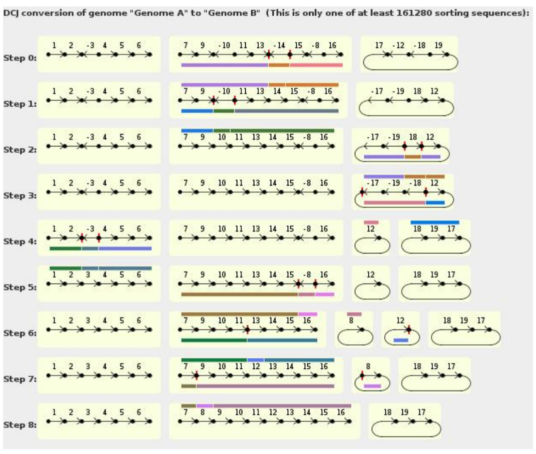
(i) Scenario 1, standard sorting when no sampling is used.


DCJ conversion of genome "Genome A" to "Genome B" (This is only one of at least 161280 sorting sequences):
Step 0:
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:

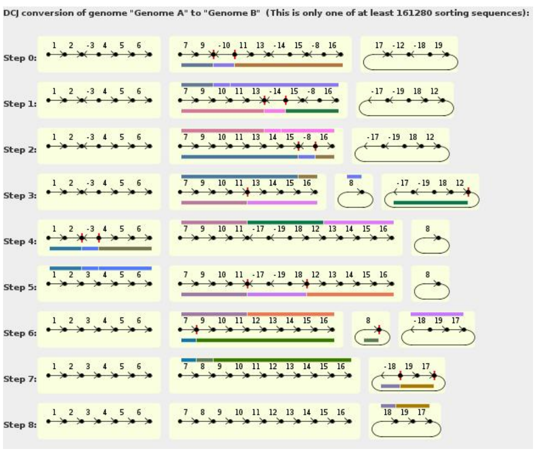
(ii) Scenario 2


DCJ conversion of genome "Genome A" to "Genome B" (This is only one of at least 161280 sorting sequences):
Step 0:
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:

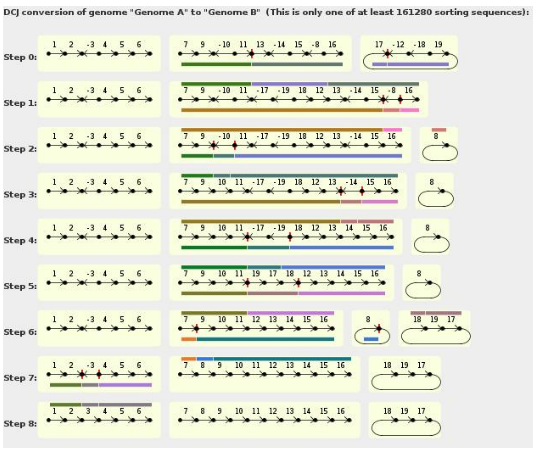
(iii) Scenario 3


DCJ conversion of genome "Genome A" to "Genome B" (This is only one of at least 161280 sorting sequences):
Step 0:
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:

(iv) Scenario 4


DCJ conversion of genome "Genome A" to "Genome B" (This is only one of at least 161280 sorting sequences):
Step 0:
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:

(v) Scenario 5


DCJ conversion of genome "Genome A" to "Genome B" (This is only one of at least 161280 sorting sequences):
Step 0:
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:

(vi) Scenario 6

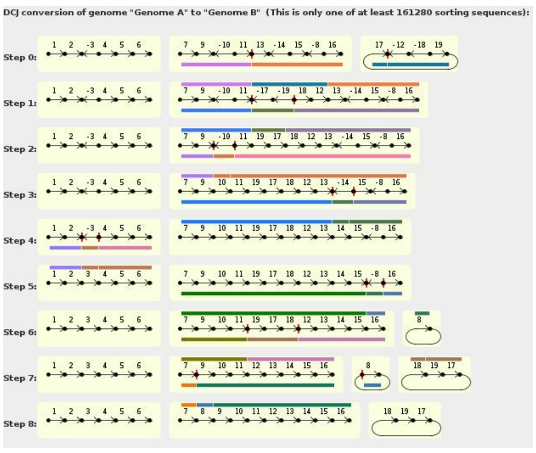

DCJ conversion of genome "Genome A" to "Genome B" (This is only one of at least 161280 sorting sequences):
Step 0:
Step 1:
Step 2:
Step 3:
Step 4:
Step 5:
Step 6:
Step 7:
Step 8:

Figure 3.15: Several different sorting scenarios for the same pair of genomes, namely $A = (\circ, 1, 2, -3, 4, 5, 6, \circ), (\circ, 7, 9, -10, 11, 13, -14, 15, -8, 16, \circ), (17, -12, -18, 19)$ and $B = (\circ, 1, 2, 3, 4, 5, 6, \circ), (\circ, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, \circ), (17, 18, 19)$.

3.3 Evaluation

In the following, we evaluate our uniform sampling method against the more naïve sampling approaches proposed in the beginning of this chapter. Apart from real data, we also chose artificial data created to represent extreme cases and to emphasize the (dis)similarities of the methods.

As the non-uniform methods were implemented re-using existing data structures and method calls of `UniMoG`, we forgo a comparison of all methods concerning time or space complexity and instead concentrate on the sampled scenarios.

For the purpose of this study, we rely on circular chromosomes only, which means the adjacency graph has only cycles. Note that sorted cycles and their vertices are not considered for sampling.

3.3.1 Overview of Sampling Methods

For the evaluation and comparison of uniform sampling, three other sampling methods were implemented in order to show the differences and the importance to not sample naïvely.

`vertex`: A random version of Bergeron sorting (by trivial cycle extraction).

`edge`: Randomly chooses a next step (each step has equal probability).

`split`: Chooses a random split group of the next step. We first sample a cycle C_i (weighted by the number of edges). Then we choose one splitgroup j_i for C_i at random (weighted by the number of elements). For the sampled splitgroup j_i we then sample a pair of vertices of C_i satisfying j_i.

3.3.2 Real Data

For evaluation with real data we chose the γ-proteobacteria data set used in [15] which they give in their appendix, derived from [7]. The species and sources are listed in Table 3.1. All of these genomes have one circular chromosome. We use four gene orders (see Table 3.2) and refer to them with their abbreviations, as given by Table 3.1.

Prior to sampling multiple scenarios, we analyse the properties of each pairwise comparison (see Table 3.3). As can be seen in the leftmost table, the comparisons of gene orders `E` and `stm` as well as `stt` and `sty` are trivial, as their adjacency graph has only one unsorted cycle of distance 1 (obviously the only DCJ operation that is performed is chosen with probability 1). We therefore focus on the non-trivial comparisons. Although `stm` and `stt` have the same DCJ distance as `sty` and `E` (second

Table 3.1: Selected species for a comparison of γ-proteobacteria [7].

Abbr.	Species	NC number	Source
`stm`	*Salmonella typhimurium* LT2	`NC_003197`	[69]
`stt`	*S. enterica* subsp. *enterica* serovar Typhi Ty2	`NC_004631`	[38]
`sty`	*S. enterica* subsp. *enterica* serovar Typhi str. CT18	`NC_003198`	[78]
`ecc*`	*Escherichia coli* CFT073	`NC_004431`	[86]
`eco*`	*Escherichia coli* K12	`NC_000913`	[17]
`ecs*`	*Escherichia coli* O157-H7	`NC_002695`	[57]
`ece*`	*Escherichia coli* O157:H7 EDL993	`NC_002655`	[80]

* species have the same gene order, combined to `E`

Table 3.2: Marker orders for the selected γ-proteobacteria species [15].

```
>sty:   1   2   4  -3   5   6  -8  -9  -7  10  )
>stm:   1   2  -4  -3   5   6   7   8   9  10  )
>stt:   1  -5   3  -4  -2   6  -8  -9  -7  10  )
>E  :   1   2   3   4   5   6   7   8   9  10  )
```

table of Table 3.3), there are more optimal scenarios for the latter (rightmost table of Table 3.3).

Table 3.3: DCJ distances (left), distances of unsorted cycles in the adjacency graphs (centre) and number of different scenarios (right) for pairwise genome comparisons of γ-proteobacteria.

	`sty`	`stt`	`E`
`stm`	4	5	1
`sty`	-	1	5
`stt`	-	-	6

	`sty`	`stt`	`E`
`stm`	1,3	1,1,3	1
`sty`	-	1	2,3
`stt`	-	-	1,2,3

	`sty`	`stt`	`E`
`stm`	64	320	1
`sty`	-	1	480
`stt`	-	-	2880

Sampling Optimal Sorting Scenarios for `sty`-`stm` Comparison

Let us illustrate the differences of each of the vertex, edge, split and uniform method with the help of a detailed example. In order to make it feasible for stepwise elaboration, we choose the comparison of `sty` and `stm` as it yields the smallest non-trivial distance and smallest number of optimal scenarios in our set of comparisons ($d_{\mathrm{DCJ}} = 4$, resp. $\mathtt{s} = 64$). The full adjacency graph of the two genomes is shown in Figure D.12 in Section D of the appendix. The two non-trivial cycles have distance 1 (thus two vertices in A and two in B) and distance 3 (thus four vertices in A and four in B). For simplicity we refer to these as C_1, resp. C_3. During sampling, when a cycle of distance 2 is created, we refer to it as C_2. Likewise, splitgroups $j = 1$ and $j = 2$ are referred to as j_1 and j_2, respectively. The different sets of cycles that emerge from DCJ operations are shown in Figure 3.16. A list of all possible sorting scenarios using the sampling methods described below is given in Section D of the appendix.

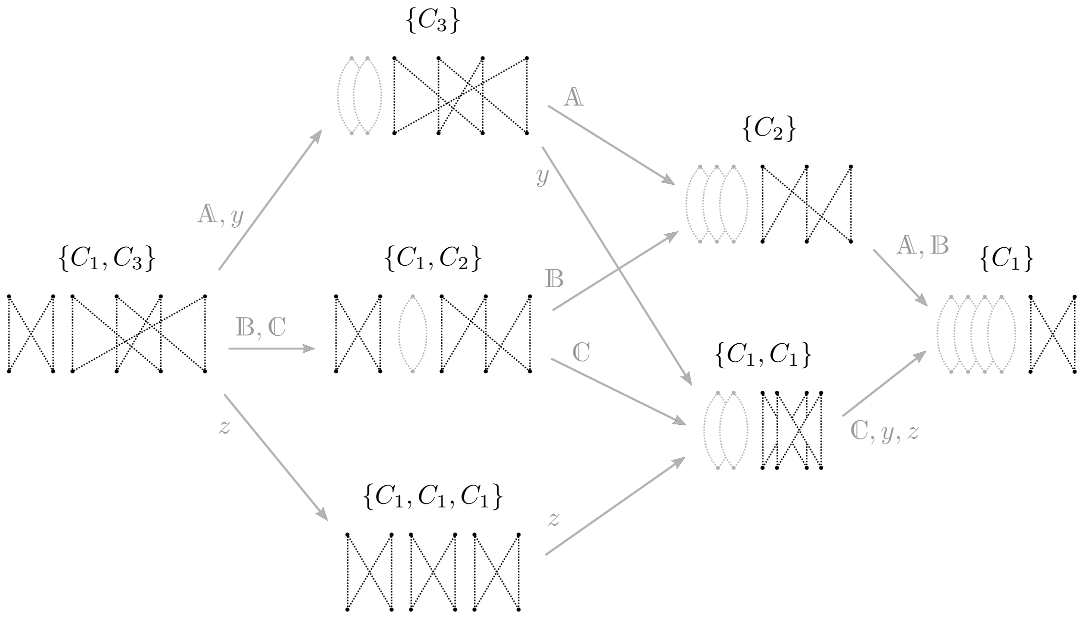


Figure 3.16: Compacted sorting space of non-trivial cycles of the `sty-stm` adjacency graph. Left: initial cycle set, right: penultimate cycle set. $\mathbb{A}$, $\mathbb{B}$, $\mathbb{C}$, y and z label the different ways (and the respective edges) for sorting.

Uniform sampling. Each of the 64 different optimal sorting scenarios has a probability of $1/64 = 0.015625$ for uniform sampling. When sampling 150 000 scenarios we hence compare each of the following methods to $150\,000/64 \approx 2344$ hits per scenario.

Vertex sampling. In each step a trivial cycle is extracted. Hence, C_3 cannot be split into two C_1-cycles. Sorting scenarios from categories y and z can thus not be obtained. In the first step of the vertex-approach there are six vertices in non-trivial cycles, hence a vertex is chosen with probability $1/6$ and the corresponding adjacency is formed in the intermediate genome. We can extract a trivial cycle from C_3 in four different positions yielding four different intermediate genomes. On the other hand, choosing either of the two adjacencies of C_2 produces two trivial cycles, thus the same intermediate genome is produced while sampling one or the other vertex of that cycle. The resulting intermediate genome is thus effectively chosen twice as often. However, this affects the probability only when in the same step there are larger cycles present. Otherwise, if there are only 4-cycles, they all have the same probability and no intermediate genome is given preference to the others. The probabilities of scenarios of each category using vertex-sampling are then:

$$\mathbb{A}: \ {}^2/_6 \cdot {}^1/_4 \cdot {}^1/_3 = 0.02\overline{7}, \quad \mathbb{B}: \ {}^1/_6 \cdot {}^2/_5 \cdot {}^1/_3 = 0.0\overline{2} \quad \text{and} \quad \mathbb{C}: \ {}^1/_6 \cdot {}^1/_5 \cdot {}^1/_2 = 0.01\overline{6}.$$

The number of scenarios that fall in each category as well as the intermediate genomes of each scenario can be looked-up in the appendix (see Section D). If we sample 150 000 scenarios we expect a scenario that falls into category $\mathbb{A}$, $\mathbb{B}$, or $\mathbb{C}$ to occur around 4166, 3333 and 2500 times, respectively. As can be seen in Figure 3.17, the implemented vertex sampling achieves these results. As this approach cannot produce

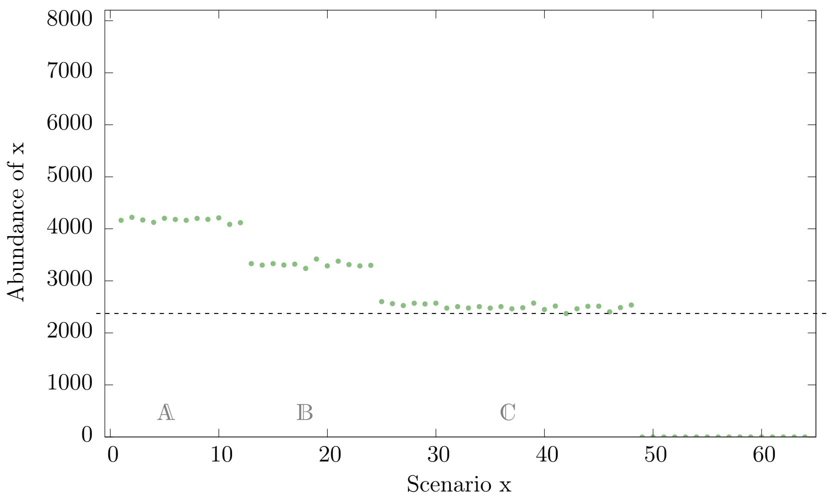


Figure 3.17: Frequency of the 64 possible sorting scenarios among 150 000 samples of the `vertex` sampling. The 2344-mark is indicated by a dashed line. The scenarios that this approach can obtain are to the left in the picture, those that this approach cannot obtain have a higher id.

any sorting scenario of category y and z these scenarios have a frequency of 0. The order of scenarios corresponds to the catogeries in alphabetic order.

Edge sampling. For the edge-approach the way in which a 4-cycle is split into two trivial cycles is irrelevant, only the resulting set of cycles matters. In the case of `sty` and `stm`, for the first step we have ${}^1/_2 \cdot (1 + 1) + {}^3/_2 \cdot (3 + 1) = 7$ possible resulting adjacency graphs. All scenarios choosing the smallest splitgroup (j_1) correspond to a scenario sampled in the same way as the previous method (categories $\mathbb{A}$, $\mathbb{B}$ or $\mathbb{C}$). Initially, C_1 and C_3 are chosen with probability ${}^1/_7$ and ${}^6/_7$, respectively. C_1 has only one splitgroup and one set of sampled vertices. Afterwards, (thus for C_3) two splitgroups exist (chosen with probability ${}^4/_6$ and ${}^2/_6$). Choosing either j_1 or j_2 leaves a cycle that has only one splitgroup. When choosing to act on C_3 first, we can either choose j_1 or j_2. In the former case we can then choose to act on C_1 with probability $1/4$ or on C_2 with probability $3/4$ since we can only extract trivial cycles and there are four possible resulting adjacency graphs. In the latter case we are left with three 4-cycles, all of them equally probable to be split in the next step. The probabilities of sampling a scenario by category are:

$$\begin{array}{rccccccccccccccccc}
 & c & \cdot & j & \cdot & v & \cdot & c & \cdot & j & \cdot & v & \cdot & c & \cdot & j & \cdot & v & & \\
\mathbb{A}: & 1/7 & \cdot & 1 & \cdot & 2/2 & \cdot & 6/6 & \cdot & 4/6 & \cdot & 1/4 & \cdot & 1 & \cdot & 1 & \cdot & 1/3 & = 1/126 & \approx 0.00793 \\
y: & | & \cdot & | & \cdot & | & \cdot & | & \cdot & 2/6 & \cdot & 1/2 & \cdot & 1/2 & \cdot & 1 & \cdot & 2/2 & = 1/84 & \approx 0.01190 \\
\mathbb{B}: & 6/7 & \cdot & 4/6 & \cdot & 1/4 & \cdot & 1/4 & \cdot & 1 & \cdot & 2/2 & \cdot & 1 & \cdot & 1 & \cdot & 1/3 & = 1/84 & \approx 0.01190 \\
\mathbb{C}: & | & \cdot & | & \cdot & | & \cdot & 3/4 & \cdot & 1 & \cdot & 1/3 & \cdot & 1/2 & \cdot & 1 & \cdot & 2/2 & = 1/56 & \approx 0.01786 \\
z: & | & \cdot & 2/6 & \cdot & 1/2 & \cdot & 1/3 & \cdot & 1 & \cdot & 2/2 & \cdot & 1/2 & \cdot & 1 & \cdot & 2/2 & = 1/42 & \approx 0.02381
\end{array}$$

When sampling 150 000 scenarios for `sty` and `stm` with the edge-approach, we thus expect around 1190, 1786, 1786, 2678 and 3571 hits per scenario, respectively. As

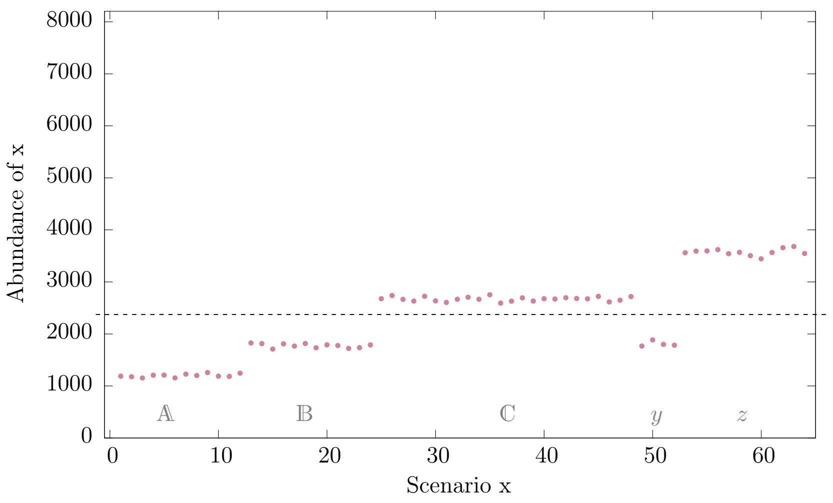


Figure 3.18: Frequency of the 64 possible sorting scenarios sampled with the `edge` approach for 150 000 samples. The order of scenarios matches that of Figure 3.17. The 2344-mark is indicated by a dashed line.

can be seen in Figure 3.18, the implemented edge sampling achieves these results. All scenarios occur, but with a deviation from the 2344-mark of the uniform expectation. The order of scenarios is the same as for Figure 3.17 (hence scenarios from y and z, that use splitgroup j_2, are to the right of the picture).

Split sampling. As C_1 has two vertices in A and C_3 has four vertices, they are chosen with probability $2/6$ and $4/6$, respectively. C_1 has only one splitgroup and C_3 has j_1 with four elements and j_2 with two elements. Categories $\mathbb{A}$, $\mathbb{B}$ and $\mathbb{C}$ are analogous to those from above, but yield different probabilities as the components are chosen in a different way. After sorting C_1 we have two possibilities: scenarios of category $\mathbb{A}$ or y. Either are chosen according to the splitgroup probabilities ($2/3$ and $1/3$, respectively). Otherwise, in the first step we choose a DCJ operation acting on C_3 with probability $4/6$. Here, we sample splitgroup j_1 with probability $2/3$ (categories $\mathbb{B}$ or $\mathbb{C}$) or j_2 with probability $1/3$ (category z). After two steps, all cycles have distance 2 or smaller, hence only one splitgroup exists for each cycle. However, the cycles are still chosen according to their relative number of vertices. In conclusion, we derive the following probabilities:

$$\begin{array}{rcccccccccccc}
 & c \cdot j \cdot v & \cdot & c \cdot j \cdot v & \cdot & c \cdot j \cdot v & & \\
\mathbb{A}: & 2/6 \cdot 1 \cdot 1 & \cdot & 4/4 \cdot 2/3 \cdot 1/4 & \cdot & 1/1 \cdot 1 \cdot 1/3 & = 1/54 & = 0.0\overline{185} \\
y: & | \cdot | \cdot | & \cdot & | \cdot 1/3 \cdot 2/4 & \cdot & 1/2 \cdot 1 \cdot 1 & = 1/36 & = 0.02\overline{7} \\
\mathbb{B}: & 4/6 \cdot 2/3 \cdot 1/4 & \cdot & 2/5 \cdot 1 \cdot 1 & \cdot & 1/1 \cdot 1 \cdot 1/3 & = 2/135 & = 0.0\overline{148} \\
\mathbb{C}: & | \cdot | \cdot | & \cdot & 3/5 \cdot 1 \cdot 1/3 & \cdot & 1/2 \cdot 1 \cdot 1 & = 1/90 & = 0.0\overline{1} \\
z: & | \cdot 1/3 \cdot 1/2 & \cdot & 1/3 \cdot 1 \cdot 1 & \cdot & 1/2 \cdot 1 \cdot 1 & = 1/54 & = 0.0\overline{185}
\end{array}$$

We implemented this method, and, as above, sampled 150 000 scenarios of the genome comparison `sty` vs. `stm`. With the above probabilities, we expect each scenarios of the categories $\mathbb{A}$, y, $\mathbb{B}$, $\mathbb{C}$ and z to occur around 2778, 4167, 2222, 1667 and 2778 times, respectively. As can be seen in Figure 3.19, the implemented split sampling achieves these results.

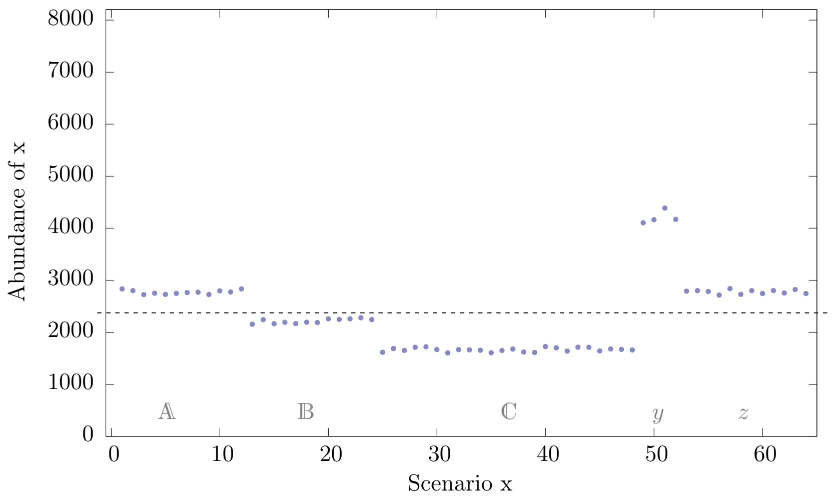


Figure 3.19: Frequency of the 64 possible sorting scenarios sampled with the `split` approach for 150 000 samples. The order of scenarios matches that of Figure 3.17. The 2344-mark is indicated by a dashed line.

An overview of the results of all four methods for the `sty-stm` comparison can be seen in Figure 3.20 (i) whose boxplot is shown in Figure 3.21 (i). In these pictures, the sorting scenarios are ordered lexicographically within the left (L) part that all methods can obtain and right (R) part that cannot be obtained by the vertex-approach. (Figure D.13 on page 202 in the appendix shows (i) with the scenarios sorted according to their category: $\mathbb{A}, \mathbb{B}, \mathbb{C}$ and x and y).

All Pairwise Comparisons for the γ-Proteobacteria Data Set

Figure 3.20 shows for each pair of species the frequency of each possible scenario for all four presented methods. It is easy to see that the pattern of the `sty-stm` comparison is also found in the distribution of frequencies of the other comparisons. Oddly, all comparisons have the distinguishable break between the j_1 and j_2 at the same proportion of scenarios. This is due to the fact that all comparisons have exactly one cycle of distance 3.

Note that the scales in each comparison differ. The x-axis corresponds to the number of scenarios given in Table 3.3. Since all comparisons were run 150 000 times, the average of each comparison is different which reflects in the scale of the y-axis.

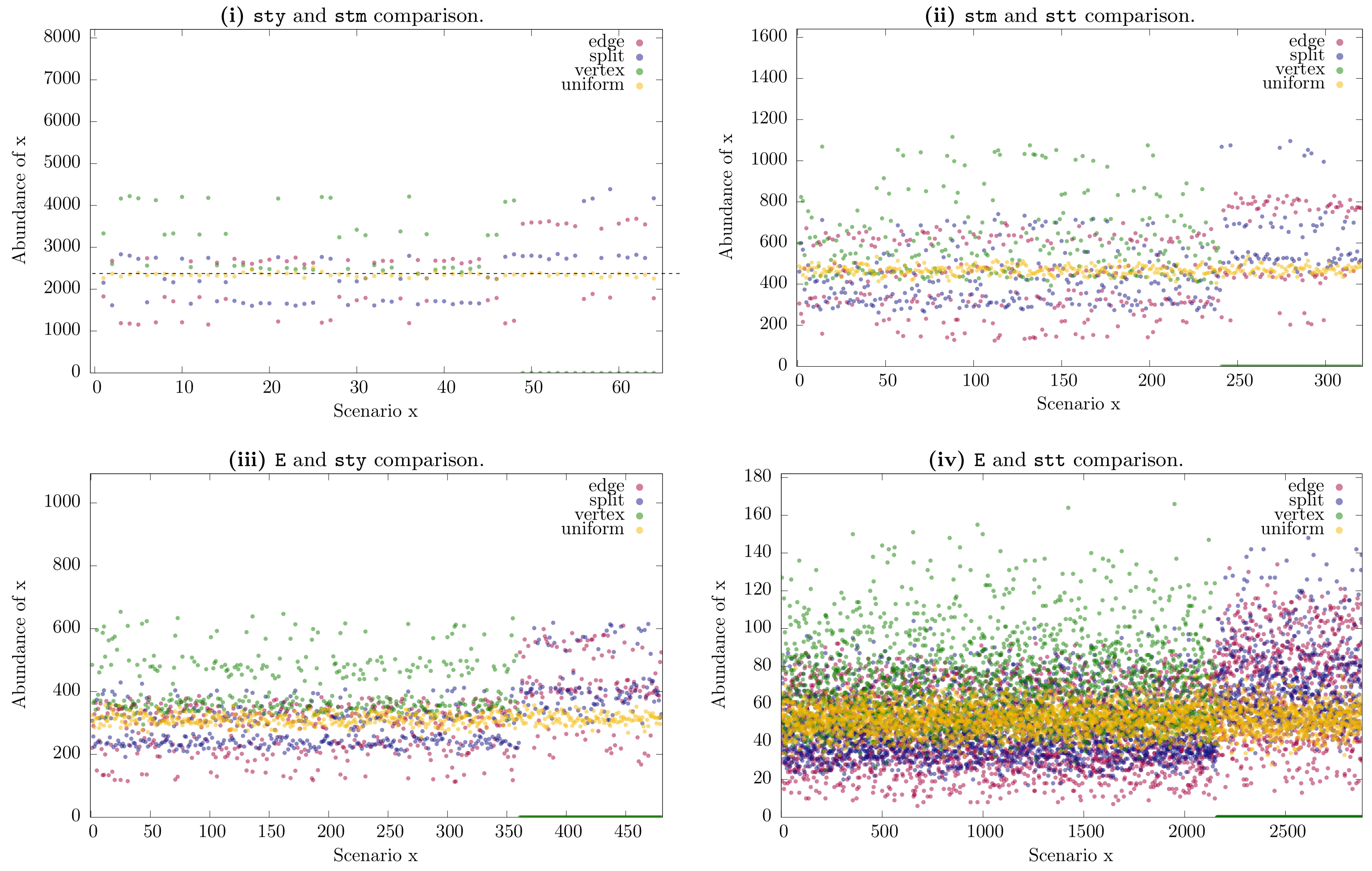


Figure 3.20: Distribution of sorting scenarios of pairs of γ-proteobacteria from `sty`, `stm`, `stt` and `E`. Comparison of `vertex`, `edge`, `split` and `uniform` sampling methods where for each comparison and each method 150 000 scenarios were sampled. Scenarios that are obtained by all methods are to the left in each diagram, those that cannot be obtained by the vertex-approach are to the right in each diagram (frequency 0). Scenarios are ordered lexicographically within the left and right part.

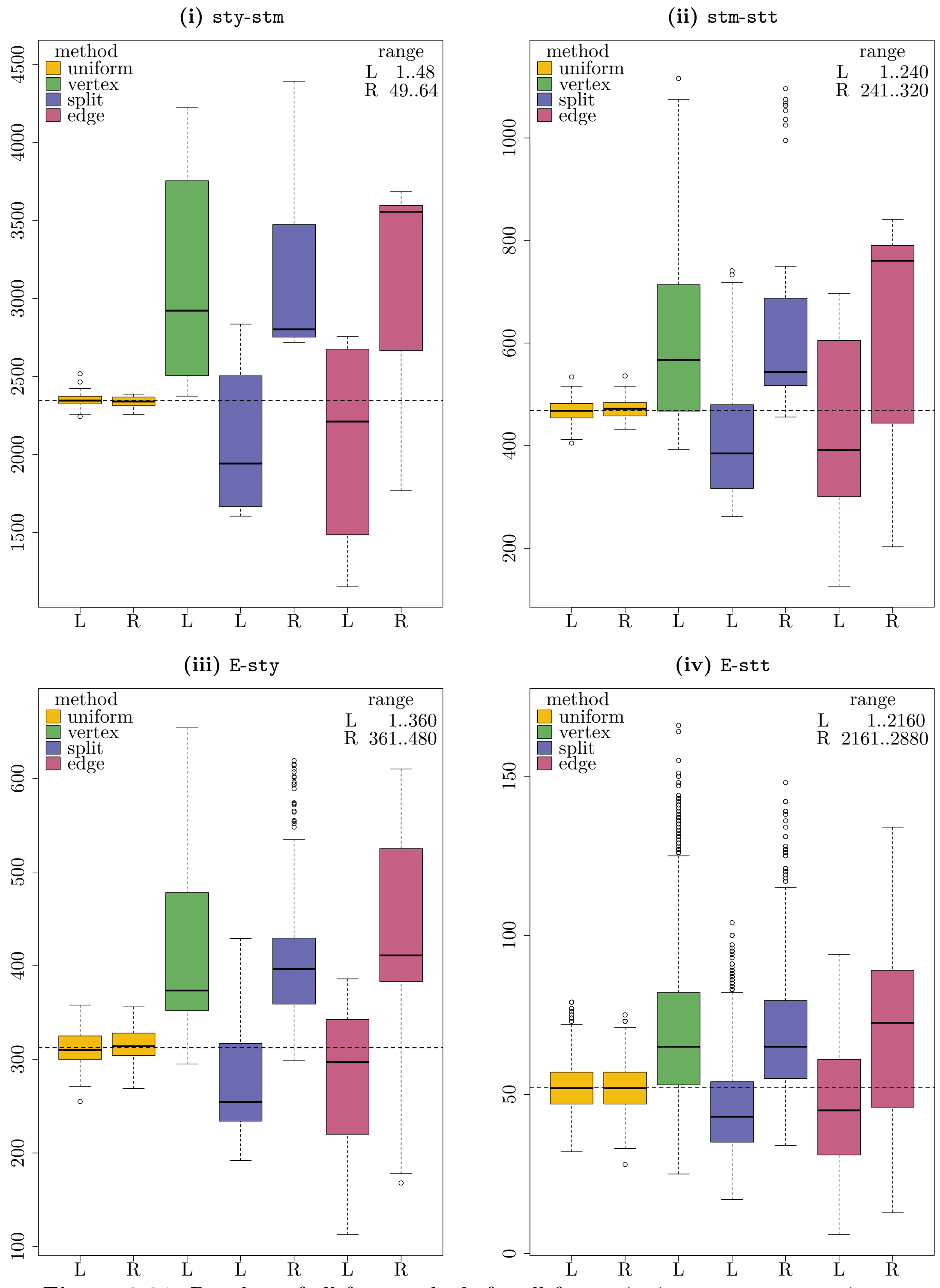


Figure 3.21: Boxplots of all four methods for all four pairwise genome comparisons. Scenarios that can be obtained by all methods are within id-range L, while the scenarios in id-range R cannot be covered by the vertex approach. The horizontal dashed line is the expected frequency for each scenario.

3.3.3 Artificial Data

In order to further show the (dis)similarities in the four methods we created special cases. First, we investigate the behaviour of the four methods when for each cycle there is only one splitgroup (the `tiny` genome). Then we increase the number of splitgroups beyond two (`interm` genome).

Example: `tiny` Genome (only one splitgroup per cycle)

The artificial `tiny` genome compared to the corresponding identity genome (`id`) has DCJ distance 7. The marker order of `tiny` is:

```
>tiny:   a   d  -b  -c  -e   f   i  -g  -h   l   k  j  )
```

The adjacency graph of `tiny` and its identity genome has three 4-cycles (each has distance 1) and two 6-cycles (each has distance 2), yielding a total number of distinct scenarios of 11 340. The vertex-approach randomly extracts a trivial cycle, which means the 4-cycles will be broken down to two trivial cycles, and the 6-cycles will be broken down to one trivial cycle and one 4-cycle. All other methods (edge, split and uniform sampling) yield the same splitgroups, since extracting a 4-cycle from a 6-cycle leaves behind a trivial cycle (which is equivalent to extracting a trivial cycle). Only the probabilities for the results are different. As the resulting cycles of all methods are the same, we expect to achieve all scenarios with any method.

When first sampling the cycle according to its size (vertex and edge approach), bear in mind that a 4-cycle has two vertices to choose from for the extraction. Choosing either of them leads to the same result, which means this scenario is chosen more often than it should be.

For each of the four methods, we sampled 800 000 scenarios. As expected, all four methods were able to produce each of the 11 340 scenarios. Figure 3.22 shows for each of the 11 340 scenarios how often it was sampled with each of the four methods. As can be seen, even for small distance cycles, edge sampling but also split and vertex sampling deviate more than the implemented uniform sampling. This becomes more apparent in Figure 3.23 which shows the boxplot for each method.

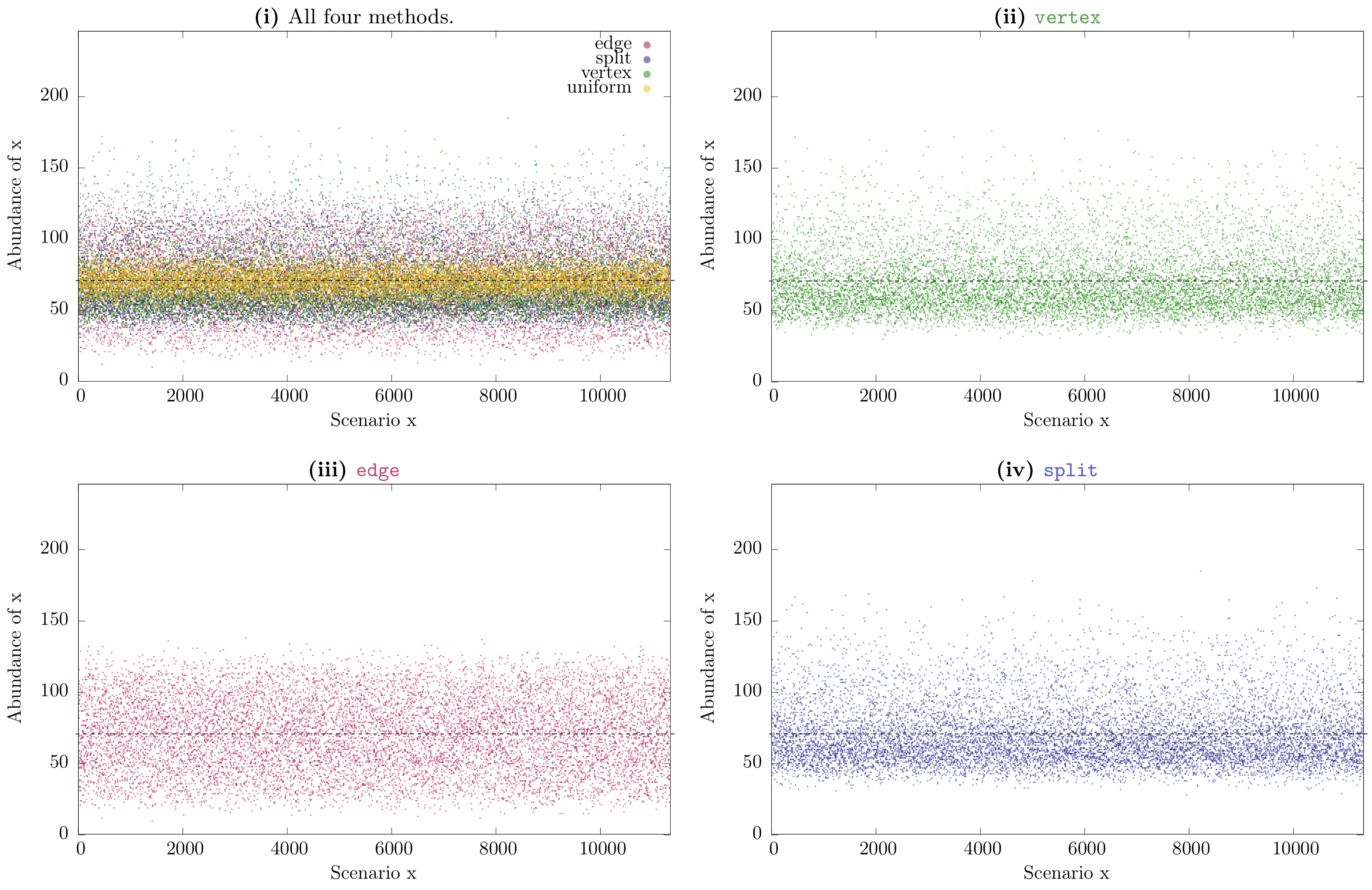


Figure 3.22: Frequency distribution of optimal sorting scenarios of `tiny` example genomes. Comparison of `uniform`, `vertex`, `edge` and `split` sampling where for each comparison and each method 800 000 scenarios were sampled. The horizontal dashed line is the expected frequency for each scenario.

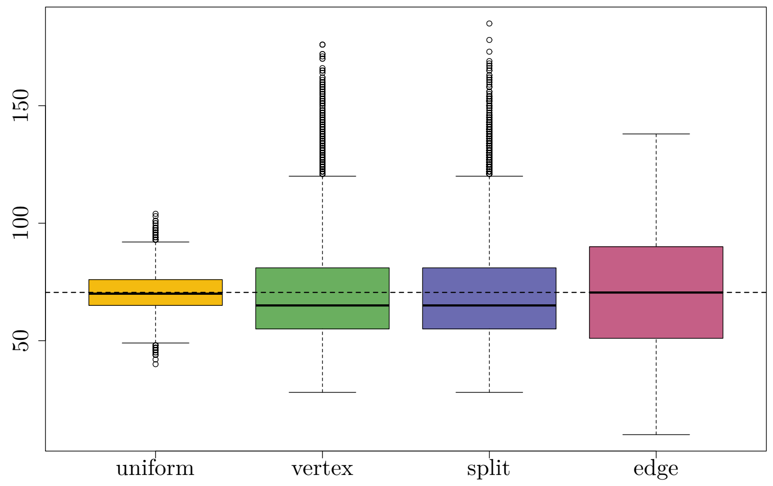


Figure 3.23: Boxplots for the 800 000 samples of the `tiny` genome for the four different methods. The horizontal dashed line is the expected frequency for each scenario.

Example: `interm` Genome (large cycle)

While for the `tiny` example on a large scale, the methods seem to be close to some extent, we now want to show the deviation in the covered sorting scenarios, when the initial adjacency graph has cycles with larger distance and thus more and larger splitgroups. We use the following marker order:

```
>interm:   a  -d   c  -b   f  -g   e  -h   i )
```

and compare it to its identity marker order. Their adjacency graph has two unsorted cycles (distances 1 and 5). The overall DCJ distance is 6 and there are $\mathtt{s} = 7776$ different optimal sorting scenarios. The larger of the cycles has the following splitgroups: $j_1 = (4, 0)$, $j_2 = (3, 1)$ and $j_3 = (2, 2)$ and a total number of 15 elements in them. All scenarios that deal with splitting a cycle into two unsorted cycles (thus the nine elements of splitgroups j_2 and j_3) are missed by vertex sampling.

We sampled 400 000 scenarios for each of the four methods. In average each scenario sampled with uniform sampling should thus occur roughly 50 times. Figure 3.24 shows the results, where in (i) all four methods are given. For a closer look, (ii) shows all except the vertex sampling on a magnified y-axis and (iii) and (iv) show the split and edge sampling, respectively. Clearly, a large part of the sorting scenario space is not produced by the vertex-approach. It can be seen easily that the edge sampling yields lower probabilities. Also the split sampling as a frayed pattern for the splitgroups j_2 and j_3 (this means there are fewer scenarios that occur often, but more that occur less). Figure 3.25 shows the boxplots of all four methods. However, since the vertex

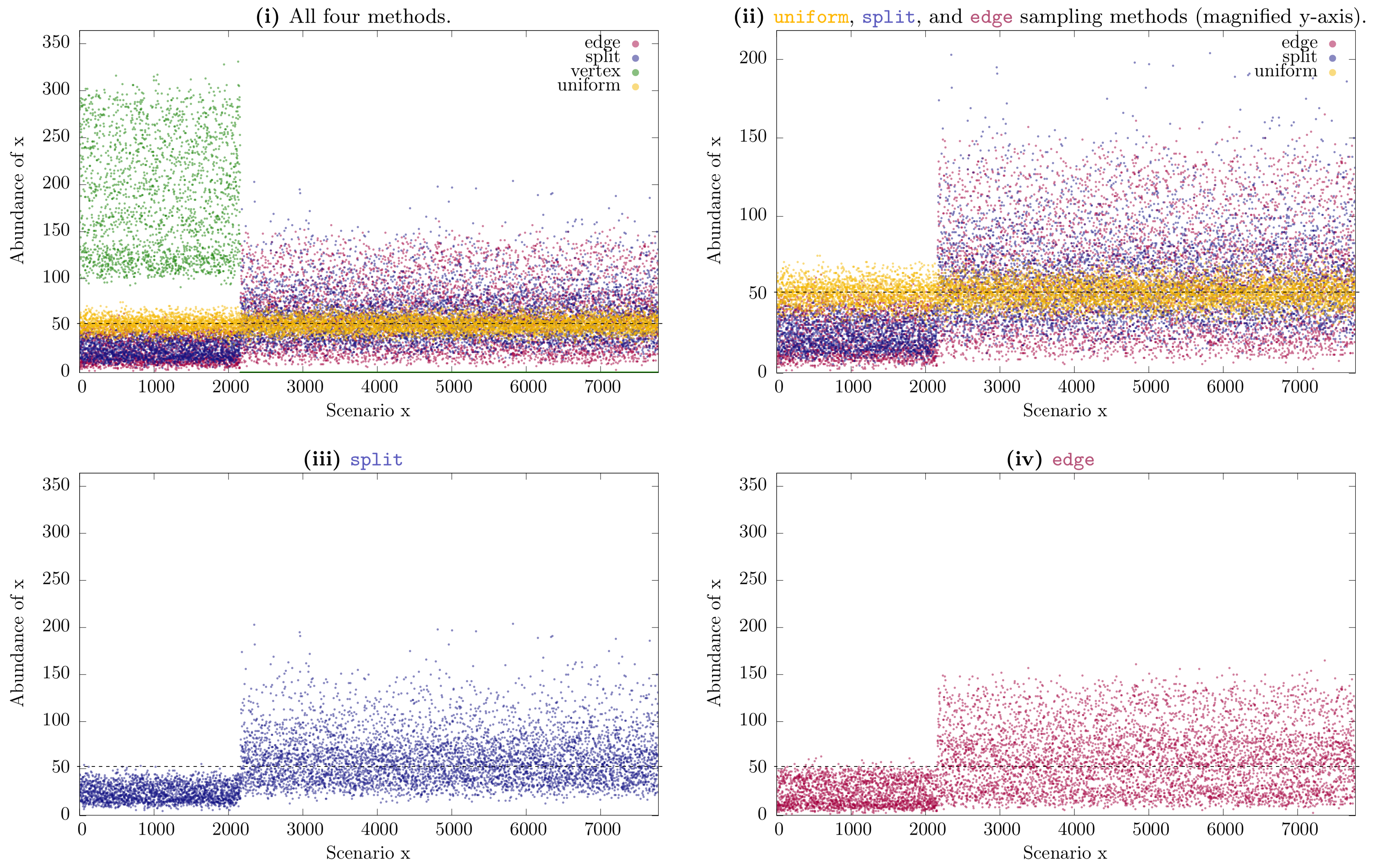


Figure 3.24: Distribution of sorting scenarios of `interm` example genomes. Comparison of `uniform`, `vertex`, `edge` and `split` sampling where for each method 400 000 scenarios were sampled. The horizontal dashed line is the expected frequency for each scenario.

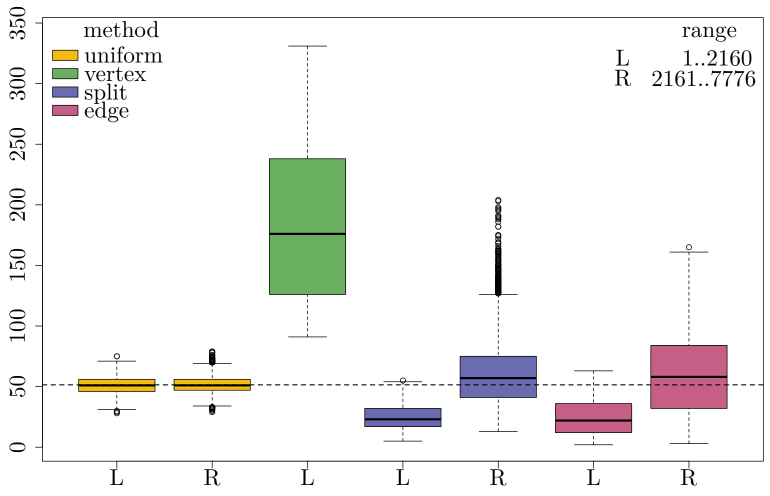


Figure 3.25: Boxplots of all four methods for the `interm` example. The scenarios within id-range L can be obtained by all methods, while the scenarios in the id-range of R cannot be covered by the vertex approach. The scenario ids are the same as in Figure 3.24. The horizontal dashed line is the expected frequency for each scenario.

approach is not able to produce most of the scenarios, the picture shows two sets of boxplots, one set for the scenarios that can be covered by all methods (L), and one set of boxplots for the other scenarios (R).

3.4 Discussion

As already mentioned above, the implementation of the different approaches was done with optimisation towards uniform sampling. The other approaches re-use method calls even if they are only a roundabout way, which made them easy to implement. It renders, however, a runtime comparison pointless. Still, the average user is likely to be only interested in one or a few comparisons for which using the non-uniform methods are unjustified.

We also were inclined to show that the sampling of large integers produces uniformly sampled scenarios. However, let us assume that we are given a pair of genomes that can be sorted in `s` $\gg 10^9$ ways, preferably with several 9-digit-buckets. A frequency plot would then have an x-axis with `s` entries, and, in order to obtain meaningful results such as the graphics done in the previous section, each scenario should occur roughly 50 times (as for the `interm` example that has 7776 different scenarios, or the `tiny` example that has 11 340 different scenarios). As computing and visualising the scenarios and results are very impractical, we abandon the idea.

So far, our implemented uniform sampling method only samples uniformly among all non-recombining sorting scenarios. That means if there are an *AA*- and a *BB*-path at the same time, lots of scenarios are missed. This is elaborated in more detail in the section below.

Adjacency Graph Instances With Recombinations

From Table 2.1 (see page 30) we know that only operations acting on a single component extracting a cycle or operations acting on a *BB*-path and an *AA*-path (or splitting a *BB*-path) are optimal. All operations acting on a cycle and another component or on an *AB*-path and another component are not optimal, and thus cycles and *AB*-paths are always sorted individually.

A *recombination of paths* is a DCJ operation of which one cut is done in an *AA*-path and the other cut is done in a *BB*-path, and after the rejoining we have two *AB*-paths. If no *AA*-path is present, all *BB*-paths must be sorted individually (and vice versa).

Taking into account not only the individual sorting of *AA*- and *BB*-paths but also their possible recombination, which could happen any time until they are resolved to trivial components, expands the sorting space enormously. The Equations (3.1) and (3.2) (page 36 ff.) thus serve only as lower bounds.

The magnitude of the extension of the sorting space is hinted at with the examples that follow. First, we will study the case where we have exactly one *AA*- and one *BB*-path. Let the *AA*-path p_a and the *BB*-path p_b have distances d_a and d_b, respectively. Then one of the cuts can be done in any *A*-vertex of p_a and the other cut can be done in any *A*-vertex of p_b, yielding a total of $(d_a+1)\cdot d_b$ possible cut position combinations, each of them having two distinct rejoins (that are both optimal [23]). As the resulting components are altered and not simply reduced, we may have the case that one *AB*-path has a higher distance than either the *AA*- or *BB*-path (at most $d_a + d_b - 1$). At the same time, instead of recombining the two paths, we can choose one operation acting on p_a or on p_b individually. This means in the first step we can choose:

(i) any of the $(d_a + 1) \cdot \lceil \frac{d_a}{2} \rceil$ operations of p_a or
(ii) any of the $(d_b + 1) \cdot \lceil \frac{d_b}{2} \rceil$ operations of p_b or
(iii) any of the $2 \cdot (d_a + 1) \cdot d_b$ recombinations.

Once the recombination is performed, the resulting *AB*-paths p' and p'' can only be sorted individually and have a combined distance of d_a+d_b-1 (where p' has a distance ranging from zero to d_a+d_b-1). Let us study one simple case in the following example.

Example 9: Let p_a be an *AA*-path with $d_a = 2$ and let p_b be a *BB*-path with $d_b = 1$. In the first step, p_a can have three different DCJ operations, p_b has only one distinct DCJ operation, but there are $(d_a + 1) \cdot d_b = 6$ possible recombinations of p_a and p_b. In total that means ten different possibilities for the first step. $\diamond$

If the number of *AA*- or *BB*-paths is higher, we have even more possibilities, as any *AA*-path could be paired with any *BB*-path or be sorted individually, as observed below.

Example 10: Given an *AA*-path p_a and two *BB*-path p_b and p'_b all with distance 1. In the first step we have three possible DCJ operations acting on a single component each, and we can recombine p_a with p_b or p_a with p'_b. Each recombination leads to four different outcomes, that means in total we have eleven possible first steps. For the next step nine cases have no *AA*-path left, meaning there cannot be further recombinations (each of these have two components with distance 1 each, i.e. two possibilities). The other two cases have p_a and either p_b or p'_b left, which means recombination is still possible and the number of outgoing edges is 6. $\diamond$

The problem of recombining was addressed in [22,23] by taking a specific *AA*-path p_a and a specific *BB*-path p_b. Then these are linked in their telomeres (in one way or the other) and two possible cycles emerge: $c_1(p_a, p_b)$ or $c_2(p_a, p_b)$ that each have distance $d = d_a + d_b$. One such cycle can have $(d + 1)^{d-1}$ scenarios, although not all of them are recombining the paths. The number of sorting sequences recombining a p_a with a p_b is bounded by $2 \cdot (d + 1)^{d-1}$ [23].

Next they computed a solution for the recombination of one pair of *AA*- and *BB*-paths using defined operations. However, in order to compute all possible recombinations or non-recombinations, an enumeration of which *AA*-path is recombined with which *BB*-path and which paths remain sorted individually is required and the solution for each of these sets needs to be computed. An algorithm to compute matchings of *AA*-paths to *BB*-paths was provided and the computation was done for relatively low numbers of paths present. However, a general solution was not provided, as counting the scenarios using the matching is impractical already for these values.

A different approach to sample DCJ sorting scenarios was presented by Miklós and Tannier [75,76]. The authors use an important finding from [23], that showed a way to transform one optimal sequence of DCJ operations into another. The proof in [23] showed that the optimal sequences are all connected, thus that any one such optimal sequence can be transformed into any other optimal sequence by replacements.

Miklós and Tannier defined a distribution θ over the set of all matchings of the complete bipartite graph $K_{n,m}$. They proceeded with showing that drawing from this distribution is equivalent to uniformly sampling DCJ scenarios. Subsequently, a *Markov chain Monte Carlo (MCMC) sampler* with stationary distribution θ was constructed. They then showed that their constructed sampler converges rapidly to its stationary distribution.

Furthermore, Miklós and Tannier studied the complexity of counting *most parsimonious DCJ scenarios* (denoted by #MPDCJ). They were able to show that this problem admits a *fully polynomial time randomized approximation scheme* (FPRAS) and thus that they can draw an optimal DCJ sorting scenario with the MCMC sampler in fully polynomial time. However, to the best of our knowledge, no implementation of this algorithm exists.

Instances without recombinations

There are many adjacency graph instances where recombinations do not play a role. This is the case for example for genomes that have only circular chromosomes and thus do not have any paths in their adjacency graph. Also a pair of co-tailed genomes, i.e. genomes whose telomeres are identical, naturally have only 1-paths and cycles in their adjacency graph and can not have recombinations of *AA*- and *BB*-paths. Furthermore, the presence of linear chromosomes does not necessarily offer recombinations. If there is no *AA*-path we cannot do recombinations, even if there are *AB*- or *BB*-paths. (The same applies to the absence of *BB*-paths). In these cases the sorting space is not restricted and our sampling method provides a scenario sampled uniformly among all possible optimal sorting scenarios.

Everything is theoretically impossible, until it is done.
Robert A. Heinlein

Chapter 4

DCJ-*indel* Model on Circular Genomes Via DCJ Model

The double cut-and-join (DCJ) model described previously is too limited when it comes towards real genomes. Considering two genomes, these rarely consist of orthologous markers only, but rather differ in inserted or lost regions.

In extension to the DCJ model on unlabelled or core genomes, Yancopoulos and Friedberg presented a concept of handling insertions and deletions in the DCJ model [91,92]. In their work (2008), much effort was put into the treatment of insertions and deletions in vertices of the adjacency graph and also in the handling of consecutive insertions or deletions. First, the complementary missing vertices are added to the adjacency graph such that subsequently both genomes have the same markers. Then the distance is derived by the traditional DCJ distance on the *generalised* adjacency graph and a surcharge rule. In the same work, they also presented an approach for DCJ with duplications and a combination of both.

Shortly after, in 2010, my master thesis contained the generalisation of the DCJ model towards indels as well as the distance computation [87]. Distance formulae/algorithms based on the labelled adjacency graph (see also Subsection 2.2.2) were published by Braga, Willing and Stoye [25] in 2010. In 2011 Braga presented detailed sorting algorithms [18] and Braga, Willing and Stoye published the entire DCJ-*indel* distance and sorting problems and solutions in [26].

Compeau [33,34] picked up on this theory and introduced the concept of optimal completion in 2012 that offers an easy way to perform insertions and deletions during sorting. Bader [3] worked on DCJ with duplications and also uses the term *insertion* and *deletion* but refers to whole chromosome loss, insertion or duplication, rather than on marker level.

In this chapter, we will review the DCJ-*indel* model, which includes all operations (and therefore all modifications) allowed under the DCJ model (see Section 2.3) and additionally allows indels (see Section 2.1). More formally: the set of allowed operations is DCJ-*id* $= \mathrm{DCJ} \cup \{\textit{indel}\}$, where DCJ includes all modifications introduced in Section 2.3. We assume unit cost, i.e. an operation $\mu \in$ DCJ-*id* uses up exactly one step in the sorting scenario.

Like the DCJ model also the general DCJ-*indel* model allows all types of components. However, for the purpose of the theory discussed in the ensuing chapters, where only unichromosomal circular genomes are handled, we restrict the elaboration of the DCJ-*indel* model to instances that have only cycles.

Definition 10 (DCJ-*indel* Distance Problem): *Given two genomes A and B with possibly unequal marker content but without duplications, find the minimum number of steps required to sort A into B using only* DCJ *operations and indel operations, called the* DCJ-*indel* distance *denoted by $d^{id}_{\mathrm{DCJ}}(A, B)$.*

The basic concepts and definitions were stated in Chapter 2 in which we also observed a first intuitive upper bound for the distance with indels stated in Inequality (2.2). We make extensive use of the data structures introduced in Section 2.2 (particularly Subsection 2.2.3). As mentioned earlier, we assume parsimony and prefer integrating or deleting several markers at once rather than inserting or deleting them separately. Sections 4.2 and 4.3 concentrate on sorting the components of the master graph and simultaneously grouping indels, and in Subsection 4.5.2 we review the theory of Compeau concerning optimal completion. Most of this chapter is based upon [26] which we subsume in the context and with adapted data structures necessary for the theory following in the ensuing chapters.

4.1 Generalising the DCJ Model and Distance

The distance of different genome modification models can be given with respect to the DCJ distance. The distance formula for a model $\mathrm{M} \in \mathcal{M}$ with $\mathrm{M} = \mathrm{R} \cup \mathrm{I}$ is then:

$$d_{\mathrm{M}}(A, B) = d^{\mathrm{I}}_{\mathrm{R}}(A, B) = d_{\mathrm{DCJ}}(A, B) + \tau^{\mathrm{I}}_{\mathrm{R}}(A, B). \tag{4.1}$$

Obviously, $\tau^{\mathrm{I}}_{\mathrm{R}} = \tau_{\mathrm{DCJ}} = 0$ for the DCJ model.

For some other genome modification models the value of τ has already been determined. For example, we will revise and determine the computation for the DCJ-*indel* model in this chapter and for the inversion and inversion-*indel* model in the following chapters (that is the computation of τ^{id}_{DCJ} and τ^{id}_{INV}). Further models and values of τ are given in Chapter 8.

4.1.1 The DCJ Operation on Labelled $\mathcal{G}$-Adjacencies

For unlabelled $\mathcal{G}$-adjacencies the DCJ operation has been defined in Section 2.3. A single cut acting on an unlabelled adjacency splits the two extremities that are common extremities or caps. Generalising towards genomes with unique markers, we use (labelled) $\mathcal{G}$-adjacencies. The label of a $\mathcal{G}$-adjacency can be formed by an arbitrary number of unique markers (the notation in Section 2.1 allows us to use strings of markers or strings of extremities). For all $\mathcal{G}$-adjacencies that are not circular singletons, splitting within a label possibly increases the number of labelled adjacencies/indel operations, hence we consider only the cut positions next to the common extremities.

A special case is a circular singleton that only exists in one of the genomes. This adjacency has a circular label, i.e. it has no common extremities but at the same time is not telomeric. Such a circular singleton can be split in any position between two adjacent extremities. More formally we define:

Definition 11 (DCJ): *The double cut-and-join operation in the presence of unique markers acts on two $\mathcal{G}$-adjacencies $p\,\ell_1^t\ell_1^h\,q$ and $r\ell_2^t\ell_2^h\,s$ of the same genome. The core adjacencies $u_1 = pq$ and $u_2 = rs$ are replaced by either $v_1 = pr$ and $v_2 = qs$ or by $w_1 = ps$ and $w_2 = rq$. The possible cut positions never have a unique extremity on both sides (except for circular singletons). And the joins never produce two labelled adjacencies. Let the cut positions be a/b for u_1 and x/y for u_2 (as depicted in Figure 4.1). The unique extremities after the operation can be distributed in several ways taking into account their order:*

(v)	(w)
a-x: $\ell(v_1) = \varepsilon$ and $\ell(v_2) = \ell_1^h\ell_1^t\ell_2^t\ell_2^h$	b-x: $\ell(w_1) = \ell_1^t\ell_1^h\ell_2^t\ell_2^h$ and $\ell(w_2) = \varepsilon$
b-y: $\ell(v_1) = \ell_1^t\ell_1^h\ell_2^h\ell_2^t$ and $\ell(v_2) = \varepsilon$	a-y: $\ell(w_1) = \varepsilon$ and $\ell(w_2) = \ell_2^t\ell_2^h\ell_1^t\ell_1^h$

The case of circular singletons is derived by removing in the above description/in Figure 4.1 both common extremities of the same adjacency. An adjacency that is unlabelled from the beginning can be handled by using the empty label ε.

4.2 DCJ and *indel* Operations on a Labelled Cycle

In this section we study the effect of DCJ and indel operations on cycles of the master graph and their impact on the DCJ-*indel* distance.

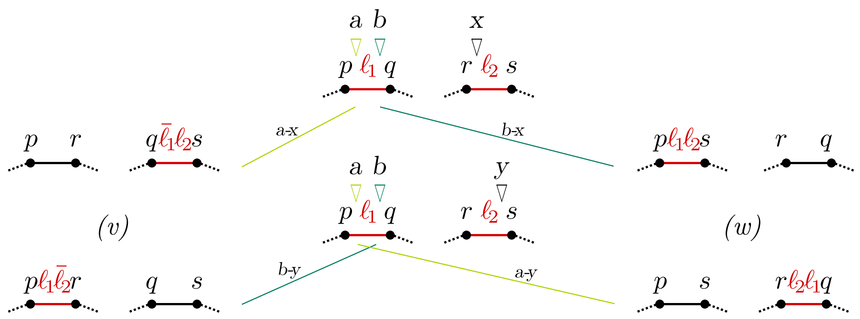


Figure 4.1: Possible double cut-and-join combinations for labelled adjacencies in the master graph. The extremities in labels are condensed to markers. In case of a circular singleton p and q or r and s are removed.

Given a cycle with unique markers in only one of the two genomes, by extracting unlabelled cycles (an optimal DCJ operation) we can *accumulate* all labels in one $\mathcal{G}$-adjacency. For an example see Figure 4.2. How this can be achieved if the labels occur in genome B is shown later in Figure 4.7, for now it is only relevant that it is possible.

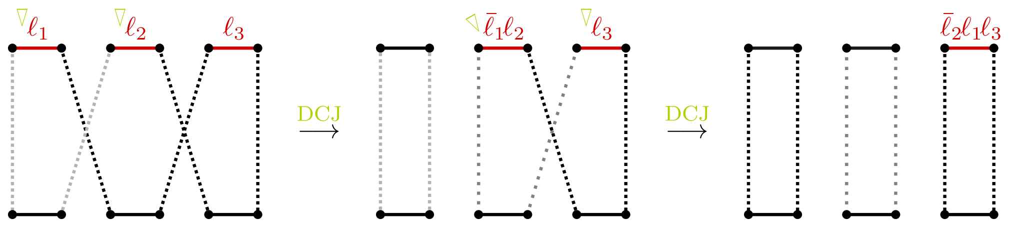


Figure 4.2: Accumulating all labels of an $\mathcal{A}$-cycle into one label by optimal DCJ operations.

When labels are present in both A-edges and B-edges, sometimes more than one labelled adjacency edge remains after using optimal DCJ operations for accumulation. For this we use the concept of *runs* from [26]. Let s be the string obtained by walking along the edges of a cycle and concatenating all unique extremities (the labels) in the direction of walking. A maximal substring consisting of labels from $\mathcal{A}$, resp. $\mathcal{B}$, is called an $\mathcal{A}$-run, resp. $\mathcal{B}$-run. An illustration of this can be seen in Figure 4.3.

Let $\Lambda(C)$ denote the number of runs of a cycle C. By construction, a cycle can have zero, one or an even number of runs. Each labelled cycle in Figure 4.2 has one run, the cycle in Figure 4.3 has two runs. Each run can be accumulated into a single label using only optimal DCJ operations, such that in the end the cycle has one label for each run. This way, we need at most one indel operation per run.

When more than two runs are present, runs can be *merged*. In the DCJ-*indel* model there are two ways to do this using only optimal DCJ operations and indel-operations

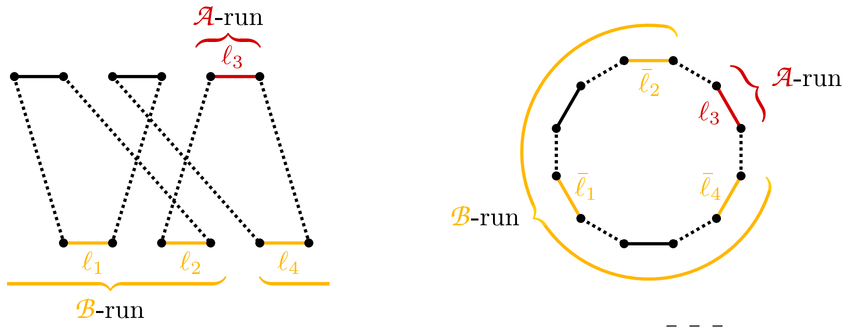


Figure 4.3: A cycle C has one $\mathcal{A}$-run (ℓ_3) and one $\mathcal{B}$-run ($\bar{\ell}_4\bar{\ell}_1\bar{\ell}_2$). Left: C in a relational diagram. Right: C in the master graph without aligning the vertices w.r.t. their genomes.

as the following two examples show.

Example 11 (Merging runs by DCJs): Figure 4.4 shows a cycle with six runs in total. By extracting ℓ_6 into a trivial cycle (one DCJ operation), the two $\mathcal{A}$-runs ℓ_5

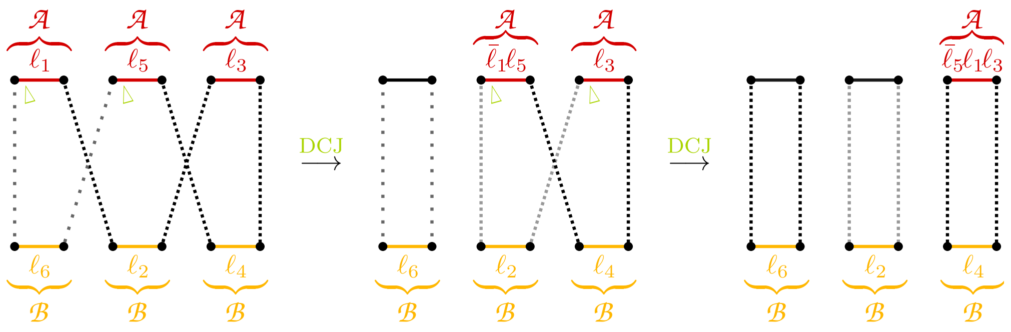


Figure 4.4: A cycle with initially six runs can be transformed into three cycles with a total of four runs by optimal DCJ operations that merge runs.

and ℓ_1 are merged into a single label. Thus, the total number of runs is reduced from six to five plus the DCJ distance is reduced by 1, all while using exactly one optimal DCJ operation. Another such operation splits the larger cycle into two trivial cycles, one with the $\mathcal{B}$-run ℓ_2 and one cycle with the $\mathcal{A}$-run $\bar{\ell}_5\ell_1\ell_3$ and the $\mathcal{B}$-run ℓ_4. Again the change in Λ is -1 and merging of runs can no longer occur. In the full sorting scenario, we use two optimal DCJ operations and four indel operations (one deletion and three insertions). ◇

The same result can be achieved when, instead of fusing two $\mathcal{A}$-runs through extracting a single $\mathcal{B}$-run, we extract also the $\mathcal{A}$-runs to the left and right of the $\mathcal{B}$-run, thus effectively fusing not only the two $\mathcal{A}$-runs in the new cycle, but also the neighbouring $\mathcal{B}$-runs in the original cycle. We can also choose a different order of insertions, deletions and DCJ operations.

Example 12 (Merging runs by *indels*): Figure 4.5 shows the same cycle as in the previous example. Now we will use indels instead of DCJ operations to decrease the number of runs. By deleting ℓ_3 (one indel operation) the two $\mathcal{B}$-runs ℓ_2 and ℓ_4 are

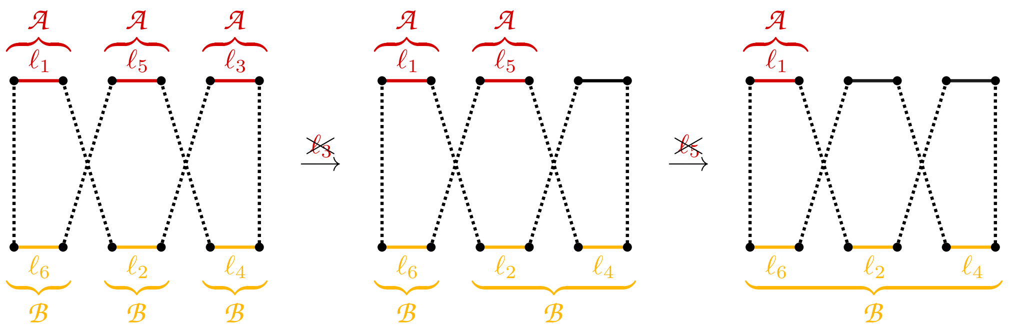


Figure 4.5: A cycle with initially six runs can be transformed into a cycle with only two runs by merging runs.

merged into a single run. Thus, the total number of runs is reduced from six to four while using a single indel operation. Another indel operation (deleting ℓ_5) yields a cycle with only one $\mathcal{A}$- and one $\mathcal{B}$-run. Again the change in Λ is -2 and merging of runs can no longer occur. In a complete sorting scenario, we use four indels (three deletions and only one insertion for the $\mathcal{B}$-run) and, in order to split the cycle into trivial cycles, we use two DCJ operations. $\diamond$

We see that treating one run separating two others takes one operation that simultaneously merges the separated runs. We can repeatedly do this until no more runs can be merged.

The *indel-potential* $\lambda(C)$ of a labelled cycle C gives the potential reduction, i.e. to how many indel operations the runs can be reduced by merging runs [26]:

$$\lambda(C) = \left\lceil \frac{\Lambda(C)+1}{2} \right\rceil . \tag{4.2}$$

Obviously, unlabelled cycles have no runs and thus the indel-potential of an unlabelled cycle C is $\lambda(C) = 0$. The indel-potential is achieved when merging runs by optimal DCJs or merging runs by indels and optimally sorting the resulting cycle.

4.3 DCJ Operations on a Pair of Labelled Cycles

A DCJ operation applied to two cycles creates a single cycle (see Table 2.1 on page 30). While cutting and joining the adjacencies of the two cycles, we watch out for $\mathcal{A}$- or $\mathcal{B}$-run(s) and we measure the effect on the indel-potential using the following definition.

Definition 12 ($\Delta\lambda$): *Let ρ be a DCJ operation acting on cycles of the master graph* $\mathrm{MG}(A,B)$*, resulting in* $\mathrm{MG}(A',B)$*. Then*

$$\Delta\lambda(A,B,\rho) = \sum_{C' \in \mathrm{MG}(A',B)} \lambda(C') - \sum_{C \in \mathrm{MG}(A,B)} \lambda(C) \tag{4.3}$$

denotes the difference in the indel-potential evoked by ρ (we use $\Delta\lambda$ for short).

Now let us have a closer look at different cycle-merging operations. Bear in mind the allowed cut positions determined in Subsection 4.1.1. After merging, the new cycle contains all labels from the original cycles, and thus also indel-types. Table 4.1 shows the change induced by DCJ operations acting on different combinations of cycles.

If both cycles have a single or no run, the cuts in each of the cycles can be done in any $\mathcal{G}$-adjacency from A. If there is more than one run in one of the initial cycles, the cut has to be in an A-vertex between an $\mathcal{A}$- and a $\mathcal{B}$-run in order to achieve the best outcome, that is fusing as many runs as possible with a single DCJ operation. If the labellings of the cycles is such that no runs can be fused, the new cycle inherits the indel-potential of the two initial cycles (first two rows of Table 4.1). If one pair of runs can be fused, the indel-potential changes by $\Delta\lambda = -1$ (rows 3-6). Finally, in the

Table 4.1: Merging of cycles with different labellings and the impact on their DCJ-*indel* distance. $\mathcal{X}$ is a cycle with arbitrary labelling.

Δd_{DCJ}	indel operands		$\to$	indel resultant	fused runs	$\Delta\lambda$	Δd_{DCJ} $+\Delta\lambda$	
+2	ε	$\mathcal{X}$		$\mathcal{X}$	/	0	+2	counter-
+2	$\mathcal{A}$	$\mathcal{B}$		$\mathcal{AB}$		0	+2	optimal
+2	$\mathcal{A}$	$\mathcal{A}$		$\mathcal{A}$	$\mathcal{A}$	−1	+1	neutral
+2	$\mathcal{B}$	$\mathcal{B}$		$\mathcal{B}$	$\mathcal{B}$	−1	+1	
+2	$\mathcal{A}$	$\mathcal{AB}$		$\mathcal{AB}$	$\mathcal{A}$	−1	+1	
+2	$\mathcal{B}$	$\mathcal{AB}$		$\mathcal{AB}$	$\mathcal{B}$	−1	+1	
+2	$\mathcal{AB}$	$\mathcal{AB}$		$\mathcal{AB}$	$\mathcal{A}$, $\mathcal{B}$	−2	0	optimal

last row of Table 4.1, we show the effect of merging two cycles that both have at least two runs. Here, two $\mathcal{A}$- and two $\mathcal{B}$-runs are fused.

4.4 Distance

Adapting Equation (4.1) to the DCJ-*indel* distance gives:

$$d^{id}_{\text{DCJ}}(A,B) = d_{\text{DCJ}}(A,B) + \tau^{id}_{\text{DCJ}}(A,B), \tag{4.4}$$

where τ^{id}_{DCJ} denotes the extra cost with respect to applying the DCJ model, i.e. the change in the number of steps in the sorting scenario when indels are used.

Any indel operation that meets the indel-potential is optimal. Inserting or deleting in a wrong position or only parts of runs would not meet the indel-potential of that cycle. Note that Definition 12 gives the change in the indel-potential, not the change in the length of the sorting scenario. It has been shown that for circular genomes, the value of τ^{id}_{DCJ} is given by the sum of all indel-potentials:

Theorem 2 ([26]): *Given two circular genomes A and B, then the minimum number of DCJ and indel operations necessary to sort A and B is given by:*

$$d^{id}_{\text{DCJ}}(A,B) = \quad |\mathcal{G}| - c \quad + \sum_{C \in \text{MG}(A,B)} \lambda(C), \tag{4.5}$$

where $\mathcal{G}$ is the set of common markers, c is the number of cycles in the master graph $\text{MG}(A,B)$ *and $\lambda(C)$ is the indel-potential of cycle C.*

Using unit costs, a generalisation of the definition of Δd_{DCJ} (see Equation (2.4) on page 29), towards all operations from M, replaces the "+1" by "$+\,\mathbb{1}_{\mu\in\text{R}}$", which means the cost of μ is only accounted for if it is an operation from R. Similarly we define:

Definition 13 ($\boldsymbol{\Delta\tau^{id}_{\text{DCJ}}}$): *Given two genomes A and B, let μ be an operation of a type from* M *acting on genome A yielding A', then*

$$\Delta\tau^{id}_{\text{DCJ}}(A,B,\mu) = \tau^{id}_{\text{DCJ}}(A',B) - \tau^{id}_{\text{DCJ}}(A,B) + \mathbb{1}_{\mu\in\text{I}} \tag{4.6}$$

gives the change in extra cost induced by μ. If $\mu \in$ R the cost are accounted for in Δd_{DCJ}.

We can now deduce:

Observation 5. Let A and B be two genomes with possibly unequal marker content but without duplications. Given an operation $\mu \in \text{M} = \text{DCJ} \cup \{\textit{indel}\}$ the change in the DCJ-*indel* distance induced by μ is as follows:

$$\begin{aligned}
&\Delta d^{id}_{\mathrm{DCJ}}(A, B, \mu)\\
&= d^{id}_{\mathrm{DCJ}}(A', B) - d^{id}_{\mathrm{DCJ}}(A, B) + 1\\
&= \big(d_{\mathrm{DCJ}}(A', B) + \tau^{id}_{\mathrm{DCJ}}(A', B)\big) - \big(d_{\mathrm{DCJ}}(A, B) + \tau^{id}_{\mathrm{DCJ}}(A, B)\big) + 1\\
&= d_{\mathrm{DCJ}}(A', B) - d_{\mathrm{DCJ}}(A, B) + \mathbb{1}_{\mu \in \mathrm{R}} + \tau^{id}_{\mathrm{DCJ}}(A', B) - \tau^{id}_{\mathrm{DCJ}}(A, B) + \mathbb{1}_{\mu \in \mathrm{I}}\\
&= \Delta d_{\mathrm{DCJ}}(A, B, \mu) + \Delta \tau^{id}_{\mathrm{DCJ}}(A, B, \mu).
\end{aligned}$$

An operation that induces a change in the master graph yielding $\Delta d^{id}_{\mathrm{DCJ}} = 0$ is optimal when assuming unit costs.

4.5 On Sorting With Indels

During the sorting process we usually assume a direction from genome A to genome B. In order to derive genome B, operations are applied to genome A only, which means we cannot change the layout of genome B (neither the arrangement of markers nor the genomic content). While sorting, unique markers from $\mathcal{A}$ are deleted which is simply done by deleting labels. Unique markers from $\mathcal{B}$ need to be introduced to the intermediate genomes and the arrangement of all markers needs to be altered to finally match that of genome B. Although in an arbitrary scenario the order of these operations may vary, from [36] we know that insertions can always be moved ahead of the DCJ operations, s.t. they occur in the first steps, and analogously the deletions can be moved aback to occur after the DCJ operations in the last steps. Figure 4.6 visualises how the sorting from genome A via intermediate genomes towards genome B can be devised.

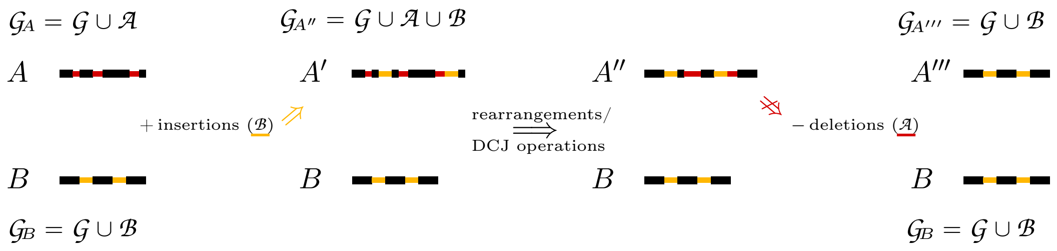

$\mathcal{G}_A = \mathcal{G} \cup \mathcal{A}$
$\mathcal{G}_{A''} = \mathcal{G} \cup \mathcal{A} \cup \mathcal{B}$
$\mathcal{G}_{A'''} = \mathcal{G} \cup \mathcal{B}$
A
A'
A''
A'''
+ insertions ($\mathcal{B}$)
rearrangements/ DCJ operations
− deletions ($\mathcal{A}$)
B
B
B
B
$\mathcal{G}_B = \mathcal{G} \cup \mathcal{B}$
$\mathcal{G}_B = \mathcal{G} \cup \mathcal{B}$

Figure 4.6: A schematic sorting of genome A into genome B. In the end we have $A''' = B$. When insertions or deletions are not present or not allowed by the model, the respective steps are omitted.

4.5.1 Introducing Insertions from a $\mathcal{B}$-run

We now show how the indel-potential can be met for inserting a $\mathcal{B}$-run to genome A when the labels are spread over several labelled $\mathcal{G}$-adjacencies and how the insertion of a $\mathcal{B}$-run is effectuated.

Due to symmetry, since $\mathcal{A}$-labels can be accumulated, so should $\mathcal{B}$-labels. If the direction of sorting were reversed, we would apply DCJ operations and accumulate the labels of each run and finally we would delete the run. If genomes A and B would be swapped and the operations performed we can reversely-engineer a sorting with insertions. This means the sequence of operations is reversed, thus we do the "swapped deletions" (which are insertions) and then the "swapped DCJ" operations. Figure 4.7 shows a cycle with three labels that form one $\mathcal{B}$-run. The reversely-engineered accumulation yields the label $\bar{\ell}_1\ell_2\bar{\ell}_3$ that is introduced to the central A-edge. How to obtain such a label is shown in the ensuing section.

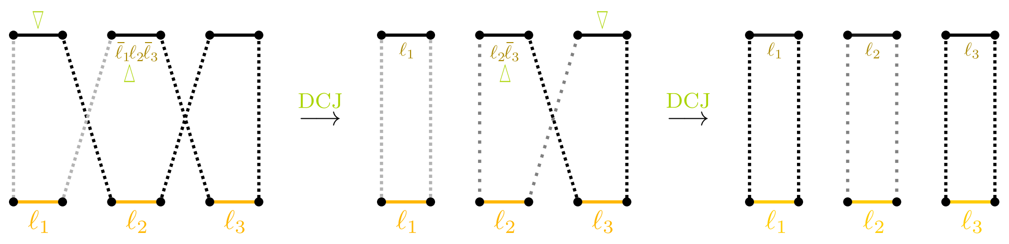


Figure 4.7: Labels of a $\mathcal{B}$-run are introduced as an accumulated label. Then, DCJ operations split the label, until ℓ_1, ℓ_2 and ℓ_3 are a single label each.

Instead of showing the graph extended by the, now common markers, ℓ_1, ℓ_2 and ℓ_3, we keep the labels in mind for easier understanding. During the DCJ part of the sorting process the accumulated run is split such that in the end the inserted markers are in the correct positions and in the correct direction (w.r.t. genome B).

4.5.2 Optimal Completion

This separation of insertions, DCJs and deletions within the sorting scenario also appears in [33], where an alternative approach was presented to compute the DCJ-*indel* distance, based on the concept of *optimal completion*. The paragraphs that follow provide an excerpt of the theory from [89], which uses the approach from Compeau also for the inversion-*indel* model (as we will present in the ensuing chapter). The content is adapted to the formalism and data structures used in this thesis (distance, master graph, the concept of runs and the indel-potential).

In the approach presented by Compeau, each indel is modelled as self-contained circular chromosome. More precisely, it is a circular singleton, composed only of the markers that are to be inserted or deleted by this indel. A *completion* of genomes A and B adds i new such circular singletons to A (these represent insertions into A) and k new such circular singletons to B (these represent deletions from A), yielding two multichromosomal circular genomes that have the same content $\mathcal{G} \cup \mathcal{A} \cup \mathcal{B}$. A completion is *optimal* when the overall indel-potential of the two genomes is achieved, thus when

$$i + k = \sum_{C \in \mathrm{MG}(A,B)} \lambda(C), \tag{4.7}$$

where $\lambda(C)$ is the indel-potential of cycle C.

Construction of an indel. Let r be a $\mathcal{B}$-run of a cycle C in $\mathrm{MG}(A,B)$, composed of m labels. Then let s_r be the circular singleton obtained from $\mathrm{MG}(A,B)$ by walking through the path that corresponds to the run r and concatenating in this very order the extremities of its m labels. We close the circular chromosome concatenating also the last to the first extremity. Such a singleton s_r is called *run-singleton* (it contains exactly one complete run, in this case run r).

Introduction of an indel to $\mathbf{MG}(\boldsymbol{A},\boldsymbol{B})$. The *complementation* of genome A towards A' by the run-singleton s_r introduces s_r to the master graph. The run-singleton remains by itself (not integrated to the chromosome). This way, m more common markers are produced ($|\mathcal{G}'| = |\mathcal{G}|+m$), introducing m new adjacencies in each genome (that means m new A-edges as well as m new B-edges, and $2m$ more extremity edges). At the same time, $m-1$ new unlabelled cycles are created. Furthermore, the cycle C in $\mathrm{MG}(A,B)$ whose run r we introduced is transformed into a cycle C' in $\mathrm{MG}(A',B)$ and contains the same labels as C except for the m labels of r. We have

$$c' = c + m - 1, \quad \text{i.e.} \quad \Delta c = m - 1,$$

where c, resp. c', are the number of cycles before and after the introduction. Thus we have

$$d_{\mathrm{DCJ}}(A',B) = (|\mathcal{G}| + m) - (c + m - 1) = |\mathcal{G}| - c + 1 = d_{\mathrm{DCJ}}(A,B) + 1.$$

Proposition 4 ([89]): *If we add the run-singleton s_r of a $\mathcal{B}$-run r to genome A*

resulting in genome A', the overall indel-potential is achieved, that is,

$$\sum_{C' \in \mathrm{MG}(A',B)} \lambda(C') = \left(\sum_{C \in \mathrm{MG}(A,B)} \lambda(C) \right) - 1. \tag{4.8}$$

(Analogous for the addition of the run-singleton $s_{r'}$ of an $\mathcal{A}$-run r' to genome B.)

Proof. Let C_r be the cycle in $\mathrm{MG}(A,B)$ that contains r. We then add the run-singleton s_r to genome A resulting in genome A'. If C_r originally had one or two runs, clearly the sum of the indel-potentials in $\mathrm{MG}(A',B)$ decreases by 1 with respect to $\mathrm{MG}(A,B)$. If C_r originally had more runs, r was situated between two $\mathcal{A}$-runs. These are now no longer separated by r and become one single run in $\mathrm{MG}(A',B)$, and this also guarantees that the sum of the indel-potentials decreases by 1. □

This behaviour of merging runs can also be observed in Example 12 on page 76.

Integration of a run-singleton. In order to complete the operation, we still need to integrate the singletons that we introduced so that we obtain a unichromosomal genome. Again, let r be a $\mathcal{B}$-run and let A' be the genome composed of A and the run-singleton. We know that $d_{\mathrm{DCJ}}(A',B) = d_{\mathrm{DCJ}}(A,B) + 1$ and in order to integrate the singleton we need to perform exactly one DCJ operation on two A-edges of $\mathrm{MG}(A',B)$, such that one is part of the chromosome of A and the other is part of the run-singleton [64,90]. An *optimal integration* is an integration that preserves the runs of the master graph.

Proposition 5 ([89]): *Any integration of the run-singleton of a $\mathcal{B}$-run r with m labels into the chromosome of A which creates a new unlabelled cycle in the master graph is optimal. (Analogous for the integration of an $\mathcal{A}$-run into the chromosome B.)*

Proof. A circular singleton that is added and integrated to the master graph increases the number of cycles by m. For the integration (a DCJ operation) one A-edge of one cycle C is chosen and one A-edge of the newly added edges of the singleton. The integration splits C into two cycles C_1 and C_2. If one of these two cycles is unlabelled, then all runs of C must be remaining in the other cycle, therefore, the runs of the graph are preserved. □

Using the results in this sections for the initial genome A and the resulting genome A', let λ be short for the sum of indel-potentials in the initial master graph. Adding the singleton to the master graph adds $m-1$ new cycles, and integrating it optimally splits an existing cycle into two cycles. We thus have the initial distance $d^{id}_{\text{DCJ}} = |\mathcal{G}| - c + \lambda$ and after the integration we have:

$$d^{id}_{\text{DCJ}}{}' = (|\mathcal{G}| + m) - (c + m) + (\lambda - 1) = |\mathcal{G}| - c + \lambda - 1 = d^{id}_{\text{DCJ}} - 1.$$

We observe that performing one integration complies with Observation 5:

$$\begin{aligned}
\Delta d^{id}_{\text{DCJ}} &= d^{id}_{\text{DCJ}}{}' - d^{id}_{\text{DCJ}} + \mathit{cost}(\mu) \\
&= \big(d_{\text{DCJ}} + \tau^{id}_{\text{DCJ}} - 1\big) - (d_{\text{DCJ}} + \tau^{id}_{\text{DCJ}}) + \mathit{cost}(\mu) \\
&= \big(d_{\text{DCJ}} - d_{\text{DCJ}} + \mathit{cost}(\mu \in \text{R})\big) + \big(\tau^{id}_{\text{DCJ}} - 1 - \tau^{id}_{\text{DCJ}} + \mathit{cost}(\mu \in \text{I})\big) \\
&= \Delta d_{\text{DCJ}} + \Delta \tau^{id}_{\text{DCJ}} \ .
\end{aligned}$$

We have two possibilities to account for μ. First, we can choose that the addition of the r-singleton to the master graph has no cost and μ corresponds to the necessary DCJ operation to integrate the r-singleton into the corresponding cycle. Second, we could choose to penalise the insertion of s_r to the corresponding adjacency and allow the expansion of the master graph free of charge. In either way in the sorting scenario we have thus the initial genome and in the next step the intermediate genome that has s_r situated in the correct position. Hence, we treat the complete process from the initial genome to the genome with the run integrated as one operation and therefore μ has cost 1 using up exactly one step in the sorting process and we have $\Delta d^{id}_{\text{DCJ}} = 0$ if we achieved an optimally integrated completion.

With the previous results we have a straight recipe for the construction of an *optimally integrated completion* of genomes A and B. At each step we can decide arbitrarily whether we optimally integrate the run-singleton s_r of a $\mathcal{B}$-run to A, or the run-singleton $s_{r'}$ of an $\mathcal{A}$-run to B, until no more runs exist in the master graph. In the end we have two unichromosomal circular genomes A^* and B^* with the same content.

In Example 13 we build one optimally integrated completion in three steps (see the next page).

Example 13 (Optimal integration): Given two unichromosomal circular genomes $A = \{(a, x, -c, y, b, -z, -d)\}$ and $B = \{(a, u, b, c, v, d)\}$ and their master graph has one cycle C with two $\mathcal{A}$-runs ($x^t x^h$ and $z^t z^h y^t y^h$) and two $\mathcal{B}$-runs ($v^t v^h$ and $u^t u^h$), as depicted in Figure 4.8.

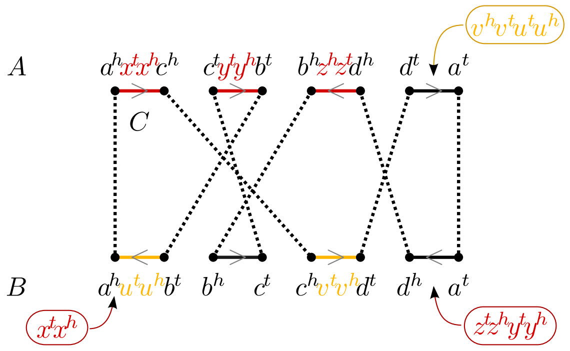
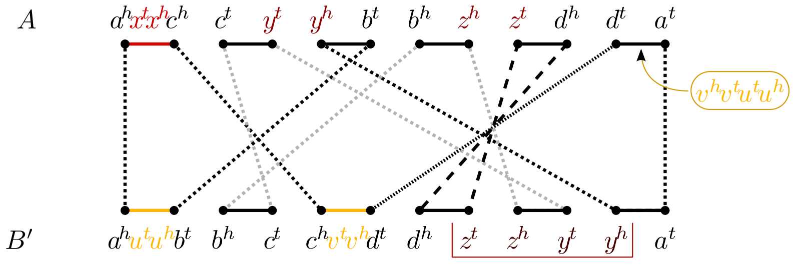


Figure 4.8: For genomes $A = \{(a, x, -c, y, b, -z, -d)\}$ and $B = \{(a, u, b, c, v, d)\}$ we show three singletons and the positions for an integration of these to $\mathrm{MG}(A, B)$ that lead to an optimally integrated completion.

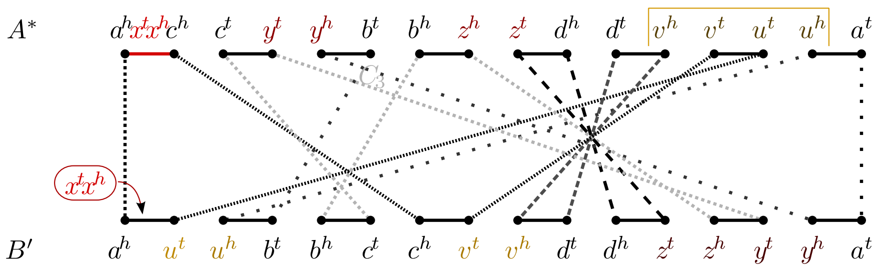
Since $\lambda(C) = 3$ we need to perform three optimal integrations. We choose the three singletons (zy), $(-vu)$ and (x) in this order described in the following.

(zy) : We integrate the singleton $(z^t z^h y^t y^h)$, composed of the labels of an $\mathcal{A}$-run, into the chromosome of genome B, creating $B' = \{(a, u, b, c, v, d, z, y)\}$.



$\mathrm{MG}(A, B')$ now has three cycles of which two are unlabelled and the third contains two runs (u and v, formerly two runs, now form a single run, $-vu$).

$(-vu)$: The singleton $(v^h v^t u^t u^h)$, composed of the labels of the $\mathcal{B}$-run, is integrated into the chromosome of A, creating $A^* = \{(a, x, -c, y, b, -z, -d, -v, u)\}$.



MG(A^*, B') has five cycles, of which four are unlabelled and one has an $\mathcal{A}$-run.

(x) : Finally, we integrate $(x^t x^h)$, composed of the labels of the last $\mathcal{A}$-run, into the chromosome of genome B', creating $B^* = \{(a, x, u, b, c, v, d, z, y)\}$, resulting in MG$(A^*, B^*)$ composed of six unlabelled cycles, see Figure 4.9.

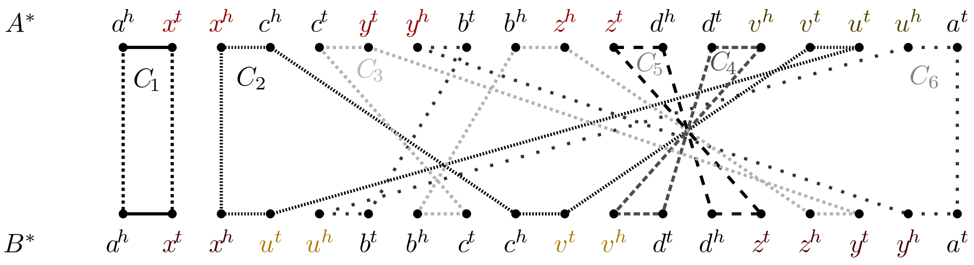

A^*
d^h x^t x^h c^h c^t y^t y^h b^t b^h z^h z^t d^h d^t v^h v^t u^t u^h a^t
C_1
C_2
C_3
C_5
C_4
C_6
B^*
d^h x^t x^h u^t u^h b^t b^h c^t c^h v^t v^h d^t d^h z^t z^h y^t y^h a^t

Figure 4.9: After the optimal integration of the runs from the genomes of Figure 4.8, the two resulting genomes $A^* = \{(a, x, -c, y, b, -z, -d, -v, u)\}$ and $B^* = \{(a, x, u, b, c, v, d, z, y)\}$ have five more common markers than A and B do, but also five more cycles in MG(A^*, B^*).

We now have $\mathcal{G} = \{a, b, c, d, x, y, z, v, u\}$, $\mathcal{A} = \{\}$, $\mathcal{B} = \{\}$, and indeed, $d_{\text{DCJ}}(A, B) = d_{\text{DCJ}}(A^*, B^*)$. ◇

Note that in Example 13 we could also choose to integrate runs in a different order, for example starting with a $\mathcal{B}$-run. Then the two $\mathcal{A}$-runs would be fused such that one further integration of a $\mathcal{B}$-run as well as the integration of an $\mathcal{A}$-run needs to be done.

However impenetrable it seems, if you don't try it, then you can never do it.

ANDREW WILES

CHAPTER 5

Inversion-*indel* Distance Problems Via DCJ(-*indel*) Distance

The *inversion* model allows only inversions (reversals) on genomes with the same genomic content but without duplications. The *inversion distance problem* in genome comparison searches for the minimum number of inversions necessary to transform one unichromosomal genome into another and the *inversion sorting problem* requests a sequence of inversions that achieves this minimum number. Hannenhalli and Pevzner (1995) gave the first algorithm for calculating the inversion distance and solving the inversion sorting problem in polynomial time for two linear genomes [55]. Later (in 2000), it was shown that a similar result holds for circular genomes [71,72].

The DCJ model is less restrictive than the inversion model, allowing more types of rearrangements and chromosomes. Therefore, it can often achieve a shorter scenario than the inversion model. This difference is caused by so-called *bad* instances of the inversion model.

El-Mabrouk (2000) proposed an extension to the inversion model, the inversion-*indel* model, which comprises inversions with insertions and deletions (indels) [42,43]. For the case where there are no insertions, the author presented an algorithm for resolving bad instances of the inversion-*indel* model. Although claimed, this way does not always lead to the minimum number of inversions and deletions possible, as we will see later. Apart from that, the case where additionally insertions are treated was worked out only heuristically.

In 2013, Willing *et al.* showed that when the two genomes have insertions and deletions but none of the bad instances, it is always possible to optimally sort the two genomes without creating bad instances [89]. Therefore, we could show that in these cases the inversion-*indel* distance equals the DCJ-*indel* distance, for which efficient distance formulae and sorting algorithms exist [26] (see also Chapter 4).

In this chapter, we intend to cover all basics concerning the *inversion* as well as the *inversion-indel* model. For this, we identify all necessary analyses on the relational diagram as well as cost and distance considerations. In the style of previous work, we present results for subproblems and special cases (the different subproblems are visualised in Figure 5.1).

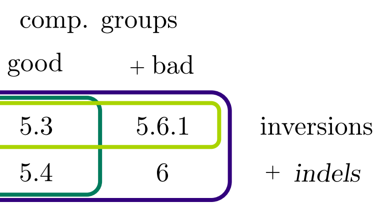


Figure 5.1: Overview of inversion-indel distance subproblems. For each subproblem the chapter/section in which it is covered in this thesis is given.

First, we study the case where none of the bad instances exist, briefly reviewing known results for two genomes with the same marker sets ($\mathcal{A}$ and $\mathcal{B}$ are empty) [54,72] and then showing the solution to genomes with unequal content [89].

We continue with the case in which we have no restriction towards the presence of bad instances. After treating all $\mathcal{AB}$-cycles, we introduce a further abstraction of the employed data structures, the *labelled component group tree*. Moreover, we specify how the data structures and their interrelation are used in computing the solution of the inversion-*indel* distance. We first reproduce the computation of the inversion distance with bad instances using the tree. For the computation of distances including labels, we introduce a novel data structure, the *labelled bad component group tree*, at the end of this chapter. The solution to the general case, without the previously formed restrictions, is presented in the ensuing chapter(s).

5.1 Distance Relations

The two subjects of this chapter, the *inversion* and *inversion-indel* distances, are formally defined as follows.

Definition 14 (Inversion Distance Problem): *Given two unichromosomal circular genomes A and B over the same set of markers $\mathcal{G}$, but without duplications, find the minimum number of steps required to sort A into B using only inversions called the* inversion distance *denoted by $d_{\mathrm{INV}}(A, B)$.*

Definition 15 (Inversion-*indel* Distance Problem): *Given two unichromosomal circular genomes A over $\mathcal{G}_A$ and B over $\mathcal{G}_B$ (without duplications) find the minimum* *number of steps required to sort A into B, using only inversions, insertions and deletions (indels), called the* inversion-*indel* distance *denoted by* $d^{id}_{\text{INV}}(A,B)$.

In context of genome modification models (see Subsection 1.2.2), we have $\mathrm{R} = \{\text{inversion}\}$ and allow $\mathrm{I} = \{\textit{insertion}, \textit{deletion}\}$ for the inversion-*indel* model and restrict content modifications to $\mathrm{I} = \{\}$ for the inversion model.

We seek to find distance values for d_{INV} and d^{id}_{INV}, that may, or may not, include treating of bad instances, w.r.t. the DCJ distance. The generalised genome modification distance from Equation (4.1), for the inversion, resp. the inversion-*indel* distance, is thus broken down to:

$$d_{\text{INV}}(A,B) = d_{\text{DCJ}}(A,B) + \tau_{\text{INV}}(A,B), \quad \text{and} \tag{5.1}$$
$$d^{id}_{\text{INV}}(A,B) = d_{\text{DCJ}}(A,B) + \tau^{id}_{\text{INV}}(A,B), \tag{5.2}$$

such that τ_{INV} and τ^{id}_{INV} give the respective offsets to the DCJ distance. In conjunction with the offset of the DCJ-*indel* distance towards the DCJ distance (see Section 4.4) we could also regard the offset of d^{id}_{INV} (given by τ^{id}_{INV}) as follows:

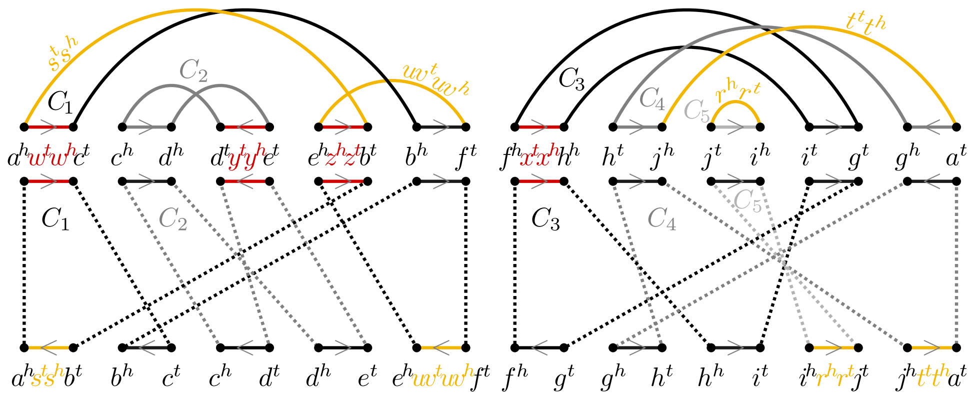
$$d^{id}_{\text{INV}}(A,B) = d^{id}_{\text{DCJ}}(A,B) + \tau^{*}_{\text{INV}}(A,B), \qquad \text{s.t. } \tau^{id}_{\text{INV}} = \tau^{id}_{\text{DCJ}} + \tau^{*}_{\text{INV}}. \tag{5.3}$$

In the current and following chapter(s) we will study and give results to the different subproblems depicted in Figure 5.1. But first, let us lay the ground work for the inversion and inversion-*indel* model.

5.2 Preliminaries

The technicalities presented in this section are fundamental to all variants of inversion models covered in this thesis. An important alteration to previous chapters is the switch from graphs to diagrams, on the grounds that for this chapter the order of vertices is important. Here, a diagram is a fixated view of the corresponding graph, for example the diagram view of the breakpoint graph or the master graph. Unless it is crucial, we use “diagram” for either type and do the analysis on edges or vertices of A (but a similar analysis could be done for elements of B). Likewise, unless of importance, we use a random fixation of the graphs and will show later, in Section 5.6, why this is possible. As we generally consider the inversion model and its variants for unichromosomal circular genomes only, all the components in the diagram are cycles. What follows is mostly a recount of the unified concepts of other data structures and models (e.g. from [12,26,54]) which we presented in [89].

First and foremost we analyse the cycles of the diagram. For this, we assign orientations to all adjacency edges of the diagram as follows: by walking through a cycle, arbitrarily in one of the two possible directions, we assign an orientation to each A- and to each B-edge (see the arrows on the genome edges in the breakpoint graph and the relational diagram in Figure 5.2). Further analysis of the cycles determines their



C_1
C_2
C_3
C_4
C_5
$s^t s^h$
$w^t w^h$
$t^t t^h$
$r^h r^t$
$a^h w^t w^h c^t$ c^h d^h $d^t y^t y^h e^t$ $e^h z^h z^t b^t$ b^h f^t $f^h x^t x^h h^h$ h^t j^h j^t i^h i^t g^t g^h a^t
$a^h s^t s^h b^t$ b^h c^t c^h d^t d^h e^t $e^h w^t w^h f^t$ f^h g^t g^h h^t h^h i^t $i^h r^h r^t j^t$ $j^h t^t t^h a^t$

Figure 5.2: Relational diagram and breakpoint graph of genomes A and B from Example 1 with orientations assigned to adjacency edges. uv represents a single marker.

character. A non-trivial cycle C that has a pair of A-edges with opposing orientations is called a *good* cycle, otherwise it is said to be *bad*. A trivial cycle is sorted and as such considered *good*.

Example 14: In Figure 5.2, C_1 and C_3 are bad cycles, C_2 and C_4 are good cycles and C_5 is a trivial cycle. Concerning the labelling, C_1 is an $\mathcal{AB}$-cycle, C_2 and C_3 are $\mathcal{A}$-cycles and C_4 and C_5 are $\mathcal{B}$-cycles. $\diamond$

Historical note: A pair of edges is called *oriented* if they are opposing, otherwise they are *unoriented*. Analogously, bad cycles are *unoriented* otherwise cycles are *oriented*.

5.2.1 Effect of an Inversion on Cycles

An inversion of a segment of genome A acts on two A-edges of the diagram and reverses the elements in-between. A *split inversion* increases the number of cycles and corresponds to an optimal DCJ operation while *neutral* and *joint inversions* correspond to neutral and counter-optimal DCJ operations, respectively. The A-edges on which an inversion acts can be part of the same cycle or of different cycles. If an inversion is applied to two A-edges of the same cycle it is a split inversion if and only if the edges' orientations are opposing, otherwise this inversion is neutral but turns this cycle into a good cycle [55]. Only good cycles can have opposing A-edges, thus an inversion acting on a single bad cycle can only be neutral. An inversion acting on two distinct cycles always merges them into a single good cycle.

Further, we consider the labelling of edges and cycles. We adopt the concepts of runs and indel-potential from the DCJ-*indel* model presented in the preceding chapter. As in the DCJ-*indel* model, an inversion acting on two labelled edges can always fuse both labels, thus reducing the number of labelled edges by 1. This also affects the number of runs or even the indel-potential. Table 5.1 shows the different combinations of labellings involved in an inversion and the effect of the inversion on both the DCJ distance and the indel-potential.

Example 14 (continued): An inversion acting on A-edges $a^h w^t w^h c^t$ and $e^h z^h z^t b^t$ of C_1 in Figure 5.2 produces new edges $a^h w^t w^h z^t z^h e^h$ and $c^t b^t$. This is a neutral inversion as no new cycle is created. Although C_1 still has the same length as before, the number of runs decreases from 4 to 2, hence $\Delta\lambda(C_1) = -1$. An equivalent result is achieved when, instead, we create $a^h e^h$ and $c^t w^h w^t z^h z^t b^t$. ◇

Table 5.1: Overview of inversions acting on cycles with different indel-types and their impact on the DCJ-*indel* distance. $\mathcal{X}$ stands for a labelling of either $\mathcal{A}$, $\mathcal{B}$, $\mathcal{AB}$ or ε. Cases in which the overall indel-potential λ is increased are omitted.

inversion		indel		indel	fused			Δd_{DCJ}	
type	Δd_{DCJ}	operands	$\rightarrow$	resultant	runs	$\Delta\Lambda$	$\Delta\lambda$	$+\Delta\lambda$	
split	0	$\mathcal{AB}\,(\Lambda > 4)$		$\mathcal{AB}, \mathcal{AB}$	$\mathcal{A}$, $\mathcal{B}$	-2	0	**0**	
				$\mathcal{AB}, \mathcal{B}$	$\mathcal{A}$	-1	0	**0**	
				$\mathcal{AB}, \mathcal{A}$	$\mathcal{B}$	-1	0	**0**	
		$\mathcal{AB}\,(\Lambda = 4)$		$\mathcal{AB}, \mathcal{B}$	$\mathcal{A}$	-1	0	**0**	
				$\mathcal{AB}, \mathcal{A}$	$\mathcal{B}$	-1	0	**0**	**optimal**
		$\mathcal{AB}\,(\Lambda < 4)$		$\mathcal{AB}, \varepsilon$		0	0	**0**	
				$\mathcal{A}, \mathcal{B}$	/	0	0	**0**	
		$\mathcal{X}$ (*other*)		$\mathcal{X}, \varepsilon$		0	0	**0**	
neutral	$+1$	$\mathcal{AB}\,(\Lambda \geq 4)$		$\mathcal{AB}$	$\mathcal{A}$, $\mathcal{B}$	-2	-1	**0**	
		$\mathcal{X}$ (*other*)		$\mathcal{X}$	–	0	0	$+1$	neutral
joint	$+2$	$\mathcal{AB}$ $\mathcal{AB}$		$\mathcal{AB}$	$\mathcal{A}$, $\mathcal{B}$	-2	-2	**0**	**optimal**
		$\mathcal{A}$ $\mathcal{A}$		$\mathcal{A}$	$\mathcal{A}$	-1	-1	$+1$	
		$\mathcal{A}$ $\mathcal{AB}$		$\mathcal{AB}$	$\mathcal{A}$	-1	-1	$+1$	
		$\mathcal{B}$ $\mathcal{B}$		$\mathcal{B}$	$\mathcal{B}$	-1	-1	$+1$	neutral
		$\mathcal{B}$ $\mathcal{AB}$		$\mathcal{AB}$	$\mathcal{B}$	-1	-1	$+1$	
		ε $\mathcal{X}$		$\mathcal{X}$	/	0	0	$+2$	counter
		$\mathcal{A}$ $\mathcal{B}$		$\mathcal{AB}$		0	0	$+2$	optimal

5.2.2 Component Groups

We now analyse cycles on a more abstract level by studying the properties of their relation referring to A-edges. Two distinct cycles C and C' in the relational diagram are said to be *interleaving* when there is at least one A-edge of C between two A-edges of C' and at least one A-edge of C' between two A-edges of C. Looking at the breakpoint diagram this is simple: two cycles are interleaving if there are B-edges (the arcs) of one cycle that intersect B-edges of the other cycle.

A *component group* $\mathcal{K}$ is a maximal set of cycles where there is a sequence of pairwise interleaving cycles from each $C_i \in \mathcal{K}$ to any other $C_j \in \mathcal{K}$. Component groups can be of three types. A trivial cycle can never interleave with another cycle and forms a *trivial* component group. Other component groups are either *good*, if they contain at least one good cycle, or *bad*, otherwise.

Example 14 (continued): The cycles C_1, C_2 and C_5 are not interleaving with any other cycle and form one component group each. Cycles C_3 and C_4 are interleaving but no further cycle interleaves with C_3 or C_4, thus it is a maximal set and they form one interleaving component group. Moreover, $\{C_5\}$ is trivial, $\{C_1\}$ is bad and $\{C_2\}$ and $\{C_3, C_4\}$ are good component groups. $\diamond$

The labelling of a component group represents the labelling of its cycles. A component group consisting of only unlabelled cycles is called an *unlabelled* or *ε-component group*. A component group containing at least one $\mathcal{A}$-label, but no $\mathcal{B}$-label is called an *$\mathcal{A}$-component group*; vice versa for *$\mathcal{B}$-component groups*. Finally, a component group that contains at least one $\mathcal{A}$- and at least one $\mathcal{B}$-label (these can occur in the same cycle, or in different cycles within the component group) is called an *$\mathcal{AB}$-component group*.

Example 14 (continued): $\{C_1\}$, but also $\{C_3, C_4\}$, are $\mathcal{AB}$-component groups, as they have labels in A as well as in B. $\{C_2\}$ is an $\mathcal{A}$-component group and $\{C_5\}$ is a $\mathcal{B}$-component group. $\diamond$

5.2.3 Component Group Relations

The maximal set of interleaving cycles that forms a component group can have relations of different kinds to the other component groups. For now, in order to have a canonical representation of the cycles and their relation, we assume that in the relational diagram the leftmost vertex in A and in B represent the same extremity and likewise the last vertex in A and in B represent the other extremity of that common marker. Hence, we choose some starting point $g \in \{+,-\} \times \mathcal{G}$ and build $\mathcal{R} = \text{MG}(A_g, B_g)$, or, if desired, $\mathcal{R} = \text{BG}(A_g, B)$. We will show later, in Section 5.6, that the choice of start marker or reading direction for the fixation is irrelevant to computing the inversion-*indel* distance.

Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be two component groups in $\mathcal{R}(A,B)$. If each A-edge of $\mathcal{K}_1$ is between the same two A-edges of $\mathcal{K}_2$, the component group $\mathcal{K}_1$ is said to be *nested* within $\mathcal{K}_2$. Otherwise, if neither of $\mathcal{K}_1$ and $\mathcal{K}_2$ is nested in the other, the two component groups are said to be *independent*. Two independent component groups $\mathcal{K}_1$ and $\mathcal{K}_2$ are said to be *linked* if for some marker $m \in \mathcal{G}$, $\mathcal{K}_1$ contains one extremity edge and $\mathcal{K}_2$ contains the other extremity edge of m. The marker m is said to be a *link* of $\mathcal{K}_1$ and $\mathcal{K}_2$. A sequence of $k \geq 1$ linked component groups is called a *chain of size k*. A chain that can not be extended to the left or right by further linked component groups is *maximal*. Let $\mathcal{K}_1$ be a component group of $\mathcal{R}(A,B)$ that has A-edges between two distinct component groups, let these be called $\mathcal{K}_2$ and $\mathcal{K}_3$. When the A-edges of $\mathcal{K}_1$ occur between $\mathcal{K}_2$ and $\mathcal{K}_3$ but also between $\mathcal{K}_3$ and $\mathcal{K}_2$ then $\mathcal{K}_1$ is said to *separate* the two component groups $\mathcal{K}_2$ and $\mathcal{K}_3$.

Example 14 (continued): The component groups $\{C_1\}$ and $\{C_3, C_4\}$ form a chain of size 2 and are linked by marker f. The trivial component group $\{C_5\}$ is nested within $\{C_3, C_4\}$, and $\{C_2\}$ is nested within $\{C_1\}$. Thus $\{C_1\}$ separates $\{C_2\}$ from both $\{C_3, C_4\}$ and $\{C_5\}$. In the same way $\{C_3, C_4\}$ separates $\{C_5\}$ from both $\{C_1\}$ and $\{C_2\}$. ◇

Historical note: An unoriented (i.e. bad) component group that does not separate any pair of bad component groups is called a *hurdle*. In the example above, the component group $\{C_1\}$ is the only hurdle. A hurdle is called a *superhurdle* if its resolution causes another bad component group to become a hurdle.

5.2.4 Effect of an Inversion on Component Groups

The effect that inversions can have on component groups depends on the inversion type (split, neutral or joint inversion) and on both the character and the indel-type of the cycle(s) on which the inversion is performed. In a good component group $\mathcal{K}$ that also contains bad cycles it is always possible to apply one or more split inversions on good cycles of $\mathcal{K}$, s.t. some bad cycle $C \in \mathcal{K}$ becomes good and only then can it be sorted with split inversions [55]. A neutral inversion acting on A-edges of a cycle C in a good or bad component group $\mathcal{K}$, turning it into cycle C', does not change the number of cycles or the set of cycles that are interleaving. However, C' is always good [55] thus, after this inversion, $\mathcal{K}$ is definitely a good component group. A joint inversion acting on A-edges of a cycle C_1 in $\mathcal{K}_1$ and a cycle C_2 in $\mathcal{K}_2$ merges these two cycles into one cycle C_3. All cycles interleaving with C_1 and all cycles interleaving with C_2 are now interleaving with C_3 and thus form a single interleaving component group, let it be called $\mathcal{K}_3$. Furthermore, all component groups that were separating $\mathcal{K}_1$ from $\mathcal{K}_2$ have cycles that are now interleaving with C_3 and thus these component groups also become part of $\mathcal{K}_3$. Moreover, the new cycle C_3 is a good cycle, thus $\mathcal{K}_3$ is a good component group [55], even if all cycles in the concerned component groups were bad cycles before the inversion.

Historical note: Sometimes a neutral inversion is referred to as the *cutting* of a (bad) component group (or hurdle) and a joint inversion acting on two cycles of different component groups is referred to as the *merging* of (bad) component groups (or hurdles) [55].

Effect on labelled component groups. The effect of inversions on different labelled cycles has been studied above. A component group that is composed of cycles of different labellings requires that we choose carefully in which cycle we perform an inversion. Later, we will expand on inversions acting on one or two labelled component groups.

5.3 Resolving Unlabelled Good Components

The instances, in which we can find an optimal DCJ scenario employing only inversions can be evaluated by directly looking at the cycles and component groups of the relational diagram of the pair of genomes. A split inversion is an optimal DCJ operation. This means, as long as we can find a cycle with a pair of opposing A-edges, an optimal inversion scenario uses exactly the same number of steps as would an optimal DCJ sorting scenario. We can find such a pair as long as we find good component groups [55]. It has long been known [54,72] that in this case the inversion distance is given by:

$$d_{\text{INV}}(A,B) = d_{\text{DCJ}}(A,B) = \left|\mathcal{G}\right| - c, \tag{5.4}$$

where c is the number of cycles in $\mathcal{R}(A,B)$, which is a random fixation of the master graph. In terms of Equation (5.1), we have $\tau_{\text{INV}} = 0$.

5.4 Resolving Labelled Good Components

Rearranging the genomic content and inserting or deleting content in the same scenario has to be carried out carefully in order to achieve the shortest scenario, (see Example 2 on page 18). For the inversion-*indel* distance the value of τ_{INV}^{id} poses not only the offset of inversions to the DCJ scenario, but also of insertions and deletions.

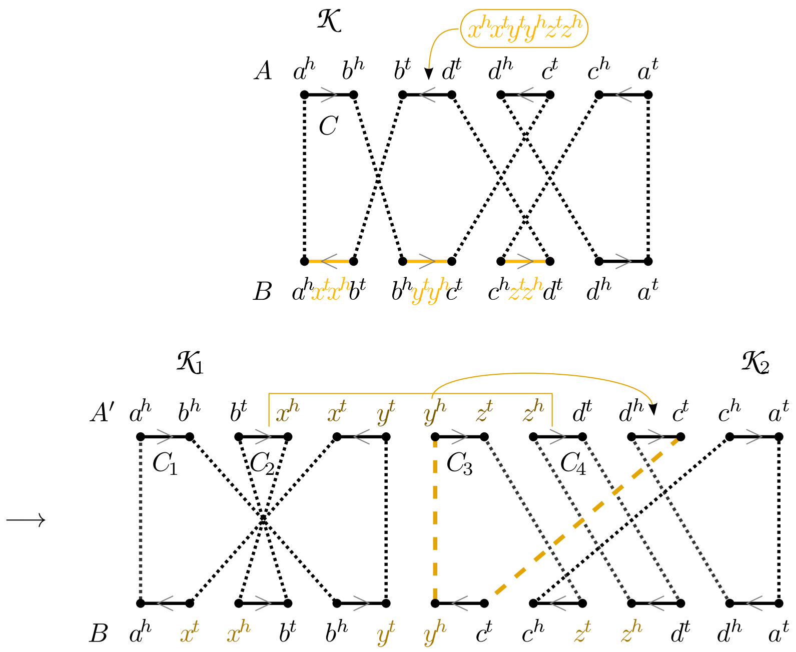
Analogous to the determination of τ_{INV}, we start with the analysis in absence of bad component groups in the diagram view of the master graph. As before, we stick to one arbitrary fixation of the master graph, denoted by $\mathcal{R}(A, B)$. First, we determine how indels, or rather their introduction to the respective opposite genome, affect the presence of bad component groups. The contents of this section have already been published in [89].

We know from before that good component groups can always be sorted without creating bad component groups. The challenge is then to identify instances in which we can perform indels such that the formerly good component groups stay good, and in which we are able to sort a good component group with the minimum number of split inversions and indels possible. The construction, introduction and integration of singletons that represent insertions or deletions was discussed in Subsection 4.5.2 on page 80 and leads to an optimally integrated completion. In the following we elaborate which of these integrations are suitable for our purpose.

5.4.1 Finding Safe Integrations

Assume we have two unichromosomal circular genomes A and B with unequal content, whose diagram $\mathcal{R}(A, B)$ has no bad component group. A *safe integration* is an optimal integration into A resulting in A' such that also $\mathcal{R}(A', B)$ has no bad component group. (The same applies to integrations into B resulting in B' and $\mathcal{R}(A, B')$.) Not all optimal integrations are safe as we show in the following example.

Example 15 (Optimal but unsafe integration): For genomes $A = \{(a, -b, d, c)\}$ and $B = \{(a, x, b, y, c, z, d)\}$ the relational diagram has one $\mathcal{B}$-component group $\mathcal{K} = \{C\}$ that is good and that has exactly one $\mathcal{B}$-run composed of all unique extremities of genome B (Figure 5.3). Among several possibilities, one optimal integration of the singleton $(-xyz)$ produces genome $A' = \{(a, -b, -x, y, z, d, c)\}$. $\mathcal{R}(A', B)$ has one good component group $\mathcal{K}_1 = \{C_1\}$, one bad component group $\mathcal{K}_2 = \{C_3\}$ and two trivial cycles, C_2 and C_3. As this integration produced a bad component group it is not a safe integration. The marker y is a link of $\mathcal{K}_1$ and $\mathcal{K}_2$. ◇



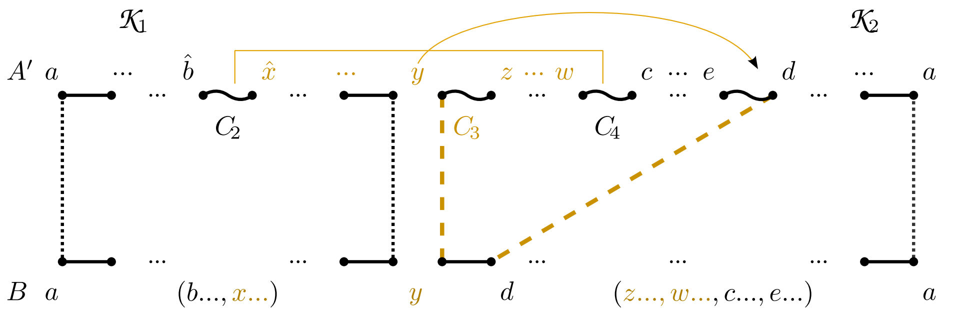
$\mathcal{K}$
$x^h x^t y^t y^h z^t z^h$
A a^h b^h b^t d^t d^h c^t c^h a^t
C
B $a^h x^t x^h b^t$ $b^h y^t y^h c^t$ $c^h z^t z^h d^t$ d^h a^t
$\mathcal{K}_1$
$\mathcal{K}_2$
A' a^h b^h b^t x^h x^t y^t y^h z^t z^h d^t d^h c^t c^h a^t
C_1 C_2 C_3 C_4
$\longrightarrow$
B a^h x^t x^h b^t b^h y^t y^h c^t c^h z^t z^h d^t d^h a^t

Figure 5.3: An optimal integration of singleton $(-xyz)$ to genome A leaves a bad component group $\{C_3\}$. Thus this is not a safe integration.

Even several bad component groups can be created by an optimal integration. Without loss of generality, let all markers in B have the same orientation and let $\mathcal{R}(A, B)$ have only one component group $\mathcal{K}$, that is good. Assume that an optimal integration of a run-singleton s_r to the chromosome of A yielding A' creates, besides one or two trivial component groups, exactly one good component group $\mathcal{K}_1$ and one bad component group $\mathcal{K}_2$ in $\mathcal{R}(A', B)$. A specific example is given in Figure 5.3 and a general illustration of this problem is given in Figure 5.4 (i). If necessary, we can flip genome A' so that the markers within $\mathcal{K}_2$ in A' have the same orientation as the markers in B. Furthermore, due to the circularity of the genomes, we can rotate the diagram so that $\mathcal{R}(A', B)$ is a chain of exactly two linked component groups $\mathcal{K}_1$ and $\mathcal{K}_2$. A link of $\mathcal{K}_1$ and $\mathcal{K}_2$ is within the optimal integration, which, in both illustrations, is marker y.

Fortunately, it is always possible to perform a safe integration, as shown in the following. If we perform an alternative optimal integration of s in the middle of the bad component group $\mathcal{K}_2$ (see Figure 5.4 (ii)), we obtain A''. In $\mathcal{R}(A'', B)$ we have either a single bad component group smaller than $\mathcal{K}_2$, or no bad component group, where the *size* of a component group is the total number of A-edges of its cycles. In general, there can be other component groups in $\mathcal{R}(A', B)$ nested within $\mathcal{K}_1$ and $\mathcal{K}_2$, but each one of these is either trivial or has at least one edge within and at least one edge outside the integrated singleton. In any case, since the component group

(i) An integration yielding $\mathcal{K}_1$ (good) and $\mathcal{K}_2$ (independent and bad). The marker y is a link of $\mathcal{K}_1$ and $\mathcal{K}_2$ and is adjacent to d in genome B.


$\mathcal{K}_1$
$\mathcal{K}_2$
A' a ⋯ $\hat{b}$ $\hat{x}$ ⋯ y $z \cdots w$ $c \cdots e$ d ⋯ a
C_2
C_3
C_4
B a $(b..., x...)$ y d $(z..., w..., c..., e...)$ a

(ii) An optimal integration into $\mathcal{K}_2$, so that y is now adjacent to d in genome A''. An unlabelled trivial cycle C_3' is created.

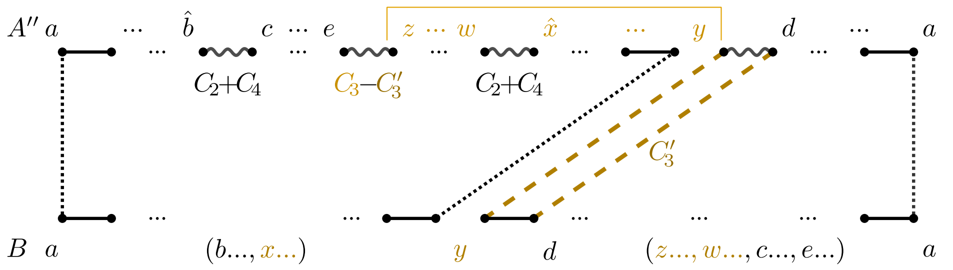

A'' a ⋯ $\hat{b}$ $c \cdots e$ $z \cdots w$ $\hat{x}$ ⋯ y d ⋯ a
$C_2{+}C_4$
$C_3{-}C_3'$
$C_2{+}C_4$
C_3'
B a $(b..., x...)$ y d $(z..., w..., c..., e...)$ a

Figure 5.4: Illustration of how to find an alternative to an unsafe integration. Only the A-edges shaped like $\sim$ from $\mathcal{R}(A', B)$ were transformed into the A-edges drawn as zigzag lines in $\mathcal{R}(A'', B)$. All other edges of the diagram were preserved. Whilst the distinct cycles C_2 and C_4 of $\mathcal{R}(A', B)$ are merged into a single cycle C_2+C_4 in $\mathcal{R}(A'', B)$, the cycle C_3 is split into two cycles (C_3' and C_3-C_3') in $\mathcal{R}(A'', B)$. The hat $\hat{}$ on markers b and x indicates that we make no assumptions about the orientation of these markers (but we know they have the same orientation in A' and A'').

in $\mathcal{R}(A, B)$ was good, at least one component group in $\mathcal{R}(A', B)$ has to be good. By extending the approach illustrated in Figure 5.4 we can show that all component groups but $\mathcal{K}_2$ are merged into a single good component group and only one bad component group, strictly smaller than $\mathcal{K}_2$, can exist in $\mathcal{R}(A'', B)$.

Proposition 6 ([89]): *Let the run-singleton s_r represent one $\mathcal{B}$-run r of $\mathcal{R}(A, B)$. At least one optimal integration of s_r into the chromosome of A is safe.*

Proof. Assume that each optimal integration of s_r to A, resulting in A', creates at least one bad component group in $\mathcal{R}(A', B)$. Then, among all possible optimal integrations of s_r, assume that we take one that produces a bad component group $\mathcal{K}'$ of the smallest size. It is always possible to perform another optimal integration of s_r, as described in Figure 5.4, into the middle of the bad component group $\mathcal{K}'$, transforming A' into A'', so that we create an unlabelled trivial cycle in $\mathcal{R}(A'', B)$. Either $\mathcal{R}(A'', B)$ does not have any bad component group (then we have a contradiction to the assumption that all optimal integrations create bad component groups), or it has a bad component group $\mathcal{K}''$ (then $\mathcal{K}''$ must be strictly smaller than $\mathcal{K}'$, and we have a contradiction to the assumption that $\mathcal{K}'$ was a bad component group of smallest size). □

Corollary 1. Let the run-singleton s_r represent one $\mathcal{A}$-run r of $\mathcal{R}(A,B)$. At least one optimal integration of s_r into the chromosome of B is safe.

This proposition is a major step not only in computing the inversion-*indel* distance on good component groups, but also in offering a procedure for sorting a pair of genomes.

5.4.2 The Inversion-*indel* Distance on Good Components

The results presented above give rise to the following theorem:

Theorem 3 ([89]): *For two unichromosomal circular genomes A and B without duplications, whose relational diagram $\mathcal{R}(A,B)$ has no bad component group, we have*

$$d^{id}_{\text{INV}}(A,B) = d^{id}_{\text{DCJ}}(A,B). \tag{5.5}$$

Proof. We know that there is at least one safe integration for each run and that by integrating one run per step (and updating each run to meet the maximality that defines a run) we perform exactly $\sum_{C\in\mathcal{R}(A,B)} \lambda(C)$ integrations. Afterwards, we have two genomes A^* and B^* that have the same markers and whose diagram $\mathcal{R}(A^*,B^*)$ has no bad component group. We have $d_{\text{DCJ}}(A^*,B^*) = d_{\text{DCJ}}(A,B)$. We already know that $d_{\text{DCJ}}(A,B) = d_{\text{INV}}(A,B)$ in absence of bad component groups, thus also $d_{\text{DCJ}}(A^*,B^*) = d_{\text{INV}}(A^*,B^*)$. In total, with adding the integrations that maintain the indel-potential, we achieve exactly the result given in Equation (5.5). □

This means in absence of bad component groups, the inversion distance that additionally allows indels builds upon only the indel-potential, or, in other terms:

$$\text{Equation (5.2): } \tau^{id}_{\text{INV}} = \tau^{id}_{\text{DCJ}}, \qquad \text{and Equation (5.3): } \tau^{*}_{\text{INV}} = 0.$$

The inversion-*indel* distance formula for two unichromosomal circular genomes A over $\mathcal{G}_A$ and B over $\mathcal{G}_B$ without duplications and whose relational diagram $\mathcal{R}(A,B)$ has no bad component group is then:

$$d^{id}_{\text{INV}}(A,B) = |\mathcal{G}| - c + \sum_{C\in\mathcal{R}(A,B)} \lambda(C), \tag{5.6}$$

where c is the number of cycles in $\mathcal{R}(A, B)$ and $\lambda(C)$ is the indel-potential of cycle C [26,89].

Since the DCJ-*indel* distance can be computed in linear time, the same is true for the inversion-*indel* distance in the absence of bad component groups [89].

5.5 Handling $\mathcal{AB}$-Cycles and $\mathcal{AB}$-Component Groups

We know that in case we have only good component groups in the relational diagram we can find an optimal DCJ-*indel* scenario that employs no other operations than inversions, insertions or deletions. Besides split inversions, Table 5.1 (on page 91) shows two other possibilities that yield $\Delta d_{\text{DCJ}} + \Delta\lambda = 0$ (which is optimal). This is achieved by employing one or two $\mathcal{AB}$-cycles each fusing an $\mathcal{A}$-run and simultaneously a $\mathcal{B}$-run. In the rest of this thesis, we will assume that, in the relational diagram, we have at most one $\mathcal{AB}$-cycle. This can be seen as follows.

Proposition 7: *Given two genomes A and B with relational diagram $\mathcal{R}(A, B)$, it is always possible to apply inversions to $\mathcal{AB}$-cycles such that we achieve a relational diagram $\mathcal{R}(A', B)$ that has at most one $\mathcal{AB}$-cycle, while at the same time conserving the inversion-*indel* distance.*

Proof. Let C_1 and C_2 be two $\mathcal{AB}$-cycles of the relational diagram $\mathcal{R}(A, B)$, then we can always apply a joint inversion that merges a pair of $\mathcal{A}$- and a pair of $\mathcal{B}$-runs, thus reducing the number of runs by 2. The operation has $\Delta d_{\text{DCJ}} = +2$ and turns C_1 and C_2 into a single good cycle C and simultaneously achieves $\Delta\lambda = -2$. Ultimately, this operation has zero extra steps compared to an optimal DCJ-*indel* scenario. Therefore, we can simply apply this kind of joint inversion until, in the end, all $\mathcal{AB}$-cycles are merged into a single large good $\mathcal{AB}$-cycle. In this way, all component groups that contained an $\mathcal{AB}$-cycle as well as all component groups separating them get merged into one big good $\mathcal{AB}$-component group [89].

If in the beginning there is only one $\mathcal{AB}$-cycle C, then C stays bad if and only if we have exactly one $\mathcal{A}$- and one $\mathcal{B}$-run (that is $\Lambda(C) = 2$ and its indel-potential is 2). Otherwise, if $\Lambda(C) > 2$, we can apply a neutral inversion on C such that it merges a pair of $\mathcal{A}$-runs and a pair of $\mathcal{B}$-runs, hence, $\Delta\Lambda = -2$. This turns C into a good cycle, simultaneously achieves $\Delta\lambda = -1$, and ultimately uses the same number of steps as an optimal DCJ-*indel* scenario.

In either way, we can guarantee that after this type of preprocessing, we have at most one $\mathcal{AB}$-cycle and all other $\mathcal{AB}$-component groups have their $\mathcal{A}$- and $\mathcal{B}$-labels in distinct cycles. □

If we have the case that an $\mathcal{AB}$-cycle exists, the $\mathcal{AB}$-component group that contains this cycle is treated in the same way as the other $\mathcal{AB}$-component groups, that have the $\mathcal{A}$- and $\mathcal{B}$-labels in separate cycles. The reason being that in this case a single joint inversion cannot fuse two pairs of runs at the same time. It can either fuse a pair of $\mathcal{A}$-runs or fuse a pair of $\mathcal{B}$-runs. Thus the two types of labels of $\mathcal{AB}$-component groups will not be used concurrently.

We can create new $\mathcal{AB}$-cycles by merging an $\mathcal{A}$- and a $\mathcal{B}$-cycle, but we observe that this procedure is irrelevant and we assume there is at most one $\mathcal{AB}$-cycle.

Observation 6. Merging an $\mathcal{A}$- with a $\mathcal{B}$-cycle (cost 2) and then merging it with a potentially existing $\mathcal{AB}$-cycle (cost 0) has overall cost 2. The same cost can be achieved while merging the $\mathcal{A}$- and the $\mathcal{AB}$-cycle (cost 1) and merging the resulting cycle with the $\mathcal{B}$-cycle (cost 1). Ultimately, both approaches merge the same cycles and separating cycles into a component group. Unlike the first case, the latter can be done even if the $\mathcal{AB}$-component group in question has no $\mathcal{AB}$-cycle.

After the preprocessing of $\mathcal{AB}$-cycles (and potentially reducing the number of bad component groups or even bringing them to zero), we transfer the information of the relational diagram into a more abstract data structure which will later allow us to determine the inversion-*indel* distance of two genomes in the general case.

5.6 The Labelled Component Group Tree

The extra cost for handling bad component groups in the inversion model can be computed using an approach from [12,14], in which a tree structure is defined representing the linking and nesting relationship of the component groups of the master graph. This has been done only for unlabelled instances, but when, at the same time, unique markers are present, the number of indels has to be taken into account and the tree data structure additionally has to represent the labelling of each component group.

We extend the original definition of a *component group tree* from [12] followed by proofs to show that we can compute the value of $\tau^*_{\text{INV}}(A, B)$ from this tree, and that the computation is independent of the rotation of the unichromosomal genomes used for the display of the diagram such as the breakpoint graph or relational diagram. After briefly reviewing the case for unlabelled instances, we introduce a novel procedure to extract the essence of the tree.

Definition 16 (Labelled Component Group Tree [12,89]): *Given two unichromosomal circular genomes A and B without duplications, we construct a temporary* *tree $t(A,B)$ which represents the component groups of $\mathcal{R}(A,B)$ and their relationship among each other, s.t.*

- *$t(A,B)$ has a* round *node for each component group $\mathcal{K} \in \mathcal{R}(A,B)$ representing $\mathcal{K}$ in character (trivial, good or bad) and in labelling (ε-, $\mathcal{A}$-, $\mathcal{B}$- or both $\mathcal{A}$- and $\mathcal{B}$-label).*
- *The children of a round node representing a component group $\mathcal{K}$ are the maximal chains of component groups nested within $\mathcal{K}$.*
- *A maximal chain of component groups is represented by a square node and its children are the round nodes representing the component groups of this chain.*

A square node is either the child of the smallest component group in which this chain is nested, or the root of the tree. The labelled component group tree *$T(A,B)$ is then the* unrooted *version of $t(A,B)$ that has only round nodes as leaves for which, if necessary, a square node is removed if it is a leaf. (T could also be a single vertex.)*

The exact value for $\tau^*_{\text{INV}}(A,B)$ can be computed from the component groups as represented in $T(A,B)$. An inversion acting on a single component group of $\mathcal{R}(A,B)$ turning it into a good component group can be indicated as marking the corresponding node in the labelled component group tree as *covered* by a short path. In the same way, an inversion acting on two component groups of $\mathcal{R}(A,B)$, merging the two component groups, and all the component groups separating these two, into a single good component group, can be seen as covering the involved corresponding nodes in $T(A,B)$ by a long covering path. Since $\tau^*_{\text{INV}}(A,B)$ is the minimum extra cost to handle all bad component groups, in the possible presence of indels, the goal for $T(A,B)$ is to find a set of covering paths which cover all bad vertices (tree cover) but which in total yield minimum cost, thus to find an *optimal tree cover.*

Lemma 1: *Given two unichromosomal circular genomes A and B and the labelled component group tree $T(A,B)$ the cost of an optimal tree cover of T under the inversion-indel model is:*

$$cost\big(T(A,B)\big) \;=\; \tau^*_{\text{INV}}(A,B)$$

where $\tau^*_{\text{INV}}(A,B)$ is the minimum additional cost for inversions to destroy all bad component groups of $\mathcal{R}(A,B)$ in the possible presence of indels.

The remainder of this section is devoted to prove the above lemma. We first show that we always construct the same component groups and subsequently the same tree. Then we show that the cost of an optimal tree cover of that tree gives $\tau^*_{\text{INV}}(A,B)$.

Proposition 8: *Let A and B be two unichromosomal circular genomes without duplications. Then for any circular rotation and/or change of reading direction of genome A and/or B, the component groups in the derived relational diagram are identical.*

Proof. A component group is a maximal set of interleaving cycles. The direction or starting point of reading the chromosomes are irrelevant to the composition of cycles (see Proposition 1, page 25). It remains to prove the same for the composition of all component groups.

A trivial cycle can never interleave with another cycle or separate any two cycles. It thus always forms a trivial component group. Furthermore, two cycles that interleave in one fixation also interleave in any other fixation, as neither direction nor starting point of reading change the alternating order of edges in the two concerned cycles. Hence, the same cycles compose the same component groups in any relational diagram of A and B. Moreover, the labelling of each component group is unaltered (both in orientation of the labels and in position).

Simultaneously, no new vertices, edges, labellings, and thus no new cycles or component groups arise by rotation of the chromosomes or change in reading direction. We have thus shown that the same component groups are present in any fixated view of $\mathrm{MG}(A, B)$. □

In the next step we show that for any fixated view of the master graph, the relations among component groups –represented by edges in $T(A, B)$– are preserved. Recall from Chapter 2 that given a unichromosomal circular genome A over the set of markers $\mathcal{G}$ we derive A_i with $i \in \{+, -\} \times \mathcal{G}$ by starting to read from A in marker i in the proposed direction ($+$ or $-$). Figure 5.5 shows a breakpoint graph plotted in circular arrangement and the corresponding unrooted component group tree and visualises the relation among component groups thus offering an intuitive insight to the proofs of the lemmata and propositions that follow.

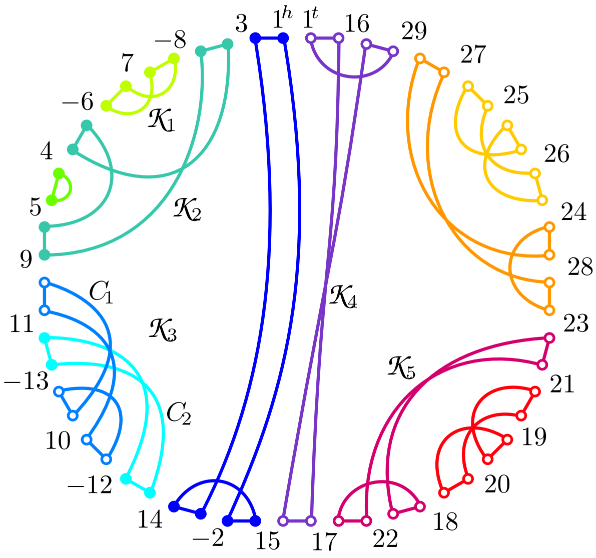
Lemma 2: *Given two unichromosomal circular genomes A and B without duplications, the fixation of the master graph $\mathrm{MG}(A, B)$ to a relational diagram $\mathcal{R}' = \mathrm{MG}(A_g, B_f)$ results in the same relation among component groups for any circular rotation and/or change of reading direction of A and/or B and the following holds:*

$$T(\mathcal{R}') \equiv T\big(\mathrm{MG}(A_i, B_j)\big), \qquad \textit{where } g, f, i, j \in \{+, -\} \times \mathcal{G},$$

in other words: for any choice of fixation of the master graph $\mathrm{MG}(A, B)$ the labelled component group trees are isomorphic.

(i) Breakpoint graph arranged for a circular display. Filled (resp. unfilled) vertices signify good (resp. bad) components. Except for $\{C_1, C_2\}$, each cycle forms its own component group.

(ii) Corresponding unrooted component group tree. Good (resp. bad) component groups are visualised by black (resp. white) round nodes.



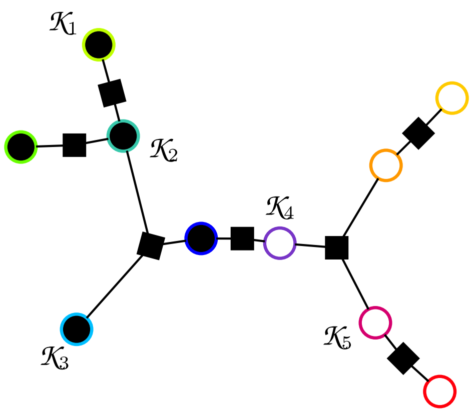


Figure 5.5: Breakpoint graph and unrooted component group tree of genome $A = \{(1, 3, -8, 7, -6, 4, 5, 9, 11, -13, 10, -12, 14, -2, 15, 17, 22, 18, 20, 19, 21, 23, 28, 24, 26, 25, 27, 29, 16)\}$ and the identity.

Proof. From Proposition 8 we know that the component groups are identical, now we prove that their relation is unaltered. From Subsection 5.2.3 we deduce that a component group can have one of the following four types of relations to one or more other component groups: (1) it separates two component groups from each other, (2) it is nested within another component group, (3) it is part of a chain of component groups, or (4) it is independent. As trivial cycles cannot interleave with other cycles or separate a pair of cycles, they each form a trivial component group which is always a leaf. In the following we concentrate on non-trivial component groups.

For the purpose of this proof, w.l.o.g. let us consider two arbitrary fixations $\mathcal{R}' = \text{MG}(A_g, B_f)$ and $\mathcal{R}'' = \text{MG}(A_i, B_j)$ for any $g, f, i, j \in \{+, -\} \times \mathcal{G}$. We show that a component group that has one or more edges connecting it with other nodes in the unrooted tree of $\mathcal{R}'$ will do so in the unrooted tree of $\mathcal{R}''$ as well.

(1) A component group $\mathcal{K}_2$ separates component groups $\mathcal{K}_1$ and $\mathcal{K}_3$ from each other if exactly one among $\mathcal{K}_1$ and $\mathcal{K}_3$ is nested within $\mathcal{K}_2$. In the diagram this is if walking along genome A after meeting edges of $\mathcal{K}_1$ we always meet edges of $\mathcal{K}_2$ before meeting edges of $\mathcal{K}_3$ and then meet again edges of $\mathcal{K}_2$ before meeting edges of $\mathcal{K}_1$. In T this is equivalent to two nodes representing $\mathcal{K}_1$ and $\mathcal{K}_3$ whose

path connecting them includes the node representing $\mathcal{K}_2$. By no rotation of the two genomes or change in reading direction or start marker, will $\mathcal{K}_2$ be moved outside of this path. (This can only happen if adjacencies and therefore the order of markers were altered.)

(2) If a component group $\mathcal{K}_1$ is nested within another component group $\mathcal{K}_2$ where $\mathcal{K}_1$ is a leaf, then clearly $\mathcal{K}_2$ separates $\mathcal{K}_1$ from the rest of the tree, and case (1) applies. If $\mathcal{K}_1$ is not a leaf, then it separates its children from $\mathcal{K}_2$ and other parents/the rest of the tree and case (1) applies.

(3) In a chain of component groups, two neighbouring component groups $\mathcal{K}_1$ and $\mathcal{K}_2$ have no other component groups in-between them but instead have a marker that is a link. Clearly, a change in reading direction does not change the neighbouring elements of a component group in the link. Altering the fixation of the master graph affects at most one point in the chain, a marker that either (i) is a link or (ii) is within one component group of the chain. In case (i) we know that there is no component group separating the elements of a chain from one another. No circular rotation of the genomes can produce a component group that separates two of the elements of the chain. Thus, in T, the elements of the chain are still all directly connected to the same square node. In case (ii) w.l.o.g. let the chain be $\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3$ and let a part of $\mathcal{K}_1$ be circularly moved such that in $\mathcal{R}''$ it appears to the right of $\mathcal{K}_3$ and let the other part of $\mathcal{K}_1$ remain to the left of $\mathcal{K}_2$. Then $\mathcal{K}_1$ becomes a component group in which all other component groups are nested, that includes $\mathcal{K}_2, \mathcal{K}_3$ and the rest of $\mathcal{R}''$ (likewise in $T(\mathcal{R}''))$. The link markers still link the elements. When constructing $T(\mathcal{R}'')$, $\mathcal{K}_1$ would be the only child of the square that is the root and a parent node of a square node whose direct children are the nested component groups (including $\mathcal{K}_2$ and $\mathcal{K}_3$). The root that is a leaf is removed and $\mathcal{K}_1$ is simply another external node attached to the same square node as $\mathcal{K}_2$ and $\mathcal{K}_3$.

(4) Two component groups $\mathcal{K}_1$ and $\mathcal{K}_2$ are independent if in the diagram one is not nested in the other and vice versa. By changing the fixation, one independent component group can become the parent of the other(s) and vice versa two nested component groups can become independent. However, in the trees of $T(\mathcal{R}')$ and $T(\mathcal{R}'')$ there still is the same path between these $\mathcal{K}_1$ and $\mathcal{K}_2$ as the separating component groups as well as links are not altered in order.

Therefore, the relation between component groups is the same for any fixation of $\mathrm{MG}(A, B)$ and the labelled component group trees are identical. $\square$

Hence, for any fixated view of the master graph, the constructed labelled component group tree is the same and the notation $T(A,B)$ is indeed sufficient. When referring to the diagram we can simply choose an arbitrary $\mathcal{R}(A,B)$ as all fixations are equivalent for the purpose of the theory discussed in this thesis.

Corollary 2. Given two unichromosomal circular genomes A and B, for the reason that T is the same, independent of the diagram fixation of $\mathrm{MG}(A,B)$, clearly we also have: $cost\big(T(\mathcal{R}')\big) = cost\big(T(\mathcal{R}'')\big)$ and using $cost\big(T(A,B)\big)$ is sufficient .

In the inversion-*indel* model each neutral or joint operation is assigned a cost according to its effect on both the inversion distance and the overall indel-potential. We transfer the costs of different inversions to cost of covering paths as follows. A neutral inversion on a cycle of a component group u corresponds to covering the vertex in T that represents u by a path of length 1, also called a *short path.* Any neutral inversion has $\Delta\lambda = 0$ (that is after processing all $\mathcal{AB}$-cycles), thus we assign costs of 1 to each short path. Otherwise, if a path consists of more than one vertex, it is called *long.* A long path with end vertices v and w is assigned the minimum cost among all possible joint inversions of cycles from the component group represented by v with cycles from the component group represented by w. From Table 5.1 on page 91 (see also Table 4.1) we learn that the cost is 1 if both v and w have an $\mathcal{A}$-label or both have a $\mathcal{B}$-label (then $\Delta\lambda = -1$), otherwise (when $\Delta\lambda = 0$) the cost is 2 [12,14,89]. A long path then represents the joint inversion acting on two cycles yielding this cost. Since v, w, and all component groups separating v and w, will be part of the new (good) component group after the inversion, they are all covered by the path and do not need to be dealt with separately. In the same way, using vertices whose path covers that of v and w (that are closer to the leaves than v or w are) merges the same bad component groups of the path of v and w and even more.

A set of covering paths that covers all bad nodes of $T(A,B)$ and that has minimum cost gives the minimum cost of ridding the relational diagram of bad component groups (whilst simultaneously considering indels).

Lemma 1 (rephrased): *Given two unichromosomal circular genomes A and B and the labelled component group tree $T(A,B)$, we have:*

$$cost\big(T(A,B)\big) \;=\; \mathcal{T}^{*}_{\mathrm{INV}}(A,B), \tag{5.7}$$

where $\mathcal{T}^{*}_{\mathrm{INV}}(A,B)$ is the minimum cost for destroying all bad component groups of $\mathcal{R}(A,B)$ under the inversion-indel model.

Proof of Lemma 1. By construction of T, no information from the diagram (that is the component groups and their relation) is lost. T is identical for any altering in start point or direction of reading the genomes necessary to construct the relational diagram. Furthermore, the costs of a neutral and a joint inversion on cycles in $\mathcal{R}(A, B)$ is represented by the same cost for covering the same vertices in $T(A, B)$. An optimal collection of inversions acting on the relational diagram is therefore also an optimal collection of the covering paths of $T(A, B)$ representing the same inversions and vice versa. $\square$

Therefore, in the rest of this thesis we will concentrate on one exemplar construction of the tree and finding an optimal tree cover and computing its cost.

5.6.1 Resolving Bad Unlabelled Component Groups

First of all, we reprocess the relation of the inversion and the DCJ distance as given by Equation (5.1). In this case, the value of $\tau_{\text{INV}}(A, B)$ corresponds to the extra cost for applying inversions that are not split inversions. It can be efficiently computed based on the direct analysis of the relational diagram that has unlabelled component groups [55] but as we transferred the cost scheme and relation of component groups to the labelled component group tree we will review the solution based on the tree. For the inversion distance without indels we will consider the tree that has no labels and for the computation of τ_{INV} we need to consider only the bad nodes. Thus we extract the essence of $T(A, B)$ such that any leaf that is not a bad round node is removed [12,55] and we derive the *bad labelled component group tree* $T_{\circ}(A, B)$. Furthermore, a leaf ℓ in $T_{\circ}$ is on a *short branch* if covering ℓ by a short path (done by performing a neutral inversion on a cycle of ℓ hence turning it into a good leaf, and transforming the tree into a bad labelled component group tree again) creates no new bad leaf. Other leaves are on *long branches* and must be covered by long paths that correspond to joint inversions. Beyond that, the example in Subsection A.4 of the appendix reveals that the removal of a leaf on a short branch may eliminate a branching and therefore leaves that were previously on a short branch may now be on a long branch.

It has been shown [12,14] that the cost of an optimal cover of $T_{\circ}$ corresponds exactly to the value $\tau_{\text{INV}}(A, B)$ and can be computed as follows:

Theorem 4 (from [12,14,55]): *Let n be the number of leaves of $T_{\circ}(A, B)$. Then*

$$\tau_{\text{INV}}(A, B) = \begin{cases} n+1 & \text{if } n \text{ is odd and all leaves are on long branches,} \\ n & \text{otherwise,} \end{cases} \tag{5.8}$$

gives the additional cost to the DCJ distance if the only allowed DCJ operations correspond to inversions and the genomes have no unique markers.

Upper bound to inversion distance with labels. The above theorem also gives rise to upper bounds to the inversion distance with unique makers. For example, the upper bound to the distance with unique markers from Equation 2.2 (see page 18) can be re-written as:

$$d^{id}_{\text{INV}}(A,B) \leq d_{\text{INV}}(A,B) + |\mathcal{A}| + |\mathcal{B}|.$$

Furthermore, we can provide an upper bound for value of $\mathcal{T}^{id}_{\text{INV}}$ in two ways that we describe below. Figure 5.6 shows the relation of the offsets between DCJ, inversion

$$\begin{tikzcd}
d_{\text{DCJ}} \arrow[r, "\mathcal{T}_{\text{INV}}"] \arrow[d, "\mathcal{T}^{id}_{\text{DCJ}}"'] \arrow[dr, "\mathcal{T}^{id}_{\text{INV}}"'] & d_{\text{INV}} \arrow[d, "\mathcal{T}^{id}_{*}"] \\
d^{id}_{\text{DCJ}} \arrow[r, "\mathcal{T}^{*}_{\text{INV}}"] & d^{id}_{\text{INV}}
\end{tikzcd}$$

Figure 5.6: The relation of inversion, DCJ and indel operations.

and indel operations and is used to illustrate the two approaches.

One possibility to provide an upper bound to $\mathcal{T}^{id}_{\text{INV}}$ is to perform first all insertions and deletions while maintaining the indel-potential, thus $\lambda(A,B)$. The result is two genomes A^* and B^* that have only common markers. Then we sort A^* and B^* with $d_{\text{INV}}(A^*,B^*)$ inversions. Re-written, this gives the following upper bound:

$$d^{id}_{\text{INV}}(A,B) \leq \mathcal{T}^{id}_{\text{DCJ}}(A,B) + \big(d_{\text{DCJ}}(A^*,B^*) + \mathcal{T}_{\text{INV}}(A^*,B^*)\big).$$

Or the other way around, first we perform all extra inversions in the tree ignoring labels. This takes $\mathcal{T}_{\text{INV}}(A|_{\mathcal{G}}\,, B|_{\mathcal{G}})$ steps and results in genomes A' and B' that have unique markers but whose diagram $\mathcal{R}(A',B')$ has no bad component group. Then, as shown in Section 5.4, we perform all remaining indels and split inversions which additionally costs $d^{id}_{\text{DCJ}}(A',B') = d^{id}_{\text{INV}}(A',B')$, overall resulting in the following upper bound:

$$d^{id}_{\text{INV}}(A,B) \leq \mathcal{T}_{\text{INV}}(A|_{\mathcal{G}}\,, B|_{\mathcal{G}}) + d_{\text{DCJ}}(A',B') + \mathcal{T}^{id}_{\text{DCJ}}(A',B').$$

The approach from El-Mabrouk [43] which seeks for the lowest cost to remove bad component groups and considers deletions (no insertions) goes about the problem via the inversion problem, i.e. computing the inversion distance as if the tree were unlabelled, adding the indel-potential of the initial genomes but subtracting 1 for each path from the tree cover that connects two labelled vertices in its sum denoted by σ:

$$d^{del}_{\text{INV}}(A,B) = d_{\text{INV}}(A|_{\mathcal{G}}\,, B) + \mathcal{T}^{id}_{\text{DCJ}}(A,B) - \sigma(tree).$$

(the correction term σ can be seen as representing the fusion of runs from Table 5.1 on page 91.) However, the computation of an (optimal) cover (thus also of σ) is incorrect which we will discuss in the following subsection.

5.6.2 The Bad Component Group Tree $T_{\circ}$

In a similar manner as for the unlabelled case El-Mabrouk [43] approached solving the inversion-*deletion* distance which also concentrated on the tree that has only bad leaves, ignoring the good leaves (although in a different notation, prior to the introduction of this tree).

Extracting the essence of the labelled component group tree T, by removing unnecessary (good) nodes (which are resolvable without extra costs regardless of their labelling as we showed in Section 5.4) is not as simple in a labelled tree as it was for the unlabelled tree. As the following example shows, not only bad leaves are necessary. Instead, also good leaves –more precisely their labels– can play a vital role in computing an optimal tree cover.

Example 16: The breakpoint graph of the two unichromosomal circular genomes $A = \{(1, 6, 2, 4, x, 3, 5, 7, 10, 8, -z, 9)\}$ and the identity ($B = \{(1, \ldots, 10)\}$) is shown in Figure 5.7 (i). The three bad cycles (C_1, C_2 and C_3) and the trivial cycle (C_4) of $\mathrm{BG}(A, B)$ are not interleaving cycles, thus they each form their own component group $\mathcal{K}_i = \{C_i\}$. Two of the component groups are separated by both the others, one is bad and labelled and one is trivial and labelled. The overall indel-potential is 2. The component group tree is constructed as in Figure 5.7 (ii). If we remove all external good nodes (thus $\mathcal{K}_4$) we have a tree with only bad leaves ($\mathcal{K}_1$ is labelled and $\mathcal{K}_3$ is unlabelled). We refer to the tree with this rigorous deletion of good leaves as $T_{\boxtimes}$.

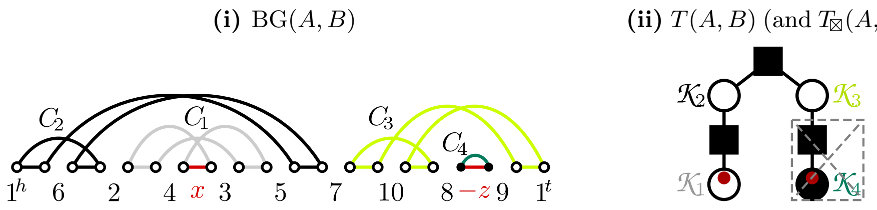


Figure 5.7: Labelled breakpoint graph and labelled component group tree of genome $A = \{(1, 6, 2, 4, x, 3, 5, 7, 10, 8, -z, 9)\}$ and the identity over $\mathcal{G} = \{1, \ldots, 10\}$.

Performing an inversion on genome A that reverses the interval $[3, 5, 7, 10]$ yields $A' = \{(1, 6, 2, 4, x, -10, -7, -5, -3, 8, -z, 9)\}$ which corresponds to a covering path as shown in Figure 5.8 (i) by the dashed line. As can be seen in the breakpoint graph $\mathrm{BG}(A', B)$ (Figure 5.8 (ii)) the inversion merges cycles C_1 and C_3 that now interleave

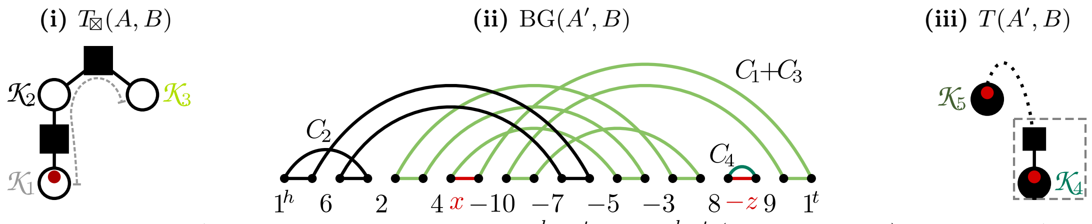


Figure 5.8: An inversion acting on edges $4^h x 3^t$ and $10^h 8^t$ (i.e. $\mathcal{K}_1$ and $\mathcal{K}_3$) of genome A yields genome $A' = \{(1, 6, 2, 4, x, -10, -7, -5, -3, 8, -z, 9)\}$. We destroyed all bad component groups and are left with two good $\mathcal{A}$-component groups.

with C_2 forming the component group $\mathcal{K}_5 = \{C_2, C_1 + C_3\}$. $\mathcal{K}_5$ is a good $\mathcal{A}$-component group, thus there are no more bad component groups left. The joint inversion reduces the number of cycles by 1, uses up one step of the sorting scenario and the overall indel-potential remains 2. $T(A', B)$ has two good $\mathcal{AB}$-nodes (both are leaves).

However, restricting inversions that destroy bad component groups to adjacency edges of bad component groups may prevent us from finding a scenario that uses fewer steps. If instead of $\mathcal{K}_3$, we apply the other cut of the inversion in $\mathcal{K}_4$, as indicated in Figure 5.9 (i), reversing the section $[3, 5, 7, 10, 8, -z]$, we get $A'' = \{(1, 6, 2, 4, x, z, -8, -10, -7, -5, -3, 9)\}$. BG$(A'', B)$ in Figure 5.9 (ii) shows the resulting cycles.

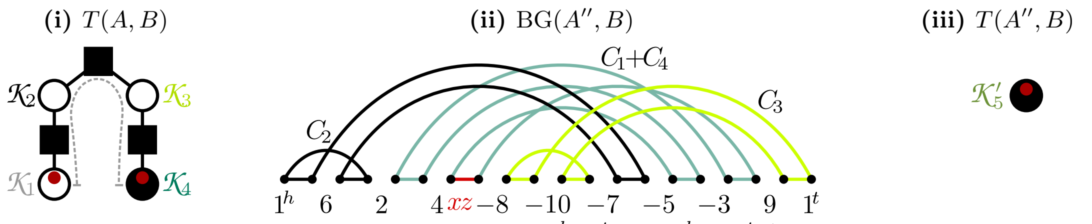


Figure 5.9: An inversion acting on edges $4^h x 3^t$ and $8^h -z 9^t$ (i.e. $\mathcal{K}_1$ and $\mathcal{K}_4$ which is trivial) of genome A yields genome $A'' = \{(1, 6, 2, 4, x, z, -8, -10, -7, -5, -3, 9)\}$. We destroyed all bad component groups and are left with a single good $\mathcal{A}$-component group.

Here, C_4 (a labelled trivial cycle) was merged with C_1 (a labelled bad cycle) and since both have labels only in A, the resulting cycle is a good $\mathcal{A}$-cycle. It is apparent that all cycles that formerly separated C_1 and C_4 are now interleaving and form a single $\mathcal{A}$-component group $\mathcal{K}_5' = \{C_2, C_3, C_1 + C_4\}$ that is good. The component group tree has now only one node which is good and $\mathcal{A}$-labelled.

We hence eliminated all bad component groups by a single joint inversion (using up one step of the sorting scenario) merging two $\mathcal{A}$-cycles thus reducing the overall indel-potential by 1 at the same time. ◇

El-Mabrouk presented a procedure for solving the inversion-*deletion* distance problem only using inversions acting on leaves of $T_{\boxtimes}$ (as constructed in the example above) [43]. Thus this procedure might produce the minimum number of bad component group merges necessary to eliminate all bad component groups when allowing indels, but not always the overall minimum number of steps (inversions, insertions and deletions).

As even the "exact" algorithm for cases in which unique markers occur only in one of the genomes (the inversion-*deletion* distance) presented in [43] is incorrect, we will not study the heuristic approach of that same publication for the case when we have unique markers in both genomes. Instead, we will give our results on the inversion-*indel* distance problem in the ensuing chapter. But first, we present our solution to circumvent the shortcomings of $T_{\boxtimes}(A, B)$ and to simplify the determination of $\tau^*_{\text{INV}}(A, B)$.

Transforming T into $T_{\circ}$

Instead of simply removing good leaves as done in [12], we designed a procedure keeping all necessary information in a condensed version of the tree. This is done by transforming it into the unrooted *bad component group tree* $T_{\circ}(A, B)$ as follows.

Initially, let $T_{\circ} = T$. Now, from external nodes inwards, leaves are removed from the tree if they are not bad round nodes. Thereby, the node u to which the removed leaf v was formerly connected is assigned the union of the labels of u and v. When not only the distance value but also an optimal sorting sequence is desired, a reference to the component groups that contain these labels needs to be upheld. In the end of the transformation, all leaves of $T_{\circ}$ are bad round nodes.

Example 17: The labelled component group tree $T(A, B)$ given in Figure 5.10 (left) is transformed into the corresponding bad component group tree $T_{\circ}(A, B)$ (right). For this, the leaves that are not bad are successively removed and their labels pushed inwards. The subtree rooted at v_1 contains no bad vertices, such that during removal of the good nodes v_1 gets assigned the union of labels of all its children and the children are removed. However, because v_1 is then a leaf itself (that is also good), the labels get pushed inwards even further (to v_2) and v_1 is ultimately removed. In the end $T_{\circ}(A, B)$ has only bad leaves. $\diamond$

We do not lose information of $T(A, B)$ when transforming it into $T_{\circ}(A, B)$, as shown by the following lemma.

Lemma 3: *Given two unichromosomal circular genomes A and B, let $T(A, B)$ be the labelled component group tree of A and B and let $T_{\circ}(A, B)$ be the bad component group*

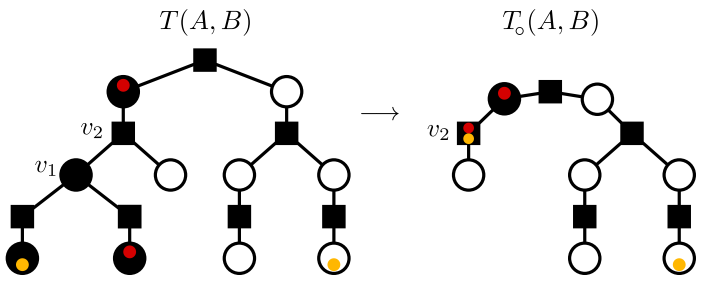


Figure 5.10: An example of a labelled component group tree $T(A,B)$ (left) and the corresponding bad component group tree $T_\circ(A,B)$ (right). The bad nodes are drawn in white. An $\mathcal{A}$-label (resp. $\mathcal{B}$-label) is represented by a red top-aligned (resp. yellow bottom-aligned) dot. The hierarchical structure matches that of the example for a breakpoint graph in Figure 5.5.

tree derived from $T(A,B)$. Then we have

$$cost\big(T_\circ(A,B)\big) \;=\; cost\big(T(A,B)\big). \tag{5.9}$$

Proof. Any set of paths yielding an optimal cover of T also yields an optimal cover of $T_\circ$. The only difference is that the labels of good nodes may now be attributed to a different node (thus shortening the covering path, but still covering the same bad nodes at the same cost). A node that gets a label during transformation of T to $T_\circ$ ultimately references any covering path ending there to an adequate component group (depending on the label). As all $\mathcal{AB}$-cycles have been merged already, $\mathcal{AB}$-nodes in T and $T_\circ$ are treated equally.

On the other hand, any optimal cover of $T_\circ$ can be found in T by using the same vertices or their respective good labelled nodes (at the same cost elongating the paths covering more good nodes). Therefore, the cost of the covers are identical. □

Corollary 3. $\tau^*_{\text{INV}}(A,B)$ is given by $cost\big(T(A,B)\big)$, thus clearly:

$$cost\big(T_\circ(A,B)\big) = \tau^*_{\text{INV}}(A,B).$$

On a side note: If the good leaves in $T(A,B)$ in Figure 5.10 (left) both had $\mathcal{B}$-labels, then v_1 (and therefore also v_2) would have only a $\mathcal{B}$-label. A path in $T_\circ(A,B)$ that uses the label in v_2 then can be found in $T(A,B)$ by choosing any of the $\mathcal{B}$-labelled vertices that lead to the labelling in $T_\circ(A,B)$, as the costs are identical and they both cover the same bad vertices.

Lemma 3 allows us, from now on, to consider simply $T_\circ(A,B)$ instead of $T(A,B)$ and to concentrate on computing $cost\big(T_\circ(A,B)\big)$ which we study in the ensuing chapter.

5.7 Chapter Summary

The two models discussed in this chapter are the inversion model and the generalisation, the inversion-*indel* model. The relation of these two and the DCJ and DCJ-*indel* distance is reflected as given earlier:

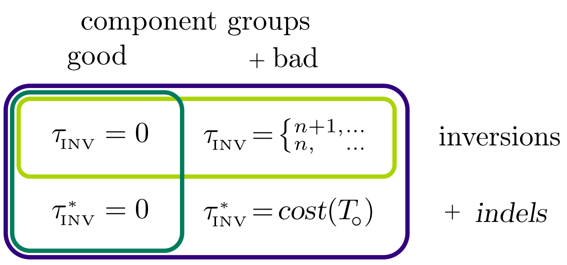
$$\text{Equation (5.1):} \quad d_{\text{INV}}(A,B) = d_{\text{DCJ}}(A,B) + \tau_{\text{INV}}(A,B),$$
$$\text{Equation (5.3):} \quad d^{id}_{\text{INV}}(A,B) = d^{id}_{\text{DCJ}}(A,B) + \tau^{*}_{\text{INV}}(A,B), \quad \text{s.t.} \quad \tau^{id}_{\text{INV}} = \tau^{id}_{\text{DCJ}} + \tau^{*}_{\text{INV}}.$$

Here, τ^{id}_{INV} gives the number of extra operations when allowing indels and using only DCJ operations that are inversions, while τ^{*}_{INV} gives the number of extra operations with respect to the DCJ-*indel* distance.

The solution to the inversion distance has long been known. In the absence of bad component groups an optimal inversion sorting scenario has the same number of steps as an optimal DCJ sorting scenario (Equation (5.4)). We were able to show that when no bad component group exists, we can always find an optimal sorting scenario allowing only inversions and indels that has the same number of steps as an optimal scenario employing DCJ and indel operations (given in Equations (5.5) and (5.6)).

Furthermore, all $\mathcal{AB}$-cycles can be merged into a single $\mathcal{AB}$-cycle without extra cost. This way the number of bad component groups may be reduced or even brought to zero. In the presence of bad component groups the inversion distance can be computed using a tree structure (Equation (5.8)), which we generalised to include labels. Furthermore, we introduced a procedure to derive a novel data structure, the bad labelled component group tree $T_{\circ}$, and showed that an optimal tree cover of $T_{\circ}$ gives the extra cost to handle bad component groups in labelled instances.

Concluding this chapter, we have the following solutions to subproblems, where the ensuing chapter deals with finding the value of $cost(T_{\circ})$.


component groups
good
+ bad
$\tau_{\text{INV}} = 0$
$\tau_{\text{INV}} = \begin{cases} n+1, & \dots \\ n, & \dots \end{cases}$
inversions
$\tau^{*}_{\text{INV}} = 0$
$\tau^{*}_{\text{INV}} = cost(T_{\circ})$
+ *indels*

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

Sherlock Holmes

Chapter 6

Optimal Tree Covers of $\boldsymbol{T}_{\circ}$

In this chapter, we continue to study the *inversion-indel model*, that is the model that allows inversions, insertions and deletions on unichromosomal circular genomes without duplications.

In the previous chapter we determined solutions to the distance problem where one or more restrictions on the genomes were made (see facing page for an illustration). We studied the cases where no bad component group is present and were able to show that in these cases we can always find optimal DCJ or DCJ-*indel* scenarios whose DCJ operations consist only of inversions (thus, that the inversion distance is equal to the DCJ distance and likewise the inversion-*indel* distance has the same value as the DCJ-*indel* distance). The case where bad component groups exist has been solved for the inversion distance problem. However, since the first attempt to solve the inversion-*indel* distance problem in 2000 [42] only heuristics have been presented. It remains to be presented an exact solution to the general inversion-*indel* distance problem, i.e. where bad component groups are present and there are unique markers in one or both genomes which we address in this chapter. For this, we generalised the (bad) component group tree to include labels and extended the cost function for covering paths in the tree.

It is the goal of this chapter to find an optimal tree cover of the labelled component group tree that gives the extra cost to handle bad component groups in the presence of labels, i.e. to determine $\mathcal{T}^{id}_{\mathrm{INV}}$. First, we give bounds to the cost of an optimal cover based on a specific tree property. However, finding an optimal cover proves to be intricate, as we identify different vital properties of the bad labelled component group tree that influence the overall cost of the cover. In the third section we determine how many paths of the same kind can be used in an optimal cover before, in the fourth section, we introduce yet another derived tree, the *residual tree* that allows us to concentrate on a finite set of canonical cases which we subsequently study. We conclude this chapter with a summary of the findings that lead to computing an optimal cover and its cost.

6.1 Covering Paths and Tree Covers

We previously determined the costs of different covering paths in the inversion-*indel* model. We aim to find a set of paths that covers all bad vertices of the tree and that in total yields minimum cost, thus an optimal tree cover. Several co-optimal covers may exist, i.e. different collections of covering paths that cover all bad nodes of the tree and that all yield the minimum cost.

Assume we have given two genomes A and B without duplications from which we construct the bad labelled component group tree $T_\circ(A, B)$. We have to consider four different kinds of labellings for the vertices. Let a node in $T_\circ(A, B)$, representing a component group $\mathcal{K}$, be addressed as $\mathcal{A}$-, $\mathcal{B}$-, ε- or $\mathcal{AB}$-node if $\mathcal{K}$ is an $\mathcal{A}$-, $\mathcal{B}$-, ε- or $\mathcal{AB}$-component group, respectively. Then let $n_\mathcal{A}$, $n_\mathcal{B}$, n_ε and $n_\mathcal{AB}$ denote the number of $\mathcal{A}$-, $\mathcal{B}$-, ε- and $\mathcal{AB}$-leaves in $T_\circ(A, B)$, respectively, forming the *leaf composition* $\mathcal{L}(A, B) := (n_\mathcal{A}, n_\mathcal{B}, n_\varepsilon, n_\mathcal{AB})$. The total number of leaves is $n = n_\mathcal{A} + n_\mathcal{B} + n_\varepsilon + n_\mathcal{AB}$. Along with the types of vertices possible, also the types of short and long paths increased (compared to the inversion model from Subsection 5.6.1).

Cost of covering paths. Destroying bad component groups requires neutral or joint inversions of which, since the tree is constructed after all $\mathcal{AB}$-cycles were treated, we are left with the inversions that amount to $\Delta d_{\text{DCJ}} + \Delta\lambda > 0$. (For a review see Section 5.5 and Table 5.1 on page 91.) Short covering paths represent neutral inversions that cost 1. Long paths cost 1 if they represent joint inversions that act on two cycles that both have $\mathcal{A}$-labels or both have $\mathcal{B}$-labels. Otherwise, if the long path represents an inversion on two cycles that do not have labels in the same genome, the cost is 2. Since the cost for $\mathcal{A}$- and $\mathcal{B}$-labels is symmetric, we can assume w.l.o.g. that $n_\mathcal{A} \geq n_\mathcal{B}$.

Previously, in the inversion model without labels, the cost relied on the number of leaves in the tree (denoted by n) and whether or not a leaf on a short branch exists. The same criteria are not sufficient when computing the cover cost with labels, as the following example shows.

Example 18: We study the bad labelled component group trees shown in Figure 6.1 that all have leaf composition $\mathcal{L}(A, B) = (2, 2, 0, 0)$ but give different optimal cover cost, depending on other properties of the trees. Let us analyse the optimal covers for the different trees.

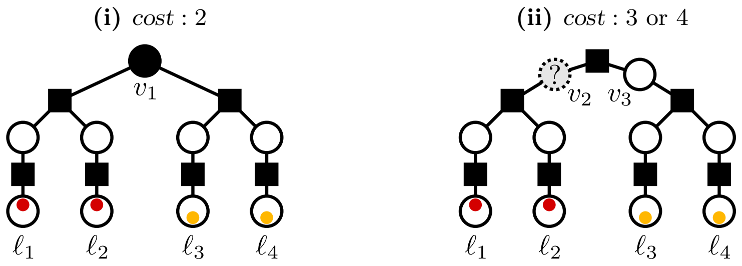


Figure 6.1: Different bad labelled component group trees that have two $\mathcal{A}$-leaves and two $\mathcal{B}$-leaves, but that have different optimal cover cost, further determined by other properties of the tree(s).

(i) The tree has a good vertex v_1 that does not need to be covered. A complete cover uses two long labelled paths: one connecting ℓ_1 with ℓ_2 and one connecting ℓ_3 with ℓ_4. The overall cost is 2 and no cover of lower cost exists.

Depending of the properties of vertex v_2, an optimal cover of the tree shown in Figure 6.1 (ii) can have cost 3 or 4.

(ii-a) If v_2 is a good vertex, it does not need to be covered but we still need to cover v_3. Using the same cover as for tree (i) and additionally covering vertex v_3 by a short path, which always has cost 1, forms a complete optimal cover of cost 3.

(ii-b) If v_2 is an unlabelled bad vertex, it also needs to be covered. Any optimal cover we can obtain has cost 4. For example, we could use the cover of (i) and additionally cover v_2 and v_3 by a long unlabelled path that has cost 2. An alternative optimal cover connects ℓ_1 with ℓ_3 and ℓ_2 with ℓ_4.

(ii-c) If v_2 is bad but has a $\mathcal{B}$-label, we can achieve cost 3. This can be done by using the same cover as for tree (i) and additionally we can cover v_2 and ℓ_3 with a long labelled path that has cost 1. (A similar cover can be achieved if the $\mathcal{B}$-label is closer to ℓ_1 or ℓ_2.)

$\diamond$

This shows that a cost formula depending only the tree properties that were identified for unlabelled trees or the leaf composition $\mathcal{L}(A, B)$ cannot exist. Thus a straightforward approach is not as simple and we first determine what paths compose an optimal cover.

The effect of choosing a single path p is best described using the following covering step. In concordance with the underlying inversion operation (see Subsection 5.6.2), all nodes covered by p are merged into a single good vertex representing the labelling of this path. For simplicity let us call this vertex a *meta-vertex*. All edges attached to any vertex of the path are then attached to the meta-vertex. If the meta-vertex is a leaf, the tree needs to be transformed into a bad labelled component group tree again.

Definition 17 (Safe/unsafe covering path): *A covering path p is called* safe *if removing the covered vertices from $T_\circ$, and transforming the tree into a labelled component group tree again, does not cause a bad node that was previously internal to become a leaf. If it does, p is* unsafe*.*

An example of an unsafe covering path can be found in Figure A.1 (see page 181) and an example of a safe covering path is given in Figure A.2 (see page 182) in the appendix. In the inversion model, an optimal cover sometimes contains a path that is unsafe (an example of a tree for which this is the case can be found in Figure A.8 (i) on page 185 and is laid out in the explanation of Equation (6.3) on the facing page).

Definition 18 (Optimal covering path): *A covering path p is called* optimal *if it is part of an optimal cover C, i.e. if it satisfies*

$$cost\big(\mathsf{C}\big) = cost\big(\mathsf{C}\backslash\{p\}\big) + cost(p). \tag{6.1}$$

Finding an optimal cover of $T_\circ(A, B)$ can be approached by finding one optimal covering path after the other, until all bad vertices are covered, thus until the cover is *complete*. For this we proceed with grouping the different paths satisfying Equation (6.1) and their costs.

Definition 19 (Homogeneous/heterogeneous path): *Let a long path in $T_\circ(A, B)$ start in vertex u and end in vertex v. If u does not have the same labelling ($\mathcal{A}$, $\mathcal{B}$, ε or $\mathcal{AB}$) as v then the path is called* heterogeneous*. Any other (short or long) path is called* homogeneous*.*

For convenience, we denote the different long paths with the labelling of their end nodes, for instance $\mathcal{A}$-$\mathcal{AB}$-path or ε-$\mathcal{AB}$-path. Initially, we study only homogeneous paths. Let $P_\mathcal{A}$, $P_\mathcal{B}$, P_ε and $P_\mathcal{AB}$ be the sets of safe optimal homogeneous paths for $\mathcal{A}$, $\mathcal{B}$, ε and $\mathcal{AB}$ respectively. We denote by C_r the set of paths of an optimal cover C that are not included in $P_\mathcal{A}$, $P_\mathcal{B}$, P_ε or $P_\mathcal{AB}$, i.e., we have:

$$cost\big(\mathsf{C}\big) - cost(P_\mathcal{A}) - cost(P_\mathcal{B}) - cost(P_\varepsilon) - cost(P_\mathcal{AB}) = cost\big(\mathsf{C}_r\big). \tag{6.2}$$

In some cases we can achieve a complete optimal cover consisting solely of safe optimal homogeneous paths, as the following section shows.

6.1.1 Cost of a Cover Bounded by Individual Subtrees

Let us first consider a tree whose leaves are all of the same type. This might also be a subtree of $T_\circ(A, B)$. Then we define:

Definition 20 (Subtree): *Given a bad component group tree $T_\circ(A, B)$, we denote by $T_\circ^{\mathcal{A}}$, $T_\circ^{\mathcal{B}}$, $T_\circ^{\varepsilon}$ and $T_\circ^{\mathcal{AB}}$ the subtrees induced by a single leaf type, i.e. by all $\mathcal{A}$-, $\mathcal{B}$-, ε- and $\mathcal{AB}$-leaves, respectively. A subtree may be empty or consist only of a single round node.*

A *combined subtree* of more than one type is denoted by the set of the partaking types. For instance $T_\circ^{\{\mathcal{A},\varepsilon\}}$ is the subtree induced by all $\mathcal{A}$- and all ε-leaves.

Unlabelled subtree. In the unlabelled case (for $T_\circ^{\varepsilon}$), when we could use more than one short path, we simply replace a pair of short paths by a long path such that at most one leaf is covered by a short path, which is always optimal. Hence, if there exists an ε-leaf on a short branch or for an even number of ε-leaves, all optimal paths are safe. Otherwise, if we have an odd number of leaves and no ε-leaf is on a short branch, one optimal path is unsafe. The cost of an optimal cover for an unlabelled tree is equal to the handling of bad component groups in the inversion distance problem (see Sections 5.3 and 5.6.1). The overall cost of $T_\circ(A, B)$ with $\mathcal{L}(A, B) = (0, 0, n_\varepsilon, 0)$ yields the well known formula [14,55] and is the same as treating $T_\circ^{\varepsilon}$ with n_ε leaves independently of the remaining tree:

$$cost\big(T_\circ^{\varepsilon}(A, B)\big) = \begin{cases} n_\varepsilon + 1 & \text{if } n_\varepsilon \text{ is odd and all leaves are on long branches,} \\ n_\varepsilon & \text{otherwise.} \end{cases} \tag{6.3}$$

Labelled subtree. Now, let us consider the case in which we have labelled leaves. Let the subtree $T_\circ^{\mathcal{A}}$ have $n_{\mathcal{A}}$ leaves, then, in an optimal cover, there will be at most one short path, as we must replace two short paths (cost 2) by a single long covering path (cost 1). The number of leaves used for safe homogeneous paths is the same as above. Additionally, if $n_{\mathcal{A}}$ is odd, we use a single path to cover the last leaf, where the length of the path depends on the length of the branch but yields cost 1 in either case. For a (sub-)tree that has only $\mathcal{A}$-leaves the cover cost is given by:

$$cost(T_\circ^{\mathcal{A}}(A, B)) = \left\lceil \frac{n_{\mathcal{A}}}{2} \right\rceil . \tag{6.4}$$

Since the cost for $\mathcal{B}$-labelled long and short paths are the same as for $\mathcal{A}$-labelled long and short paths, a similar result holds for the $\mathcal{B}$-subtree. Furthermore, we have already established in Section 5.5 that nodes with two labels use only one label at a time, either the $\mathcal{A}$- or the $\mathcal{B}$-label. The optimal cover cost for a (sub-)tree that has only $\mathcal{AB}$-leaves amounts to the same as using the cover of the $\mathcal{A}$- or $\mathcal{B}$-subtree.

Moreover, these results immediately lead to lower and upper bounds to the cost of an optimal cover of a tree $T_\circ$ that has one or more leaf types. Although proposed in [89] note that the proof was incorrect.

Proposition 9: *Let $T_\circ(A, B)$ be a bad component group tree with n leaves. Then the cost of an optimal cover is bounded as follows:*

$$\left\lceil \frac{n}{2} \right\rceil \;\leq\; cost(T_\circ(A, B)) \;\leq\; n + 1. \tag{6.5}$$

Proof. A tree with n leaves needs at least $^{n}/_{2}$ covering paths, if n is even, or at least $^{n+1}/_{2}$ covering path, if n is odd. The minimum cost for a long path is 1, thus the minimum cost of a tree cover is $\lceil ^{n}/_{2} \rceil$. On the other hand, the maximum cost of a long path is 2, thus a cover for a tree with an odd n costs at most $2 \cdot {}^{n+1}/_{2} = n + 1$. $\square$

The lower bound is met, for example, when all leaves share at least one label (in this case, all paths of the cover have cost 1). The upper bound is met, for example, when n is odd, all leaves are unlabelled and are on long branches (the greatest value of Equation (6.3)). Apart from only unlabelled leaves, the worst case can also occur if there are one or two labelled leaves that share no label with any other node. In a similar manner, there are cases in which we have more than one leaf type and still the lower bound can be achieved. Nevertheless, we will discuss tighter bounds later and concentrate on examining different properties of the tree and their influence on the tree cover cost.

6.2 Properties of $T_\circ$ Influencing the Cost of Optimal Covers

In this section we analyse general tree properties, of which some were already discussed for a specific instance in Example 18. We now broaden the look from single subtrees treated individually to a collection of subtrees and how they relate to each other. While in some trees, two or more subtrees may share nodes (and edges), there are also trees in which this is not the case. There can even be a sequence of good and/or bad nodes, with or without labels, that lies between subtrees. We are especially interested in those nodes that are bad, as a tree cover is complete if and only if it covers all internal and external bad nodes of the tree. We thus define:

Definition 21 (Separating vertex): *A bad vertex that is not contained in any of the four subtrees $T_\circ^{\mathcal{A}}$, $T_\circ^{\mathcal{B}}$, $T_\circ^{\varepsilon}$ or $T_\circ^{\mathcal{AB}}$ is called a* separating vertex.

If no separating vertex exists in $T_\circ(A, B)$, the tree is *non-separated.* If we have no separating vertex, we can lower the upper bound, as we show in Section B of the appendix. Otherwise, the placement of separating vertices in the tree can influence the cover cost, as is specified below.

6.2.1 Separation Types

If separating vertices exist, obviously an optimal tree cover needs to contain paths that cover them. As we have already seen in Example 18, treating separating vertices may increase the cost in comparison to cases in which we can treat each subtree individually. Before finding paths that cover these vertices, we identify the placement of one or more separating vertices between single subtrees or pairs of subtrees.

Definition 22 (Individual (non-)separation): *Let $T_\circ(A, B)$ be a bad component group tree with more than one leaf type. Then an individual subtree $T_\circ^{\mathcal{X}}$ of type $\mathcal{X} \in \{\mathcal{A}, \mathcal{B}, \varepsilon, \mathcal{AB}\}$ is* separated *from the rest of $T_\circ$ if there is at least one separating vertex between $T_\circ^{\mathcal{X}}$ and the subtree induced by all other leaves. Otherwise, if there is at least one other individual or combined subtree and there is no separating vertex between this subtree and $T_\circ^{\mathcal{X}}$, we call $T_\circ^{\mathcal{X}}$* non-separated.

Examples are given in Figures 6.2 and 6.3 where subtrees are separated by a sequence of bad vertices, or, otherwise, the individual subtree is directly attached to another subtree or to a sequence of good nodes.

Definition 23 (Combined (non-)separation): *Let $T_\circ(A, B)$ be a bad component group tree with three or four leaf types. Then the combined subtree $T_\circ^{\{\mathcal{X},\mathcal{Y}\}}$ induced by all leaves of types $\mathcal{X}, \mathcal{Y} \subset \{\mathcal{A}, \mathcal{B}, \varepsilon, \mathcal{AB}\}$, $\mathcal{X} \cap \mathcal{Y} = \{\}$, is* separated*, if there is a separating vertex between $T_\circ^{\{\mathcal{X},\mathcal{Y}\}}$ and the subtree induced by the leaf types that are not in $\mathcal{X} \cup \mathcal{Y}$. Additionally, $\mathcal{X}$ and/or $\mathcal{Y}$ may or may not be individually separated.*

For example in Figure 6.2 (iv) the combined $\{\mathcal{A}, \varepsilon\}$-subtree is separated from the rest, but at the same time $T_\circ^{\mathcal{A}}$ is separated from $T_\circ^{\varepsilon}$. On the other hand in tree (ii) the combined $\{\mathcal{A}, \mathcal{AB}\}$-subtree is separated from $T_\circ^{\varepsilon}$, but $T_\circ^{\mathcal{A}}$ and $T_\circ^{\mathcal{AB}}$ are non-separated.

Depending on the number of subtrees present, an individual separation mutually defines the combined subtree of the other subtrees as equally separated or equally

(i) Two subtrees are separated (full separation). **(ii)** $T_\circ^{\mathcal{A}}$ and $T_\circ^{\mathcal{AB}}$ are non-separated, but $T_\circ^{\varepsilon}$ is separated from $T_\circ^{\{\mathcal{A},\mathcal{AB}\}}$. **(iii)** All three subtrees are separated (full separation). **(iv)** Four individual separations and $T_\circ^{\{\mathcal{A},\varepsilon\}}$ is separated from $T_\circ^{\{\mathcal{B},\mathcal{AB}\}}$.

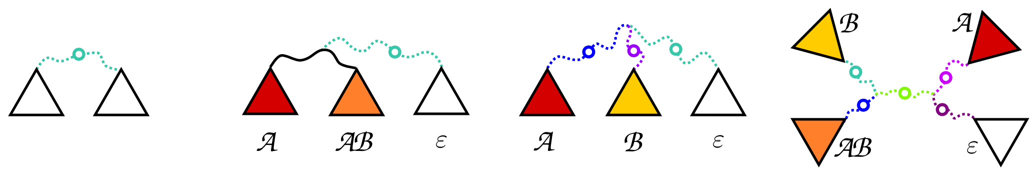


Figure 6.2: Schematic display of some examples for subtrees and their interrelation. The triangles are the subtrees and the circle denotes a separating vertex, the dotted paths can contain further separating vertices, whereas the black solid line represents a possible path that has no bad vertex.

non-separated. This becomes more apparent in Figure 6.3 where the two counter parts are coloured accordingly.

(i) Three individual non-separations. **(ii)** A combined subtree is non-separated. **(iii)** A combined subtree is separated. **(iv)** Two combined subtrees are separated.

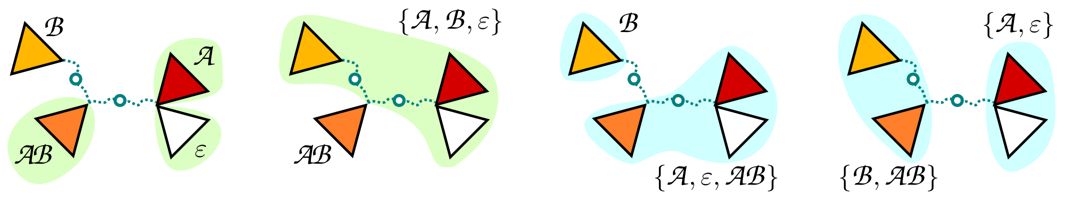

$\mathcal{B}$
$\mathcal{A}$
$\mathcal{AB}$
ε
$\{\mathcal{A},\mathcal{B},\varepsilon\}$
$\mathcal{AB}$
$\mathcal{B}$
$\{\mathcal{A},\varepsilon,\mathcal{AB}\}$
$\{\mathcal{A},\varepsilon\}$
$\{\mathcal{B},\mathcal{AB}\}$

Figure 6.3: One schematic tree with different single and combined non-separated subtrees highlighted in green and separated subtrees highlighted in blue. The triangles correspond to distinct subtrees, the circle denotes a separating vertex, the dotted paths can contain further separating vertices.

Definition 24 (Full separation): *Let $T_\circ(A, B)$ be a bad component group tree with two or more leaf types. Then we have a* full separation *if each subtree is separated.*

Full separations are depicted in Figure 6.2 for two subtrees by tree (i), for three leaf types by tree (iii) and for four leaf types by tree (iv).

In some cases using an extra path, covering solely the separating vertex or vertices, may be (co-)optimal. In other cases, however, it may be necessary to consider one, or even several, heterogeneous paths in order to achieve an optimal cover (see Subsection A.6 of the appendix for examples). For Figure 6.2 (iv) an $\mathcal{A}$-$\mathcal{AB}$-path and a $\mathcal{B}$-ε-path cover the separation while an $\mathcal{A}$-ε-path and a $\mathcal{B}$-$\mathcal{AB}$-path do not. Although an $\mathcal{A}$-$\mathcal{B}$-path and an ε-$\mathcal{AB}$-path would cover the same paths as the first suggestion, the cost would be higher.

We will see later that, more often than not, we are interested in knowing if a subtree is *not* separated from the rest of the tree, as otherwise the cost might be higher and we need to use a different set of covering paths.

6.2.2 Links of Subtrees

The presence of separating vertices can increase the cost w.r.t. the tree without them. However, we also observe that a separation might be compensated for, by making use of a specific internal labelling. As seen in Example 18, an internal label used with a labelled leaf can cover the sequence of separating vertices with lower cost compared to unlabelled paths (covers ii-b and ii-c). This motivates the following definitions, w.r.t. different types of separations.

Definition 25 ($\mathcal{A}$-ε-link): *Given a bad component group tree $T_\circ(A, B)$ with a non-empty ε-subtree $T_\circ^\varepsilon$ and a non-empty $\mathcal{A}$- or $\mathcal{AB}$-subtree, we say that there is an* $\mathcal{A}$-ε-link *in $T_\circ$ if there exists an $\mathcal{A}$-label in the separating vertex closest to $T_\circ^\varepsilon$ or in any vertex closer to or within $T_\circ^\varepsilon$.*

Definition 26 ($\mathcal{A}$-$\mathcal{B}$-link): *Given a bad component group tree $T_\circ(A, B)$ with non-empty $\mathcal{A}$- and $\mathcal{B}$-subtrees, we say that there is an* $\mathcal{A}$-$\mathcal{B}$-link *in $T_\circ$ if there exists an $\mathcal{A}$-label in the separating vertex closest to $T_\circ^\mathcal{B}$ or in any vertex closer to or within $T_\circ^\mathcal{B}$.*

$\mathcal{A}$-links in the $\mathcal{AB}$-subtree are neglected as we can simply use (or re-use) the label of an $\mathcal{AB}$-leaf directly, while still covering the same separating vertices. Analogously $\mathcal{B}$-links in the $\mathcal{AB}$-subtree are neglected.

Definition 27 ($\mathcal{B}$-ε-link): *Given a bad component group tree $T_\circ(A, B)$ with a non-empty ε-subtree $T_\circ^\varepsilon$ and a non-empty $\mathcal{B}$- or $\mathcal{AB}$-subtree, we say there is a* $\mathcal{B}$-ε-link *in $T_\circ$ if there exists a $\mathcal{B}$-label in the separating vertex closest to $T_\circ^\varepsilon$ or in any vertex closer to or within $T_\circ^\varepsilon$.*

Definition 28 ($\mathcal{B}$-$\mathcal{A}$-link): *Given a bad component group tree $T_\circ(A, B)$ with non-empty $\mathcal{B}$- and $\mathcal{A}$-subtrees, we say that there is a* $\mathcal{B}$-$\mathcal{A}$-link *in $T_\circ$ if there exists a $\mathcal{B}$-label in the separating vertex closest to $T_\circ^\mathcal{A}$ or in any vertex closer to or within $T_\circ^\mathcal{A}$.*

As $\mathcal{A}$- and $\mathcal{B}$-labels of an $\mathcal{AB}$-node are used individually (see Section 5.5) there cannot be an $\mathcal{AB}$-link. Therefore, an $\mathcal{AB}$-labelled internal vertex in the ε-subtree, for example, can be accepted as either $\mathcal{A}$-ε- or $\mathcal{B}$-ε-link but never more (the definitions of links use internal labels and not the labelling). We are, however, free to use the same vertex once as an $\mathcal{A}$-link and once as a $\mathcal{B}$-link, if necessary. Obviously, ε-links do not exist as they never help diminishing the cost and instead we could use a leaf in the subtree considered.

Definition 29 ($\mathcal{A}$-$\{\mathcal{B},\varepsilon\}$-link): *Given a bad component group tree $T_\circ(A,B)$ with non-empty $\mathcal{A}$-, $\mathcal{B}$- and ε-subtrees, we say there is an $\mathcal{A}$-$\{\mathcal{B},\varepsilon\}$-*link *in $T_\circ$ if there exists an $\mathcal{A}$-label in the separating vertex closest to the combined $\{\mathcal{B},\varepsilon\}$-subtree induced by all $\mathcal{B}$- and ε-leaves or in any vertex closer to or within the combined $\{\mathcal{B},\varepsilon\}$-subtree.*

Definition 30 ($\mathcal{B}$-$\{\mathcal{A},\varepsilon\}$-link): *Given a bad component group tree $T_\circ(A,B)$ with non-empty $\mathcal{A}$-, $\mathcal{B}$- and ε-subtrees, we say there is a $\mathcal{B}$-$\{\mathcal{A},\varepsilon\}$-*link *in $T_\circ$ if there exists a $\mathcal{B}$-label in the separating vertex closest to the combined $\{\mathcal{A},\varepsilon\}$-subtree induced by all $\mathcal{A}$- and ε-leaves or in any vertex closer to or within the combined $\{\mathcal{A},\varepsilon\}$-subtree.*

For the same reasons as above, we would not use an $\mathcal{AB}$-leaf with such a link. Also we do not consider links in the combined subtrees of two labelled leaf types (e.g. $\mathcal{A}$-$\{\varepsilon,\mathcal{AB}\}$-links or $\mathcal{A}$-$\{\mathcal{AB},\mathcal{B}\}$-links). There can be $\mathcal{A}$- or $\mathcal{B}$-labels outside of the specified vertices and subtree, but they do not necessarily produce a link. Figure 6.4 shows some examples of where a $\mathcal{B}$-labelled vertex produces a link given bad component group trees with three leaf types but different separations.

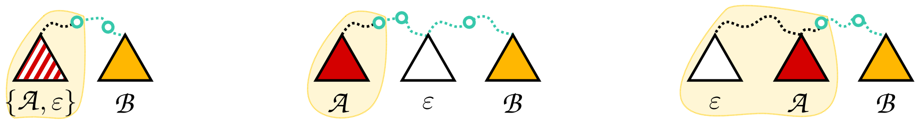


Figure 6.4: Three examples of trees with separating vertices (indicated by the coloured dotted line and the circles). In order for a $\mathcal{B}$-label to function as a link, it needs to be in a node within the shaded part of the respective exemplary tree. If this is the case we have a $\mathcal{B}$-$\{\mathcal{A},\varepsilon\}$-link in the left and right tree and a $\mathcal{B}$-$\mathcal{A}$-link in the central tree.

Links are also the reason why we have to consider all types of labels/labellings, even when they are not represented in the leaf composition. If no $\mathcal{B}$-leaves are present one might think that we simply treat the $\mathcal{AB}$-leaves as $\mathcal{A}$-leaves. However, that this should not be done is shown in the following (counter-)example.

Example 19: Figure 6.5 shows two trees of the same structure that have leaf compositions $\mathcal{L} = (3,0,2,0)$ and $\mathcal{L}' = (1,0,2,2)$. Both trees have an internal label marked by $\mathcal{B}^*$ serving as a $\mathcal{B}$-ε link but neither tree has a $\mathcal{B}$-leaf. First, we consider the left tree for which the best we can achieve is cost 5. One optimal cover uses a homogeneous $\mathcal{A}$-path, a homogeneous ε-path and connects the third $\mathcal{A}$-leaf with $\mathcal{B}^*$ in order to cover the separation. For the tree on the right we can find an optimal cover that has cost 4 and is composed as follows: a homogeneous ε-path, an $\mathcal{A}$-$\mathcal{AB}$-path and a path that uses the $\mathcal{B}$-label of the $\mathcal{AB}$-leaf with B^*. $\diamond$

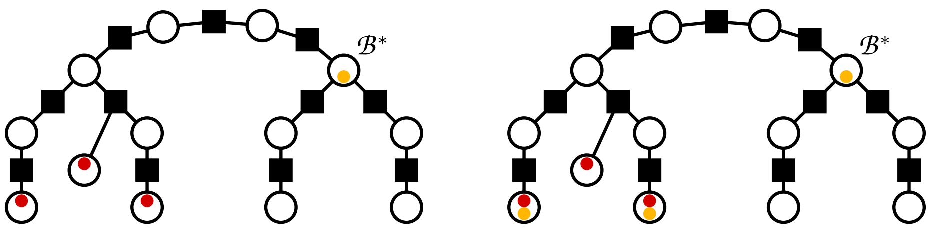


Figure 6.5: Two trees without $\mathcal{B}$-leaves. Left: The tree has $\mathcal{L} = (3, 0, 2, 0)$ and a $\mathcal{B}$-ε-link that cannot be used. The optimal cost is 5. Right: The tree has $\mathcal{L} = (1, 0, 2, 2)$ and the $\mathcal{B}$-ε-link is used with an $\mathcal{AB}$-leaf in order to obtain an optimal cover (cost 4).

Thus, even if the other tree properties were the same, the cost of an optimal cover of a tree with leaf composition $\mathcal{L} = (n_{\mathcal{A}}, 0, n_{\varepsilon}, n_{\mathcal{AB}})$ is not always the same as the cost of an optimal cover of a tree with leaf composition $\mathcal{L}' = (n_{\mathcal{A}} + n_{\mathcal{AB}}, 0, n_{\varepsilon}, 0)$. In consequence, we cannot simply affiliate all $\mathcal{AB}$-leaves to $\mathcal{A}$-leaves if $\mathcal{B}$-leaves are absent.

6.2.3 Short Paths

In the inversion model a leaf can be covered by a short path if its elimination from the tree and updating the tree to a bad component group tree again does not produce a new leaf. Historically, the corresponding branch of this leaf is referred to as *short branch*. In some instances not all short branches can be covered optimally by short paths, for example, when the covering (removal) of one short branch causes another short branch to become a long branch. In this case, the latter can no longer be safely covered by a short path (an elaboration for unlabelled trees can be found in Subsection A.4). Fortunately, the inversion distance offset, as given in Equation (6.3), asks only to find at most one of such short branches.

When allowing several types of subtrees, the definition of short branch for ε-leaves requires extensive study of the structure and the properties of the tree.

Example 20 (Short ε-branch): The bad component group tree $T_\circ(A, B)$ as shown in Figure 6.6 (i) has leaf composition $\mathcal{L} = (2, 0, 3, 0)$. The branch of ℓ_1 has no further bad vertex, hence fulfilling the definition of "short branch". However, the insufficiency of this definition becomes apparent when the homogeneous $\mathcal{A}$-path is visually covered: Leaf ℓ_1 has i_1 and i_2 as bad vertices, before it reaches another branching node in the ε-subtree (the tree is a *fortress*). Covering ℓ_1 by a short path thus either causes $T_\circ^{\mathcal{A}}$ and $T_\circ^{\varepsilon}$ to become separated or, if $T_\circ^{\mathcal{A}}$ is already removed, produces a new bad leaf (i_1). An optimal cover of this tree has cost 5. ◇

It is thus necessary to refine the definition of "short" in presence of labelled subtrees.

(i) Initial tree with non-separated $\mathcal{A}$- and ε-subtrees has leaf ℓ_1 on a short branch.

(ii) Covering the $\mathcal{A}$-subtree reveals that an optimal cover has cost 5.

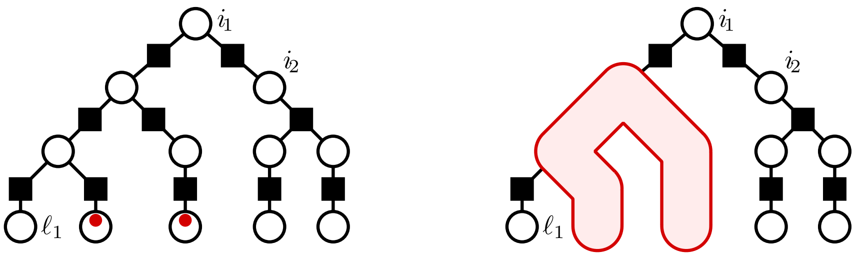


Figure 6.6: Short ε-branches are not always safe. Covering ℓ_1 by a short ε-path lets i_1 and i_2 become separating vertices.

Definition 31 (Solo ε-leaf): *An unlabelled leaf ℓ in $T_\circ(A, B)$ is a* solo ε-leaf *if the elimination of ℓ from $T_\circ$, by turning the component group that ℓ represents into a good one, and transforming the tree into a bad component group tree again, neither transforms a bad node that was previously internal into a leaf nor produces a new separation of subtrees (nodes of the ε-subtree to become separating vertices). Otherwise this vertex is referred to as being on a* long branch*.*

We have already seen that the tree depicted in Figure 6.6 has no solo ε-leaf. Now we give an example of a tree that does.

Example 21 (Solo ε-leaf): Figure 6.7 depicts a similar tree as given in Example 20, except its nodes i_1 and i_2 are good. The two subtrees are still overlapping. Here,

(i) Initial tree.

(ii) Optimal cover yields cost 4.

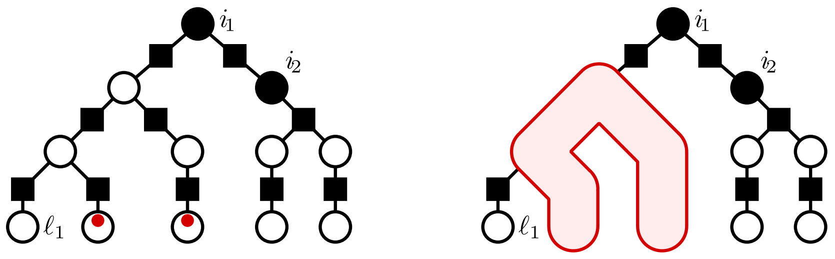


Figure 6.7: The same tree as in Figure 6.6, except i_1 and i_2 are good nodes. Covering the $\mathcal{A}$-subtree reveals that covering ℓ_1 by a short ε-path neither creates a new separation nor a new leaf.

covering ℓ_1 with a short path does not separate the two subtrees and also does not create a new bad leaf. Hence, we can achieve an optimal cover with cost 4. $\diamond$

A solo ε-leaf could, if necessary, be covered by a short path (it would be safe although not always optimal). Note that if $n_\varepsilon = 1$, the ε-subtree consists of a single vertex, and any bad vertices of the “long branch” correspond to a sequence of separating vertices. By the cost scheme, it is always optimal to replace two short ε-paths of a cover by one long ε-path. It is therefore merely interesting if one such solo ε-leaf does or does not exist instead of searching for them all. As stated earlier, it is irrelevant to check for “short” or “solo” labelled leaves, they are simply used with another leaf that shares a label.

6.2.4 Property Set of $T_\circ$

Now that we elaborated the influencing properties, we briefly discuss further tree properties that do not influence the cost in a positive way. For instance, we would like to remind the reader of Observation 6 (see page 100) which states that forming new $\mathcal{AB}$-labelled vertices that are then (re)-used with an $\mathcal{AB}$-cycle is not to be the method of choice. Furthermore, the branch length of labelled leaves is neglected. Besides ruling this out, we can also dismiss some other properties.

Observation 7. An internal label positioned in such a way that, if used, it will render an ε-leaf into a solo ε-leaf, can be neglected. This becomes exceedingly obvious in a situation where $r_\varepsilon = 1$ and we have a separation. The cost of a short ε-path plus the cost of using a link (e.g. from an $\mathcal{A}$-leaf to an $\mathcal{A}$-ε-link) amounts to 2, while using the labelled leaf for a heterogeneous path with the ε-leaf directly, also produces cost 2 and covers the same vertices. Also, the latter does not need an internal label and is thus always possible.

For obvious reasons, internal labels in a subtree whose leaves have the same label are neglected, e.g. an $\mathcal{A}$-label in the $\mathcal{A}$- or $\mathcal{AB}$-subtree. However, what happens if there are two links that span a separation and they are not being used for homogeneous paths with a leaf?

Observation 8. Labelled internal vertices that fulfil the properties of links for either side of the separation, such that they could be used to cover solely the separation at hand, can be neglected for such a use, as using directly the leaves to span a separation is either co-optimal or optimal.

Observation 9. The set of tree properties of the bad labelled component group tree $T_\circ(A, B)$ that is necessary (and also sufficient) to consider, in order to find an optimal complete tree cover, consists of:

Leaf composition $\mathcal{L}(A, B) = (n_\mathcal{A}, n_\mathcal{B}, n_\varepsilon, n_\mathcal{AB})$ of the bad labelled component group tree $T_\circ(A, B)$. (Bear in mind the pushing of labels of good leaves to interior vertices when transforming T into $T_\circ$.)

Separations of subtrees when more than one leaf type is present, where, besides the arrangement of subtrees, also the number of separating vertices can affect the cost of an optimal cover.

Links that are specifically located labels which may compensate for a separation ($\mathcal{A}$-labels in the $\mathcal{B}$-, ε- or $\{\mathcal{B}, \varepsilon\}$-subtrees, and $\mathcal{B}$-labels in the $\mathcal{A}$-, ε- or $\{\mathcal{A}, \varepsilon\}$-subtrees).

Solo $\boldsymbol{\varepsilon}$-leaf that can be covered by a short ε-path and may reduce the cover cost compared to the same leaf being on a long branch. If this leaf is the only ε-leaf existing it corresponds to a non-separated ε-subtree.

An optimal cover may depend on one of these properties, different properties at a time, combinations or, for example, two links being present at the same time. The careful analysis conducted in the remainder of this chapter shows that this set of properties is sufficient.

6.3 Heterogeneous Paths

Between all the different properties that we have to consider in order to find an optimal complete cover, we seek to maximise the number of homogeneous paths used, such that we need only to determine which paths compose $\mathtt{C}_r$ (from Equation (6.2)). In the following, we determine which and how many homogeneous paths are safe, depending only on the leaf composition.

Apart from (short or long) homogeneous paths connecting two leaves, there are heterogeneous paths either from leaves to links or to another leaf. For example, if there were a cover that contained a short $\mathcal{AB}$-path as well as a short $\mathcal{A}$-path, the cover could not be optimal, as replacing these two short paths by a single heterogeneous path results in lower cost. While there are cases in which using heterogeneous paths for covering separations is not necessary, or even counter-optimal, other cases require the use of such paths in order to achieve an optimal cover. Figure 6.8 shows covers using homogeneous paths for the subtrees. In each case the cost is compared to a cover that would use two heterogeneous paths instead. If for trees (ii) and (iii) both separating nodes shared a label, the path covering the separation would have cost 1, producing lower cost for tree (ii) and the same cost for tree (iii) as a heterogeneous cover in the respective cases. As can be seen, each tree has to be analysed carefully and no universal handling will work.

(i) The shown homogeneous cover has cost 3, whereas a heterogeneous cover has higher cost.

(ii) The shown homogeneous cover has cost 4, the same as a heterogeneous cover.

(iii) The shown homogeneous cover has cost 5, but a heterogeneous cover has lower cost.

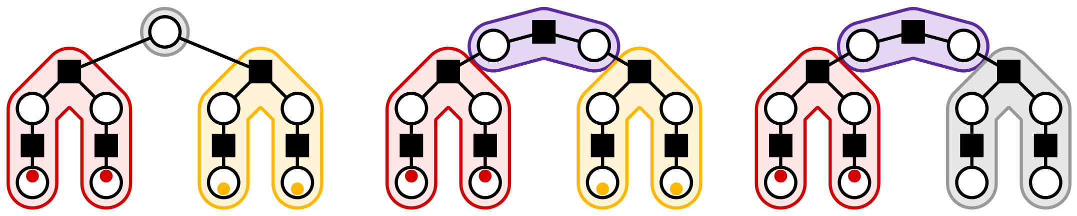


Figure 6.8: Three trees that show that each instance has to be analysed individually and no general rule to the handling of how to cover separating vertices exist.

We already provided some bounds with specific necessary preconditions (for example given in Section B of the appendix). Also, as stated earlier, we never have more than one short path in each subtree, which also holds across certain labelled subtrees.

Let us now observe the implications of separating vertices. Although there are a variety of separations and thus also a variety of heterogeneous paths possible, we observe that there is an upper bound to using heterogeneous paths of one kind, as the following lemma shows.

Lemma 4: *Let* $T_\circ^{\mathcal{X}}$ *and* $T_\circ^{\mathcal{Y}}$ *be two subtrees of* $T_\circ(A, B)$ *that have leaf types* $\mathcal{X}, \mathcal{Y} \in \{\mathcal{A}, \mathcal{B}, \mathcal{AB}, \varepsilon\}$*, where* $\mathcal{X} \neq \mathcal{Y}$*. Then there always exists an optimal cover that uses at most two paths connecting an* $\mathcal{X}$*-leaf with a* $\mathcal{Y}$*-leaf.*

Proof. For contradiction let us assume that using three (respectively four) heterogeneous paths, each connecting a leaf of the $\mathcal{X}$-subtree with a leaf of the $\mathcal{Y}$-subtree is always optimal.

All of the heterogeneous paths cover the same separating vertices (if there are any). We replace two of these heterogeneous paths by one homogeneous path in the $\mathcal{X}$- and by another in the $\mathcal{Y}$-subtree. The separating vertices are still covered by the remaining heterogeneous path(s). (In order to cover all internal vertices, we choose the leaves that will form the homogeneous paths in such a way, that they have at least one vertex in common with the heterogeneous path.) If the two subtrees share a label ($T_\circ^{\mathcal{AB}}$ and either $T_\circ^{\mathcal{A}}$ or $T_\circ^{\mathcal{B}}$), the cost of two heterogeneous paths is 2, and the replacement cost is also 2. Otherwise, the cost of the two heterogeneous paths is 4 and the replacement cost is either 3 or 2, depending on the number of labelled subtrees involved. This is a contradiction to the assumption that using the same type of heterogeneous paths three (resp. four) times is always optimal (it is either co-optimal or suboptimal), we can thus always do the described replacement. □

We conclude that we can always find an optimal cover using at most two of the same type of heterogeneous paths connecting leaves and that all other leaves are used for short or long homogeneous paths including links. In total this yields a maximum of $\binom{\#\text{types}}{2}$ heterogeneous paths. In the worst case this means, for a tree with all four leaf types present, we could have a maximum of twelve heterogeneous paths thus cost 20. On the other side, for trees with a large number of leaves we can use a large number of homogeneous paths which for each labelled subtree decrease the upper bound given by Inequality (6.5).

We discover that, apart from the findings of Lemma 4, there are tighter upper bounds to the total number of heterogeneous paths ending in the same subtree as we elaborate on the following pages.

$\mathcal{A}$-Leaves used for Heterogeneous Paths

For now, let us concentrate on the use of heterogeneous paths ending in one specific subtree. In the worst case we could have six heterogeneous paths connecting $\mathcal{A}$-leaves with the leaves of other types. This means all other $\mathcal{A}$-leaves are used for long or short homogeneous paths. However, the following proposition shows that we can always replace a cover using four $\mathcal{A}$-labelled leaves for heterogeneous paths by a cover that uses only two $\mathcal{A}$-labelled leaves.

Proposition 10: *Any cover that uses four $\mathcal{A}$-leaves for heterogeneous paths can be replaced by a cover with lower or equal cost using only two $\mathcal{A}$-leaves for heterogeneous paths.*

Proof. Lemma 4 shows that in order to have four heterogeneous paths (using $\mathcal{A}$-leaves) in the beginning, there have to be at least an additional two subtrees other than the $\mathcal{A}$-subtree. Assume we have three, resp. four types of leaves in $T_\circ$, such that we have the following combination of paths:

(1) two $\mathcal{A}$-$\mathcal{B}$-paths and two $\mathcal{A}$-ε-paths, or
(2) two $\mathcal{A}$-$\mathcal{B}$-paths and two $\mathcal{A}$-$\mathcal{AB}$-paths, or
(3) two $\mathcal{A}$-ε-paths and two $\mathcal{A}$-$\mathcal{AB}$-paths, or
(4) two $\mathcal{A}$-$\mathcal{B}$-paths, one $\mathcal{A}$-ε-path and one $\mathcal{A}$-$\mathcal{AB}$-path, or
(5) one $\mathcal{A}$-$\mathcal{B}$-path, two $\mathcal{A}$-ε-paths and one $\mathcal{A}$-$\mathcal{AB}$-path, or
(6) one $\mathcal{A}$-$\mathcal{B}$-path, one $\mathcal{A}$-ε-path and two $\mathcal{A}$-$\mathcal{AB}$-paths.

When we have three types of subtrees we consequently have two pairs of heterogeneous paths in the possible combinations of cases (1-3) shown in Figure 6.9 (left). One duplicate of each is removed and we add a long homogeneous $\mathcal{A}$-path and one new heterogeneous path connecting the two other subtrees with each other. The first two cases reduce the cost by 1 and the third case does not change the cost. In each case, the internal vertices are still covered. Hence, the reduction of the $\mathcal{A}$-subtree can be done from four to two heterogeneous $\mathcal{A}$-paths if three subtrees are involved.

Now, let us assume we have four leaf types. Among the possible heterogeneous types ($\mathcal{A}$-$\mathcal{B}$-, $\mathcal{A}$-ε- and $\mathcal{A}$-$\mathcal{AB}$-paths) there must be one that occurs twice (cases (4), (5) and (6) from above whose distribution of paths is displayed in Figure 6.9 (right)). For each, we remove an $\mathcal{A}$-$\mathcal{B}$-path and add a long homogeneous $\mathcal{A}$-path. For (4) and (5) an $\mathcal{A}$-ε-path is removed and a $\mathcal{B}$-ε-path is added. For (6) we remove a duplicate $\mathcal{A}$-$\mathcal{AB}$-path and connect the $\mathcal{AB}$-leaf to a $\mathcal{B}$-leaf. In each of the three cases we diminish the cost by 1 while still covering the internal vertices.

The distribution of paths is schematically displayed in Figure 6.9 for each of the six cases. In conclusion, instead of using four leaves for heterogeneous paths, we can

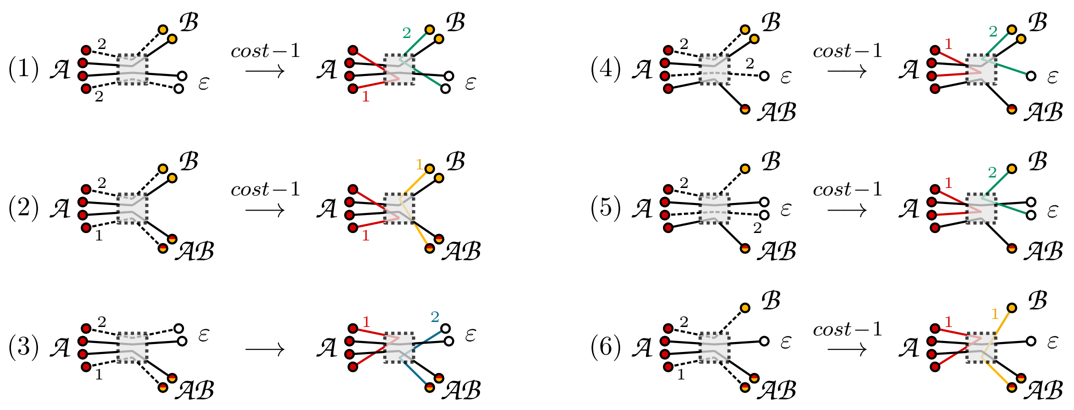


Figure 6.9: Replacements for heterogeneous $\mathcal{A}$-paths such that two instead of four $\mathcal{A}$-leaves are used while yielding lower or equal costs. Solid paths are unchanged. Dashed paths are replaced by paths indicated in different colours. Old and new costs are marked next to the respective path. The shaded box represents the (unknown) internal layout of the tree, including separating vertices.

always find a better or equal cover that uses only two $\mathcal{A}$-leaves for heterogeneous paths and that still covers the internal vertices. □

Proposition 11: *Any cover that uses three $\mathcal{A}$-leaves for heterogeneous paths can be replaced by a cover with lower or equal cost using only one $\mathcal{A}$-leaf for heterogeneous paths.*

Proof. With only three heterogeneous paths that end in $\mathcal{A}$-leaves, we can either have two subtrees other than the $\mathcal{A}$-leaves present which means one type of heterogeneous paths occurs twice (cases (1)-(6)), or there are all four subtrees present, thus we have three distinct types of heterogeneous paths (case (7)). (Lemma 4 shows we do not need to consider three times the same type.)

For cases (1-6), we remove one heterogeneous path of each type, such that the duplicate type remains. Then we add a homogeneous $\mathcal{A}$-path and additionally a heterogeneous path connecting the two other subtrees. For case (7), we can replace the $\mathcal{A}$-$\mathcal{B}$-path and an $\mathcal{A}$-ε-path by a homogeneous $\mathcal{A}$-path and connect the $\mathcal{B}$-subtree with both ε- and $\mathcal{AB}$-subtree. The cost is equal, but only two heterogeneous paths with $\mathcal{A}$-leaves are used. In case there were only two $\mathcal{A}$-leaves before, one of them was re-used before the replacement and we could thus omit the homogeneous $\mathcal{A}$-path, reducing the overall cost.

Figure 6.10 shows schematically the cases and path replacements. In all seven cases

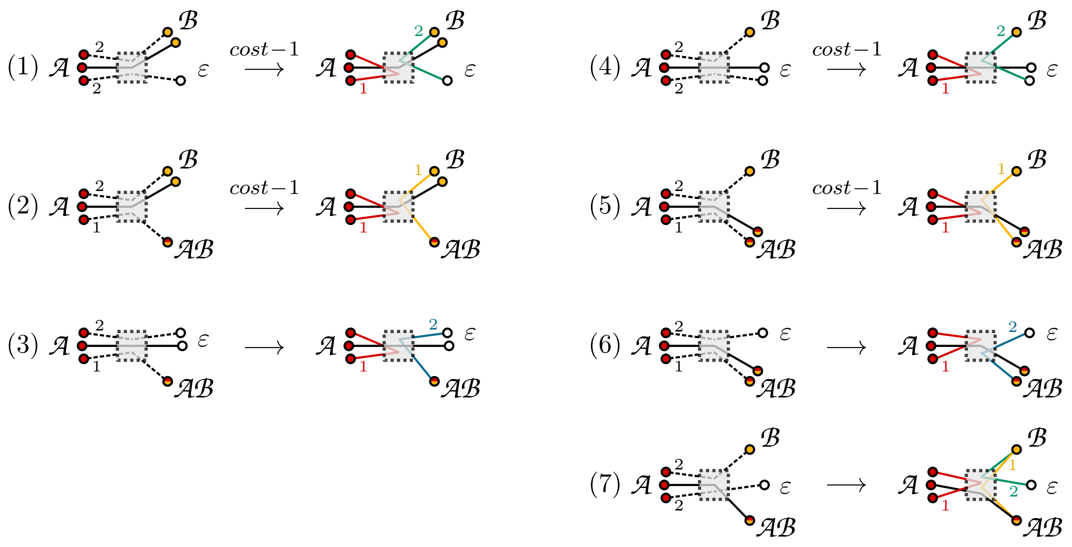


Figure 6.10: Replacements for heterogeneous $\mathcal{A}$-paths that yield lower costs and use one instead of three $\mathcal{A}$-leaves. Legend can be found in caption of Figure 6.9.

we can achieve lower or equal cost and use only one $\mathcal{A}$-leaf for a heterogeneous path while still covering the internal vertices. □

Any tree cover that uses more than four leaves for heterogeneous paths can be reduced to a cover with only two heterogeneous $\mathcal{A}$-paths by finding a set of paths in the cover that satisfy the above proposition and replacing it by those with lower cost.

The above results do not imply that all other $\mathcal{A}$-leaves are used for long homogeneous $\mathcal{A}$-paths only. Bear in mind that there might be single leaves left (for example to be covered by a short path), and also that labelled leaves might be used for paths with a link. Unsafe paths also create new leaves that are unaccounted for, which also has to be considered. Furthermore, just because there is a cover with lower cost, it does not mean it is the overall lowest possible cost, as this does not take into account the same way of replacing heterogeneous paths in the other subtrees.

$\mathcal{B}$-Leaves used for Heterogeneous Paths

Proposition 12: *Any cover that uses three or four $\mathcal{B}$-leaves for heterogeneous paths can be replaced by a cover with lower or equal cost by using one homogeneous $\mathcal{B}$-path thus reducing the number of $\mathcal{B}$-leaves used for heterogeneous paths by 2.*

Proof. As the cost scheme for $\mathcal{A}$- and $\mathcal{B}$-labels is symmetric, obviously, we can always find a cover with lower or equal cost by using one instead of three heterogeneous $\mathcal{B}$-paths and we can always replace a cover using four heterogeneous $\mathcal{B}$-paths with a cover that has lower or equal cost using only two heterogeneous $\mathcal{B}$-paths while still covering the internal vertices (analogous to Propositions 10 and 11). □

ε-Leaves used for Heterogeneous Paths

We continue showing that the same is true for the ε-subtree.

Proposition 13: *Any cover that uses four ε-leaves for heterogeneous paths can be replaced by a cover with lower or equal cost using only two ε-leaves for heterogeneous paths.*

Proof. Lemma 4 shows that in order to have four heterogeneous paths (ending in ε-leaves) in the beginning, there have to be at least two other subtrees. Assume we have three, resp. four leaf types, such that we have the following combination of paths:

(1) two ε-$\mathcal{A}$-paths and two ε-$\mathcal{B}$-paths, or
(2) two ε-$\mathcal{A}$-paths and two ε-$\mathcal{AB}$-paths, or
(3) two ε-$\mathcal{A}$-paths, one ε-$\mathcal{B}$-path and one ε-$\mathcal{AB}$-path, or
(4) one ε-$\mathcal{A}$-path, one ε-$\mathcal{B}$-path and two ε-$\mathcal{AB}$-paths.

The distribution of paths is displayed in Figure 6.11. Further cases can be achieved by swapping $\mathcal{A}$- and $\mathcal{B}$ and are not looked into due to symmetry of costs.

Let us consider trees with only two other subtrees first. We remove one of each duplicated heterogeneous path. In case (1) we add a homogeneous ε-path and an $\mathcal{A}$-$\mathcal{B}$-path which yields the same overall cost. In case (2) we add a homogeneous ε-path and an $\mathcal{A}$-$\mathcal{AB}$-path reducing the cost by 1. In each case, the internal vertices are still covered. Hence, the reduction of the ε-subtree can be done from four to two heterogeneous paths if three subtrees are involved.

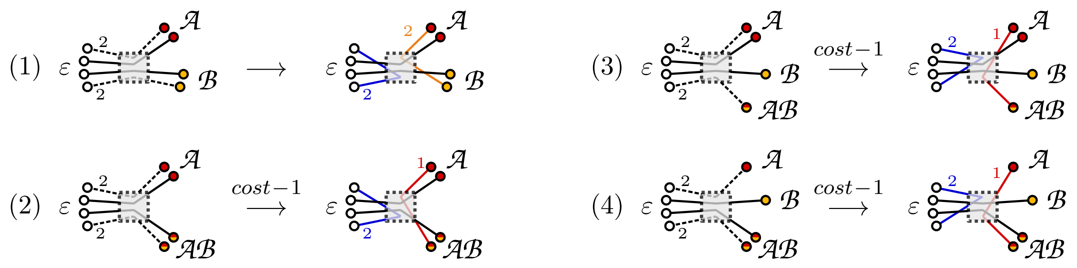


Figure 6.11: Replacements for heterogeneous ε-paths from four to two employed ε-leaves yielding lower or equal costs. Legend can be found in caption of Figure 6.9.

Now, let us assume we have four leaf types, cases (3) and (4) are depicted in Figure 6.11 on the right. For each, we remove one ε-$\mathcal{AB}$- and one ε-$\mathcal{A}$-path and add a long homogeneous ε-path and one $\mathcal{A}$-$\mathcal{AB}$-path, diminishing the overall cost by 1 in each case and still covering the internal vertices.

Thus we can always find a better or equal cover that uses only two ε-leaves for heterogeneous paths and that still covers the internal vertices. □

Proposition 14: *Any cover that uses three ε-leaves for heterogeneous paths can be replaced by a cover with lower or equal cost using only one ε-leaf for heterogeneous paths.*

Proof. With only three heterogeneous paths that end in ε-leaves, we can either have two subtrees other than the ε-subtree present which means one type of heterogeneous paths occurs twice (cases (1)-(4)), or there are four subtrees present, thus we have three distinct types of heterogeneous paths (case (5)). (Lemma 4 shows it cannot be three times the same type.)

For case (5), we can replace the ε-$\mathcal{A}$-path and ε-$\mathcal{B}$-path by an $\mathcal{A}$-$\mathcal{AB}$-, a $\mathcal{B}$-$\mathcal{AB}$- and a homogeneous ε-path. The cost is equal, but only two heterogeneous paths with ε-leaves are used while either we break up a homogeneous $\mathcal{AB}$-paths or re-use the single $\mathcal{AB}$-leaf twice. In case there were only two ε-leaves, one of them was re-used before the replacement. Here, we only break the $\mathcal{B}$-ε-path and connect the $\mathcal{B}$-leaf to the $\mathcal{AB}$-leaf. A homogeneous ε-path is not necessary but we still reduce the overall cost.

Otherwise, due to symmetry of cost for $\mathcal{A}$- and $\mathcal{B}$-nodes, we have two other subtrees present (in cases (1) and (3) that is $\mathcal{A}$ and $\mathcal{B}$, in cases (2) and (4) that is $\mathcal{A}$ and $\mathcal{AB}$), one of which has two heterogeneous paths with the ε-subtree. We remove one heterogeneous path of each type, such that the duplicate type remains. Then we add a homogeneous ε-path and additionally a heterogeneous path connecting the two other subtrees. While in cases (1) and (3) we achieve equal cost, for cases (2) and (4) we diminish the cost by 1.

Figure 6.12 shows schematically the five cases. In all five cases we can achieve lower

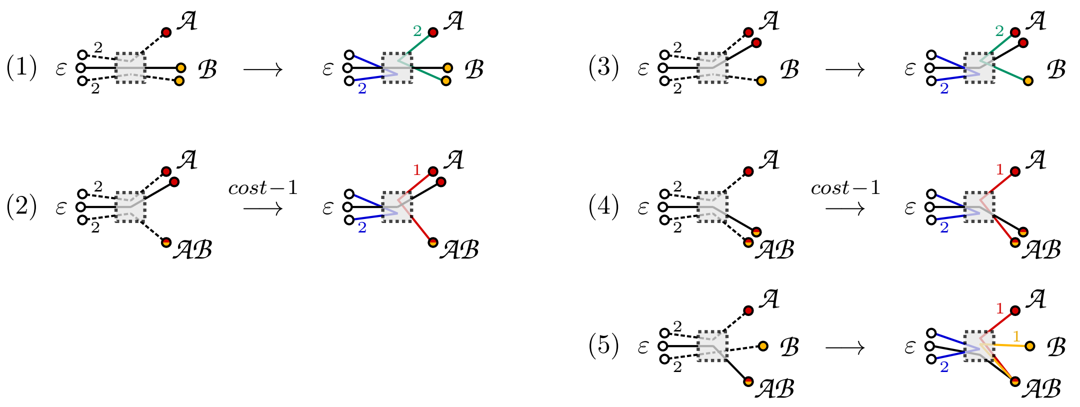


Figure 6.12: Replacements for heterogeneous ε-paths that yield lower costs when one instead of three ε-leaves are employed. Legend can be found in caption of Figure 6.9.

or equal cost and have only one ε-leaf used for a heterogeneous path while still covering the internal vertices. □

$\mathcal{AB}$-Leaves used for Heterogeneous Paths

Contrary to the other types of subtrees, for the $\mathcal{AB}$-subtree we need to consider more than one (resp. two) leaves for heterogeneous paths, as the following example shows.

Example 22 ($\mathcal{AB}$-Reduction): Figure 6.13 shows two trees that have the same separation and $\mathcal{A}$- and $\mathcal{B}$-subtrees, merely the $\mathcal{AB}$-subtree has two leaves in the left tree and four leaves in the tree depicted on the right. There are several co-optimal covers for the left tree that have cost 4. Obviously, using a homogeneous $\mathcal{AB}$-path in the tree

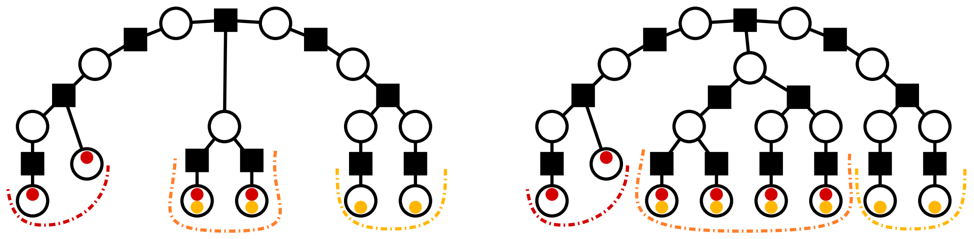


Figure 6.13: Left: $\mathcal{L}(A, B) = (2, 2, 0, 2)$. An optimal cover has cost 4 and can be achieved by using an $\mathcal{A}$-$\mathcal{AB}$-path, a $\mathcal{AB}$-$\mathcal{B}$-path and an $\mathcal{B}$-$\mathcal{A}$-path. Right: $\mathcal{L}(A, B) = (2, 2, 0, 4)$. An optimal cover has cost 4 and can be achieved by using two $\mathcal{A}$-$\mathcal{AB}$-paths and two $\mathcal{B}$-$\mathcal{AB}$-paths.

on the right-hand side has cost 1 and yields the tree on the left that has cover cost 4, resulting in overall cost of 5. An alternative and optimal cover uses two $\mathcal{A}$-$\mathcal{AB}$-paths and two $\mathcal{B}$-$\mathcal{AB}$-paths, achieving cost 4 in total. Another example with four types of subtrees is depicted in Subsection A.5 of the appendix. $\diamond$

Thus reducing the use of heterogeneous $\mathcal{AB}$-paths to two paths is not optimal.

Proposition 15: *Any cover that uses six $\mathcal{AB}$-leaves for heterogeneous paths can be replaced by a cover with lower or equal cost using only four $\mathcal{AB}$-leaves for heterogeneous paths.*

Proof. Having six heterogeneous paths ending in $\mathcal{AB}$-leaves means two of them are used for each of the $\mathcal{A}$-, $\mathcal{B}$- and ε-subtree (overall cost 8). Replacing one ε-$\mathcal{AB}$-path (cost 2) and one $\mathcal{B}$-$\mathcal{AB}$-path (cost 1) by one $\mathcal{B}$-ε-path (cost 2) and a homogeneous $\mathcal{AB}$-path (cost 1) yields the same cost while using only four $\mathcal{AB}$-leaves for heterogeneous paths and simultaneously not leaving a separating path uncovered. $\square$

Proposition 16: *Any cover that uses five $\mathcal{AB}$-leaves for heterogeneous paths can be replaced by a cover with lower or equal cost using only three $\mathcal{AB}$-leaves for heterogeneous paths.*

Proof. As Lemma 4 shows, each type of heterogeneous path occurs at most twice. With five heterogeneous paths from $\mathcal{AB}$-leaves all four subtree types must be present. While two types of heterogeneous paths occur twice, one of the other subtrees has only a single heterogeneous path ending in the $\mathcal{AB}$-subtree. Let this be (1) the ε-subtree or (2) the $\mathcal{B}$-subtree. As Figure 6.14 indicates, it is always possible to replace one

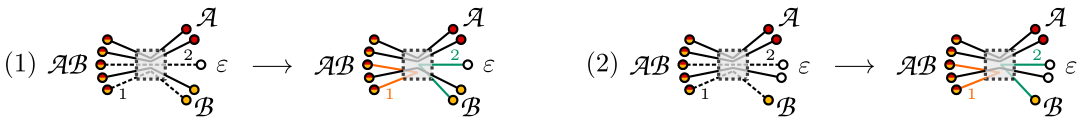


Figure 6.14: Replacements for heterogeneous $\mathcal{AB}$-paths yielding lower costs while employing three instead of five $\mathcal{AB}$-leaves. Legend can be found in caption of Figure 6.9.

$\mathcal{AB}$-$\mathcal{B}$-path and one $\mathcal{AB}$-ε-path by one $\mathcal{B}$-ε-path and a homogeneous $\mathcal{AB}$-path at the same cost while still covering the internal vertices. $\square$

We thus have reduced the number of $\mathcal{AB}$-leaves used for heterogeneous paths to four, resp. three leaves.

Looking back at the use of homogeneous paths and the remaining cover C_r (see Equation 6.2 on page 116) we determine that C_r has at most two heterogeneous paths of each type and does not have more than two heterogeneous paths that use $\mathcal{A}$-leaves (resp. $\mathcal{B}$, ε). Also the number of paths of C_r that use $\mathcal{AB}$-leaves for heterogeneous paths is at most 4. However, there may be short paths remaining in C_r and also paths connecting one leaf with a link or a single internal vertex is covered.

6.4 The Residual Tree T_r

In this section we use the above findings to determine the numbers of safe optimal homogeneous paths of each leaf type. As we determined upper bounds for the use of heterogeneous paths, consequently, depending on the number of leaves in a specific subtree, various leaves are used for homogeneous paths. Before determining the exact layout of C_r, we first reduce the bad labelled component group tree to its core, the residual tree, whose optimal cover corresponds to C_r.

Let $T_\circ(A, B)$ with leaf composition $\mathcal{L}(A, B) = (n_{\mathcal{A}}, n_{\mathcal{B}}, n_\varepsilon, n_{\mathcal{AB}})$ be reduced by using safe optimal homogeneous paths until we have a tree whose leaf composition satisfies the bounds for heterogeneous paths, yielding the *residual tree* $T_r(A, B)$. Let the number of $\mathcal{A}$-, $\mathcal{B}$-, ε- and $\mathcal{AB}$-leaves in T_r be denoted by $r_{\mathcal{A}}$, $r_{\mathcal{B}}$, r_ε and $r_{\mathcal{AB}}$, respectively. Then the *residual leaf composition* is the quadruple $\mathcal{L}_r(A, B) = (r_{\mathcal{A}}, r_{\mathcal{B}}, r_\varepsilon, r_{\mathcal{AB}})$.

For the ε-subtree, the costs of short and long homogeneous paths differ. The following (counter-)example best describes why, although we can reduce the number of heterogeneous paths ending in the ε-subtree to two or one, the value of r_ε might be higher than that.

Example 23: Figure 6.15 shows two trees that have the same separation and $\mathcal{A}$- and $\mathcal{B}$-subtrees, merely the ε-subtree has a single leaf in the left tree and three leaves in the tree depicted on the right (they both do not have links). The left tree shows an example of an optimal cover that has cost 4. Obviously, using a homogeneous ε-path

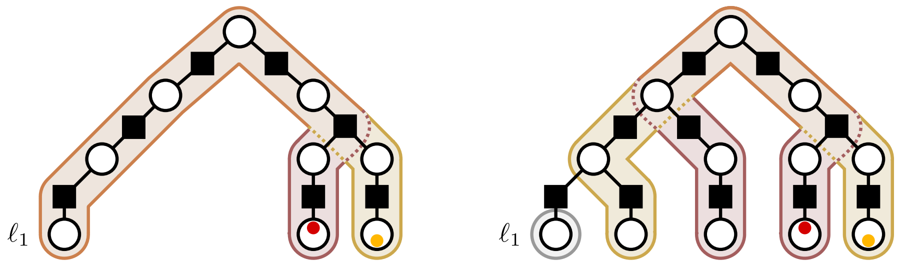


Figure 6.15: Two trees with the same separations and no links. Left: A tree with $\mathcal{L} = (1, 1, 1, 0)$ and an optimal cover that has cost 4. Right: A tree with $\mathcal{L} = (1, 1, 3, 0)$ and an optimal cover of cost 5 using two heterogeneous paths and a short ε-path.

in the tree on the right-hand side has cost 2 and yields the tree on the left that has cover cost 4, resulting in overall cost of 6. An alternative, and optimal, cover uses a short ε-path and two heterogeneous paths, resulting in cost 5 in total. $\diamond$

Hence it is not always optimal to reduce the number of ε-leaves to a single leaf. This is why, for $n_\varepsilon \leq 3$, we do not reduce the ε-subtree and $r_\varepsilon = n_\varepsilon$. Otherwise, for $n_\varepsilon > 3$, the reduction yields $r_\varepsilon = 2$ for even n_ε and $r_\varepsilon = 3$ for odd values of n_ε.

Similarly, in the $\mathcal{A}$- and $\mathcal{B}$-subtrees, as each long homogeneous path uses exactly two leaves, $n_\mathcal{A}$ is reduced by an even number. Due to Propositions 10 and 11 that show that we can reduce the number of $\mathcal{A}$-leaves used for heterogeneous paths, and bearing in mind single remaining leaves, the reduction yields $r_\mathcal{A} = 3$ for odd $n_\mathcal{A}$ and $r_\mathcal{A} = 2$ for even $n_\mathcal{A}$. For $n_\mathcal{A} \leq 3$ there is no reduction by homogeneous paths and $r_\mathcal{A} = n_\mathcal{A}$. Corollary for the $\mathcal{B}$-subtree. From now on, we assume w.l.o.g. that $r_\mathcal{A} \geq r_\mathcal{B}$.

For the $\mathcal{AB}$-subtree, we have $r_\mathcal{AB} = n_\mathcal{AB}$ if $n_\mathcal{AB} \leq 4$, otherwise the reduction yields $r_\mathcal{AB} = 4$ for even $n_\mathcal{AB}$ and $n_\mathcal{AB} = 3$ for odd values of $n_\mathcal{AB}$.

Recall, as described in the beginning of this chapter, how a general covering path creates a *meta-vertex* and the difference between safe and unsafe paths (see for instance Figures A.1 and A.2 on pages 181 and 182 in the appendix). The effect of a reduction step is essentially the same as for a general covering path. However, we choose only safe optimal homogeneous paths for the reduction thus assuring that no further costs are induced.

A subtree that does not share a vertex with the other subtrees can be treated entirely independent of these. In previous work it was shown that, in this case, the unlabelled subtree can be covered by safe paths entirely such that r_ε leaves remain [55]. In order to ensure that all internal bad nodes are covered, the leaves for the homogeneous paths are chosen such that not direct siblings are used for one homogeneous path. This is best illustrated with the concept of meta-vertices: Covering paths that cross or overlap will be part of the same meta-vertex (if we do it one after the other, one such path will consume the meta-vertex that represents the other) and thus no new bad internal vertices or new bad leaves are created. The same can be applied to a labelled subtree that does not share vertices with other subtrees.

Although having a *solo ε-leaf* does not always have an effect on the overall cost, among the r_ε residual ε-leaves, we generalise the reduction independent of the case at hand and choose to keep at least one of the solo ε-leaves. This way, we do not deprive us later of the possibility to use it for an optimal cover. This is achieved by starting the reduction of the unlabelled subtree with ε-leaves on long branches for the long homogeneous paths. If there exists a solo ε-leaf it will thus be left behind among the r_ε leaves of $T_r(A, B)$.

For covering a subtree that shares one vertex or more with another subtree, the approach of Hannenhalli and Pevzner [55] is adapted in [43,44]. We treat each induced subtree by itself, using only the correct number of homogeneous paths, such that $\mathcal{L}_r(A,B) = (r_{\mathcal{A}}, r_{\mathcal{B}}, r_{\varepsilon}, r_{\mathcal{AB}})$ leaves are left behind. Again, at least one of the solo ε-leaves should remain.

Lemma 5: *Given a bad labelled component group tree $T_\circ(A,B)$ that has leaf composition $\mathcal{L}(A,B) = (n_{\mathcal{A}}, n_{\mathcal{B}}, n_{\varepsilon}, n_{\mathcal{AB}})$, we have:*

$$cost\big(T_\circ(A,B)\big) = \frac{n_{\mathcal{A}} - r_{\mathcal{A}}}{2} + \frac{n_{\mathcal{B}} - r_{\mathcal{B}}}{2} + (n_{\varepsilon} - r_{\varepsilon}) + \frac{n_{\mathcal{AB}} - r_{\mathcal{AB}}}{2} + cost\big(T_r(A,B)\big), \quad (6.6)$$

where T_r is the residual tree derived from $T_\circ$ (by the reduction described in this section) and that has residual leaf composition $\mathcal{L}_r(A,B) = (r_{\mathcal{A}}, r_{\mathcal{B}}, r_{\varepsilon}, r_{\mathcal{AB}})$.

Proof. The cost of the reduction is easily derived: By Propositions 10-16 we have $|P_{\mathcal{A}}| = \frac{n_{\mathcal{A}} - r_{\mathcal{A}}}{2}$ homogeneous safe paths for $\mathcal{A}$-leaves (and respectively for $\mathcal{B}$-, ε and $\mathcal{AB}$-leaves) and the cost for a homogeneous long path is 1 in a labelled subtree and 2 in the unlabelled subtree.

By the reduction we do not refrain from the original optimal cost, because the links that are present in $T_\circ(A,B)$ are not removed from the tree during the reduction, they are thus still present in $T_r(A,B)$. Also, if there is a solo ε-leaf, during the reduction this one will be left such that it remains in $T_r(A,B)$ and labelled leaves on short branches do not need special treatment. By the reduction we do not introduce new separating vertices. When the number of leaves of a non-empty subtree is even, no new bad leaf or separating vertex is created, as in each subtree there remain at least two leaves. When there is an odd number of leaves in a non-empty subtree, the reduction stops when there are only three leaves left. No new leaf or separating vertex is introduced. The reduction and the residual tree maintain the optimal cost of the bad component group tree, as no feature is introduced to T_r that would induce higher costs. Obviously, if, for some subtree $T_\circ^{\mathcal{X}}$ of type $\mathcal{X} \in \{\mathcal{A}, \mathcal{B}, \varepsilon, \mathcal{AB}\}$, no reduction is performed, we have $n_{\mathcal{X}} = r_{\mathcal{X}}$ and no cost is added. □

In concordance with the first lower and upper bounds from Proposition 9 on page 118 we can tighten the bounds to concern only the residual tree $T_r(A,B)$ with its r leaves:

$$\left\lceil \frac{r}{2} \right\rceil \leq cost(T_r) \leq r + 1. \quad (6.7)$$

Furthermore, the dissociation of the number of leaves for each leaf type, namely $r_{\mathcal{A}}$, $r_{\mathcal{B}}$,

r_ε and $r_{\mathcal{AB}}$ instead of simply r, allows us to particularise:

$$\left\lceil \frac{r_{\mathcal{A}} + r_{\mathcal{B}} + r_{\mathcal{AB}}}{2} \right\rceil + r_\varepsilon \leq cost(T_r). \tag{6.8}$$

As an $\mathcal{A}$-$\mathcal{B}$-path has cost 2 the lower bound might not be met in all cases. At the end of this chapter, we will see for which residual trees the lower bound is met.

Rather than bounds or approximations, we seek an exact solution. Now that we reduced the bad component group tree $T_\circ(A, B)$ to the residual tree $T_r(A, B)$ we can focus on finding an optimal cover for T_r and on computing its cost. First, we analyse residual trees with a single leaf type. Starting from there, we add another leaf type and yet another until in the end we have analysed all residual leaf compositions that are necessary. All studied cases then compose the exact solution to the general case in which we have a residual tree with all leaf types present.

6.4.1 Residual Trees With One Type of Leaf Labelling

As before, we first concentrate on trees where all leaves are of the same type, i.e. only one value among $r_{\mathcal{A}}$, $r_{\mathcal{B}}$, r_ε and $r_{\mathcal{AB}}$ is greater than zero. Obviously, no heterogeneous path is used. For a tree that has one or two leaves we use a single homogeneous path. For trees with three leaves we use a long homogeneous path and either a short path if it is on a short branch, or a long path, re-using a leaf. Here, an $\mathcal{AB}$-subtree that has four leaves can be optimally reduced to two leaves.

Unlabelled leaves. For a bad component group tree $T_\circ(A, B)$ with $\mathcal{L}(A, B) = (0, 0, n_\varepsilon, 0)$, the reduction to the residual tree $T_r(A, B)$ stops at $r_\varepsilon \geq 3$. The covers and costs of the unlabelled residual tree depend on the tree's properties as given below. Each line represents one case that is assigned an identifier (ID), provides a set of covering paths ($\mathtt{C}_r$) and its cost (a more elaborate description of the notation can be found in Section C of the appendix).

Tree properties	**ID**	**Cover $\mathtt{C}_r$ of $T_r(A, B)$**	$cost(\mathtt{C}_r)$
r_ε is even, or	I	ε–ε	2
r_ε odd **and** solo ε-leaf exists, or	S	$\{\varepsilon\} \cup \mathtt{C}'_r$ of T'_r with $\mathcal{L}'_r = (0, 0, r_\varepsilon - 1, 0)$	r_ε
none of the above apply	W	ε–ε, ε–$\underline{\varepsilon}$	4

Using long homogeneous paths for the leaves is optimal and safe if $r_\varepsilon < 3$ (cases 'I' and 'S') , or if $r_\varepsilon \geq 3$, which must then have a solo ε-leaf (case 'S'). For the latter, a short ε-path further reduces the tree and we have $\mathcal{L}_r' = (0,0,2,0)$. Otherwise, if no solo ε-leaf exists, we use two long homogeneous paths of which one is unsafe, rendering an internal node a leaf, resulting in $\mathcal{L}_r' = (0,0,2,0)$. This is case 'W' from above for which it is always optimal to simply re-use one leaf (indicated by underlining the leaf type as $\underline{\varepsilon}$) instead of considering the internal nodes for the covering path.

The cost of residual tree cover C_r added to the reduction cost $n_\varepsilon - r_\varepsilon$ composes the overall cost of $T_\circ$ yielding the well known formula [12,14] for the inversion distance offset (examined in Subsection 6.1.1).

Labelled leaves. For labelled leaves, a tree with three leaves on long branches also uses an unsafe long homogeneous path. However, the third leaf can be used with one of the other leaves at the same cost as a short path. In the other cases, we use long homogeneous paths until at most one leaf remains. This can be covered by a short path. Overall, for labelled leaves of type $\mathcal{X} \in \{\mathcal{A}, \mathcal{B}, \mathcal{AB}\}$ this yields:

$$cost\big(T_r(A,B)\big) = \left\lceil \frac{r_\mathcal{X}}{2} \right\rceil,$$

which added to the reduction cost $\frac{n_\mathcal{X} - r_\mathcal{X}}{2}$ yields the lower bound to the inversion-*indel* distance offset.

6.4.2 Residual Trees With Two Types of Leaf Labelling

In this section we study the case where two types of leaves are present in the residual tree $T_r(A,B)$. We assume $r_\mathcal{A} \geq r_\mathcal{B}$, hence, we either have both ε- and $\mathcal{AB}$-leaves or we have $\mathcal{A}$-leaves and one other leaf type. Some leaf compositions can be treated easily, as the following proposition shows.

Observation 10. In a residual tree $T_r(A,B)$ with exactly two leaves that are of different types, using a single heterogeneous path is always optimal and results in $cost(T_r) = 1$ if both have an $\mathcal{A}$- or both have a $\mathcal{B}$-label and $cost(T_r) = 2$ otherwise.

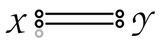
The above observation gives optimal covers and their cost of the following trivial leaf compositions:

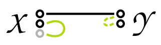
$$(1,1,0,0),\ (1,0,1,0),\ (1,0,0,1),\ (0,0,1,1).$$

The reduction, as given in this section, stops when we reach three leaves because for the residual tree we might have to use an unsafe path. Contrary to above, the values of n and r now represent two types of leaf labellings. However, an unsafe path can still only occur when there are three leaves left in total. We could reduce each subtree to at most two leaves by using only safe paths. That this is optimal for all combinations of two leaf types is shown below.

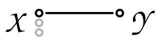
Proposition 17: *Let $T_r(A,B)$ be a residual tree with $\mathcal{L}_r(A,B) = (r_\mathcal{A}, r_\mathcal{B}, r_\varepsilon, r_{\mathcal{AB}})$ that has two empty subtrees and two non-empty subtrees $T_\circ^\mathcal{X}$ and $T_\circ^\mathcal{Y}$ of types $\mathcal{X}, \mathcal{Y} \in \{\mathcal{A}, \mathcal{B}, \varepsilon, \mathcal{AB}\}$, where $\mathcal{X} \neq \mathcal{Y}$. If $T_\circ^\mathcal{X}$ has more than two leaves, it is always optimal to use a long homogeneous $\mathcal{X}$-path and derive $r_\mathcal{X} = 1$.*

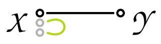
Proof. Let us assume we reduced the concerned subtree of type $\mathcal{X}$ with $n_\mathcal{X} \geq 3$ leaves by an even number and that $r_\mathcal{X} = 3$ leaves are left. Furthermore, let $h_\mathcal{X}$ be the number of $\mathcal{X}$-leaves used for heterogeneous paths which we know from Lemma 4 must be two or fewer. If $h_\mathcal{X} = 2$, it means one of the $\mathcal{X}$-leaves is covered elsewise (either by a short path, or as a link, which would be unnecessary since a possible separation would be covered by the heterogeneous paths). In all combinations of subtree types present, it is always possible, with lower or equal cost, to use two homogeneous paths such that at most one heterogeneous path remains (possibly re-using a leaf in the other subtree or using a link). The graphic below shows in black the $\mathcal{X}$-leaves used for heterogeneous paths and in grey the other residual leaves.

$h_\mathcal{X} = 2:$ 

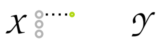


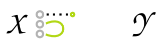
If $h_\mathcal{X} = 1$, two leaves are unaccounted for. This means they must both be used for either a long homogeneous path, or links and short paths. For two leaf types, one link is enough and two short paths would always be replaced by a long homogeneous path. This means we can always use a long homogeneous $\mathcal{X}$-path and, if necessary, re-use a leaf for the links which would be optimal.

$h_\mathcal{X} = 1:$ 



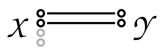
If $h_\mathcal{X} = 0$, it must mean either there is no separation or we use a link to cover it. In either case we can use two leaves in a long homogeneous path such that only one leaf remains. The remaining leaf is covered either by a short path or by a long path (connected to a link or a re-used leaf). The graphics below shows the three grey $\mathcal{X}$-leaves of which at most one is used as a link.

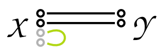
$h_\mathcal{X} = 0:$ 



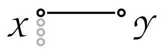
In any case, using one more homogeneous path (thus reducing the number of leaves used for heterogeneous paths to a single leaf) is always possible.

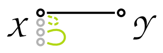
Now let us assume that $n_\mathcal{X} \geq 4$ is even such that $r_\mathcal{X} = 4$ leaves are left. If $h_\mathcal{X} = 2$ it means $r_\mathcal{X} - h_\mathcal{X} = 2$ leaves are left and can be covered by a long homogeneous path, as the following graphics shows.

$h_{\mathcal{X}} = 2:$ 

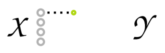


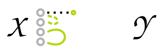
For $h_{\mathcal{X}} = 1$ three leaves must be left for homogeneous paths, of which two are used in a long path and the last remains for a short or long path.

$h_{\mathcal{X}} = 1:$ 



For $h_{\mathcal{X}} = 0$ four leaves are left. At most one of them is used as a link such that at least one pair of $\mathcal{X}$-leaves can be covered by a long homogeneous path as the following graphics shows.

$h_{\mathcal{X}} = 0:$ 



Thus, when exactly two leaf types are present, it is always possible to achieve a value of $r_{\mathcal{X}} \leq 2$ by using only homogeneous paths in the reduction while not increasing the cover cost. $\square$

When there are more than two leaf types, counter-examples showed that the above proposition does not apply to ε- and $\mathcal{AB}$ leaves (see Examples 22 and 23 on pages 133 and 135). However, for $\mathcal{A}$- and $\mathcal{B}$-leaves we can extend the above proposition towards three or four present leaf types.

Corollary 4 ($\mathcal{A}$- and $\mathcal{B}$-leaves in trees with three or more leaf types)**.** The argumentation of Proposition 17 is also valid for $\mathcal{A}$-leaves when more than two leaf types are present. We use safe paths in the reduction and, at the same time, at most two leaves are used for heterogeneous paths. As short and long paths for the $\mathcal{A}$-leaves have the same cost, any third leaf left in the reduction can simply be covered by a long homogeneous paths connecting it with another $\mathcal{A}$-leaf. The same is true for $\mathcal{B}$-leaves.

Observation 11. A non-separated ε-subtree in the residual tree $T_r(A, B)$ can only exist if $r_\varepsilon \geq 2$, or if n_ε is odd and T_r has a solo ε-leaf. Otherwise, by a reduction to $r_\varepsilon = 1$, the ε-subtree consists solely of a single vertex, which is the leaf itself, and the bad vertices on the long branch become separating vertices.

Proposition 18: *Given a labelled component group tree $T_\circ$ with two or more leaf types. If the $\mathcal{A}$-subtree is non-separated before the reduction of $T_\circ$ to the residual tree T_r, then afterwards either there is also no separation, or it is compensated for.*

Proof. For a bad labelled component group tree with even $n_{\mathcal{A}}$, the reduction yields two leaves. The way the reduction works, the root of the subtree $T_\circ^{\mathcal{A}}$ is still the same, thus no separating vertices were deleted or introduced. In case $n_{\mathcal{A}} \geq 3$ is odd, the $\mathcal{A}$-subtree is reduced to a single leaf. If this leaf is on a short branch, clearly only the single bad vertex remains and no separating vertices are deleted or introduced. Otherwise, if the last leaf is on a long branch, the internal vertices of that branch separate this leaf (the subtree) from the combined subtree of the other types, even if the subtree was non-separated in $T_\circ$. However, during the reduction, the covering paths of the $\mathcal{A}$-subtree are merged such that there is an internal meta-vertex with an $\mathcal{A}$-label in a way that allows to compensate for the new separating vertices by an $\mathcal{A}$-link. Because of this, and the fact that a long and a short path in the $\mathcal{A}$-subtree have the same cost, short branches do not need special care during the reduction. A visualisation of this is given in the appendix in Subsection A.2. □

Corollary 5. Clearly, the reduction for the $\mathcal{B}$-subtree and the $\mathcal{AB}$-subtree also compensate for long branches if the subtrees were non-separated before.

In summary, for two leaf types a residual tree has either one or two leaves in each of the two subtrees. We can thus effectively also omit leaf compositions with higher values leaving only the following to check for:

$$\mathbf{(0,0,1,2), (0,0,2,1), (0,0,2,2), (2,1,0,0), (2,2,0,0)}$$
$$\mathbf{(1,0,2,0), (2,0,1,0), (2,0,2,0), (1,0,0,2), (2,0,0,1), (2,0,0,2).}$$

Instead of elaborating the optimal covers determined for each of the leaf compositions, at this point we confine ourselves to describing only two cases. The complete list of covers for residual trees with two leaf types can be found in Subsection C.2 of the appendix. The optimal solutions were derived by an exhaustive analysis.

Example 24: Let $T_r(A,B)$ be a residual tree and let $\mathcal{L}_r(A,B) = (2,0,0,2)$ be the corresponding leaf composition. Using two heterogeneous $\mathcal{A}$-$\mathcal{AB}$-paths results in cost 2. This cover is optimal for any type of separation and no further properties can diminish the cost. This cover is also optimal for leaf compositions $(1,0,0,2)$ and $(2,0,0,1)$ where one leaf is re-used. ◇

Now we analyse a leaf composition where the different properties of the tree influence not only the cost but also the composition of covering paths.

Example 25: Let $T_r(A,B)$ be a residual tree that has leaf composition $\mathcal{L}_r(A,B) = (0,0,2,1)$. A brief notation of the influencing properties is shown below, which we subsequently elaborate (a more elaborate description of the notation can be found in Section C of the appendix). Each line represents one case that is assigned an identifier (ID) and provides a set of covering paths ($\mathtt{C}_r$) and its cost.

Tree properties		**ID**	**Cover $\mathtt{C}_r$ of $T_r(A,B)$**	***cost*$(\mathtt{C}_r)$**
No separating vertex exists,	or	I	ε–ε, $\mathcal{AB}$	3
an $\mathcal{A}$-ε-link exists,	or	La	ε–ε, $\mathcal{AB}$–$\mathcal{A}^*$	3
a $\mathcal{B}$-ε-link exists,	or	Lb	ε–ε, $\mathcal{AB}$–$\mathcal{B}^*$	3
a solo ε-leaf exists,	or	S	ε, ε–$\mathcal{AB}$	3
none of the above apply		W	ε–$\mathcal{AB}$, ε–$\underline{\varepsilon/\mathcal{AB}}$	4

If there is no separation, we use a long homogeneous ε-path and a short $\mathcal{AB}$-path amounting to overall cost 3 (this is cover 'I'). Otherwise, if the two subtrees are separated, we can still achieve cost 3: If there is an $\mathcal{A}$- or a $\mathcal{B}$-label that serves as link to the ε-subtree, we use a homogeneous ε-path and the $\mathcal{AB}$-leaf with the link label (covers 'La' or 'Lb' where the * indicates the labelled internal vertex). Thus, although $r_{\mathcal{A}} = 0$ the $\mathcal{A}$-ε-link is useful (the same applies to the $\mathcal{B}$-ε-link). If there is a solo ε-leaf, it can be covered by a short path and the other two leaves are covered by a long heterogeneous path (this is cover 'S'). If nothing of the above can be applied, the worst case ('W') uses a heterogeneous path and connects the second ε-leaf with any leaf covering its branch of bad vertices (re-using either of the other ε- or $\mathcal{AB}$-leaves would be optimal, where the underlining represents re-used leaves). ◇

6.4.3 Residual Trees With Three Types of Leaf Labelling

In this section we allow for one additional leaf type in the analysis, thus $\mathcal{A}$-leaves and two other leaf types are present. Again, we elaborate only exemplary cases and give a full catalogue of cases for three leaf types in Subsection C.3 of the appendix.

Example 26: For $\mathcal{L}_r(A,B) = (1,0,3,1)$ the exhaustive analysis showed that we can always find an optimal cover that uses a long homogeneous ε-path. Thus we can effectively reduce this case to leaf composition $\mathcal{L}'_r = (1,0,1,1)$. In fact, any leaf composition $\mathcal{L}_r(A,B) = (1,0,\mathit{odd},\mathit{odd})$ has the new base $\mathcal{L}'_r = (1,0,1,1)$. For this *further* reduction, the respective paths (ε- and/or $\mathcal{AB}$-paths) and their costs are additional to the new base case. ◇

That finding an optimal cover for the residual tree is not always as easy as this, is shown in the next example. Here, sometimes several properties must be present in order to achieve a certain cover cost.

Example 27: Let $T_r(A,B)$ be a residual tree with leaf composition $\mathcal{L}_r(A,B) = (2,2,2,0)$. Depending on the type of separation and the presence of solo ε-leaves or links, we have three different cost values. The table below lists, in a brief notation, all necessary optimal covers. The notation is similar as before (a more elaborate description can be found in Section C of the appendix). An $\mathcal{A}^*$ signifies an internal $\mathcal{A}$-node that is a link and a node that is re-used is underlined.

Tree properties				**ID**	**Cover $\mathbb{C}_r$ of $T_r(A,B)$**	*cost*$(\mathbb{C}_r)$
No separation, or				I	$\mathcal{A}$–$\mathcal{A}$, $\mathcal{B}$–$\mathcal{B}$, ε–ε	4
$\mathcal{A}$-subtree non-separated, or				IIa	$\mathcal{A}$–$\mathcal{A}$, $2\times(\mathcal{B}$–$\varepsilon)$	5
$\mathcal{B}$-subtree non-separated, or				IIb	$\mathcal{B}$–$\mathcal{B}$, $2\times(\mathcal{A}$–$\varepsilon)$	5
$\mathcal{A}$- and $\mathcal{B}$-subtree are separated **and**:						
	$\mathcal{A}$-$\mathcal{B}$-link	and	ε-subtree non-sep.	La	$\mathcal{A}$–$\mathcal{A}^*$, $\mathcal{A}$–$\underline{\mathcal{A}}$, $\mathcal{B}$–$\mathcal{B}$, ε–ε	5
or	"	"	solo ε-leaf exists	SLa	ε, ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{A}^*$, $\mathcal{B}$–$\mathcal{B}$	5
or	"	"	$\mathcal{A}$-ε-link exists	LLa	$2\times(\mathcal{A}$–$\mathcal{A}^*)$, ε–ε, $\mathcal{B}$–$\mathcal{B}$	5
or	$\mathcal{B}$-$\mathcal{A}$-link	and	ε-subtree non-sep.	Lb	$\mathcal{B}$–$\mathcal{B}^*$, $\mathcal{B}$–$\underline{\mathcal{B}}$, ε–ε, $\mathcal{A}$–$\mathcal{A}$	5
or	"	"	solo ε-leaf exists	SLb	ε, ε–$\mathcal{B}$, $\mathcal{B}$–$\mathcal{B}^*$, $\mathcal{A}$–$\mathcal{A}$	5
or	"	"	$\mathcal{B}$-ε-link exists	LLb	$2\times(\mathcal{B}$–$\mathcal{B}^*)$, ε–ε, $\mathcal{A}$–$\mathcal{A}$	5
none of the above				W	$\mathcal{A}$–$\mathcal{B}$, $\mathcal{B}$–ε, ε–$\mathcal{A}$	6

The worst case (cover 'W' with cost 6) only applies if all of the requirements of other cases are not met. This occurs when both the $\mathcal{A}$- and the $\mathcal{B}$-subtree are separated and we can find neither an $\mathcal{A}$-$\mathcal{B}$-link and at the same time have an $\mathcal{A}$-ε-link, a solo ε-leaf or the ε-subtree non-separated, nor can we find a $\mathcal{B}$-$\mathcal{A}$-link and at the same time have a $\mathcal{B}$-ε-link, a solo ε-leaf or the ε-subtree non-separated. ◇

Similarly to the tree for $\mathcal{L}_r(A,B) = (2,2,0,4)$ that was given in Example 22 (see page 133) not all ε- and $\mathcal{AB}$-subtrees can be reduced further. We know from Example 23 on page 135 that for $\mathcal{L}_r(A,B) = (1,1,3,0)$ we can not always do the reduction by a homogeneous ε-path. Fortunately, except for this situation (full separation of all subtrees and a solo ε-leaf but neither an $\mathcal{A}$-$\{\mathcal{B},\varepsilon\}$-link nor a $\mathcal{B}$-$\{\mathcal{A},\varepsilon\}$-link present) an exhaustive analysis showed that there is always an optimal cover that uses a long homogeneous ε-path yielding the new leaf composition $\mathcal{L}'_r = (1,1,1,0)$. The cost is then $2 + \mathit{cost}(\mathcal{L}'_r)$ for the respective tree properties and $\mathbb{C}_r = \mathbb{C}'_r \cup \{\varepsilon\text{–}\varepsilon\}$.

In Subsection C.3 of the appendix we give optimal tree covers and their costs for each property combination necessary. The costs were derived in an exhaustive analysis. For each leaf composition we chose covers that are simple or most general, although co-optimal covers may exist. The analysis showed that, for three leaf types, essentially only the leaf compositions listed below are necessary base cases. Leaf types that have three or more leaves and cannot be reduced further are emphasized in cyan if they refer to ε-leaves and in orange if they refer to $\mathcal{AB}$-leaves.

$$\begin{aligned}
n_{\mathcal{B}} = 0 :\ & (1,0,1,1), (1,0,1,2), (1,0,2,1), (1,0,2,2),\\
& (2,0,1,1), (2,0,1,2),\\
& (2,0,2,1), (2,0,2,2), (2,0,2,3).\\
n_{\varepsilon} = 0 :\ & (1,1,0,1), (1,1,0,2),\\
& (2,1,0,1), (2,1,0,2), (2,1,0,3),\\
& (2,2,0,1), (2,2,0,2), (2,2,0,3), (2,2,0,4).\\
n_{\mathcal{AB}} = 0 :\ & (1,1,1,0), (1,1,2,0), (1,1,3,0),\\
& (2,1,1,0), (2,1,2,0), (2,2,1,0), (2,2,2,0).
\end{aligned}$$

All other cases of three leaf types can be optimally reduced using long homogeneous paths until one of the base cases is reached. For example, $(2,0,5,5)$ is further reduced to $(2,0,1,1)$, while $(2,0,2,5)$ is reduced only to $(2,0,2,3)$. The optimal cover and its cost can then be derived by taking into account the homogeneous path(s) for the further reduction and the cover and cost given by the respective base case (for the same tree properties).

6.4.4 Residual Trees With Four Types of Leaf Labelling

We now examine the remaining residual trees in which all four subtrees are present. Again, we find that some leaf compositions can be further reduced and some cannot. There are easy leaf compositions such as $\mathcal{L}_r = (2,1,1,2)$ for which the following cover yields cost 4 and is always optimal: $\mathcal{B}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–ε. Furthermore, an exhaustive analysis showed that also leaf compositions $(2,1,3,2)$, $(2,1,1,4)$ and $(2,1,3,4)$ can always be optimally reduced to $(2,1,1,2)$ by using the corresponding homogeneous path(s) and then the base cover accordingly.

For other leaf compositions we find, that although they are *in general* not reducible, they are *mostly reducible*, which means except for one or a few combinations of tree properties, reduction is possible. An example for this is $\mathcal{L}_r = (2,2,1,3)$ whose optimal cover yields a cost lower than the reduction if, and only if, both the $\mathcal{A}$- and the $\mathcal{B}$-subtrees are separated while at the same time we neither have an $\mathcal{A}$-$\mathcal{B}$-link nor a $\mathcal{B}$-$\mathcal{A}$-link. When other properties of the tree are given, we can optimally reduce the $\mathcal{AB}$-subtree and look up the respective properties in $\mathcal{L}_r' = (2,2,1,1)$.

Novel are the cases where the ε-subtree is reducible but the $\mathcal{AB}$-subtree is *not* reducible. This is the case for the residual leaf composition $\mathcal{L}_r = (2,1,3,3)$, for example.

Here, $\mathcal{L}_r$ can be optimally further reduced to $\mathcal{L}_r' = (2, 1, 1, 3)$ for which we can then look up which properties allow the use of homogeneous $\mathcal{AB}$-paths and which do not.

The covers and their costs for the individual leaf compositions are too extensive and will be shown only in the appendix in Subsection C.4.

6.4.5 Strategy for Reduction in the General Case

In conclusion, when approaching the residual tree by reduction, we would generally stop at $r_\mathcal{A}, r_\mathcal{B}, r_\varepsilon \leq 3$ and $r_\mathcal{AB} \leq 4$ when allowing any types of subtrees and arrangements. We showed that, if we have more than one leaf type, the $\mathcal{A}$-subtree can be reduced from three to a single leaf. The same is true for the $\mathcal{B}$-subtree. Furthermore, after our exhaustive analysis that was done above, we can quickly check if it is optimal to do another reduction in any of the cases. We summarise the findings below.

Reduction of the $\mathcal{A}$-subtree when $r_\mathcal{A} = 3$:

If we have at least one other non-empty subtree, a reduction of $r_\mathcal{A} > 2$ to $r_\mathcal{A} \leq 2$ is optimal. Otherwise, using another homogeneous $\mathcal{A}$-path is unsafe but still optimal.

Reduction of the $\mathcal{B}$-subtree when $r_\mathcal{B} = 3$:

If we have at least one other non-empty subtree, a reduction of $r_\mathcal{B} > 2$ to $r_\mathcal{B} \leq 2$ is optimal. Otherwise, using another homogeneous $\mathcal{B}$-path is unsafe but still optimal.

Reduction of the ε-subtree when $r_\varepsilon = 3$:

Except for $\mathcal{L}_r = (1, 1, 3, 0)$ all residual leaf compositions with $r_\varepsilon = 3$ have at least one optimal cover that uses a homogeneous ε-path. If we have at least one other non-empty subtree, then for the ε-subtree only a single leaf remains. Otherwise, for $\mathcal{L}_r = (0, 0, \mathbf{3}, 0)$ the homogeneous ε-path is optimal but, if no solo ε-leaf exists, it is unsafe.

Reduction of the $\mathcal{AB}$-subtree when $r_\mathcal{AB} = 3$:

If we have residual leaf compositions $(1, 1, 2, 3)$, $(2, 0, 2, 3)$, $(2, 1, r_\varepsilon, 3)$ or $(2, 2, r_\varepsilon, 3)$ we cannot always use a long homogeneous $\mathcal{AB}$-path. For other leaf compositions we do the reduction step, although for $\mathcal{L}_r = (0, 0, 0, 3)$ it is optimal but not always safe.

Reduction of the $\mathcal{AB}$-subtree when $r_\mathcal{AB} = 4$:

Except for leaf compositions $(2, 2, r_\varepsilon, 4)$ and $(2, 1, 2, 4)$, any residual tree that has four $\mathcal{AB}$-leaves has at least one optimal cover that uses a long homogeneous $\mathcal{AB}$-path, which means the tree can be further reduced such that the $\mathcal{AB}$-subtree has only two leaves.

A faster check is done when simply adding up the $r_{\mathcal{A}}$ and $r_{\mathcal{B}}$ values, let this number be denoted by $S_{\mathcal{AB}}$. Assuming they are present and not the only subtree, the ε- and the $\mathcal{AB}$-subtree can be further reduced if $S_{\mathcal{AB}} < 2$, while for $S_{\mathcal{AB}} = 2$ only the $\mathcal{AB}$-subtree and for $S_{\mathcal{AB}} > 2$ only the ε-subtree can be further reduced.

Even in the cases that are not generally reducible by homogeneous $\mathcal{AB}$- or ε-paths some trees have properties that allow for a reduction. Details must be looked up for the specific leaf compositions.

Further Observations

We observe the following from Table 6.1 (page 149) and from the catalogue of leaf compositions given in Section C of the appendix.

The leaf compositions $(1, 0, 2, 2)$, $(1, 0, 2, 4)$, $(2, 0, 2, 1)$, $(2, 0, 2, 3)$, $(0, 0, 2, 1)$ and $(0, 0, 2, 3)$, that have no $\mathcal{B}$-leaf, sometimes require the use of a $\mathcal{B}$-link with an $\mathcal{AB}$-leaf. The latter two of these may even require the use of an $\mathcal{A}$-link with an $\mathcal{AB}$-leaf.

So-called *re-used* leaves occur mostly in cases where n is odd and we produce an unsafe path because there exists no solo ε-leaf or because a labelled leaf is not yet covered and for simplicity connected to a previously used leaf (whether it is on a short or on a long branch). Remarkably, also for even n re-used leaves can occur in rare cases, namely for residual leaf compositions $(2, 2, 0, 0)$ and $(2, 2, 2, 0)$, and only, if using a link is necessary. In either case all leaves are used solely in long covering paths, thus an even number are used which lets no single leaf stand behind. Nonetheless, we still need a link in order to complete the cover. Consequently, a previously covered labelled leaf is connected to the link.

The tree properties can evoke *three* different cover cost values for each leaf composition $(2, 2, r_\varepsilon, 0)$ with $r_\varepsilon \geq 0$. In all four cases, the range of values between worst and best case is 2.

The upper bound, which means $cost(T_r) \leq r + 1$, is met for the worst case of the residual leaf compositions $(0, 0, 2, 1)$, $(0, 0, 3, 0)$, $(1, 0, 2, 0)$, $(1, 1, 1, 0)$ and $(1, 1, 3, 0)$. These all have an odd number of leaves in the residual tree and all involve ε-leaves. All other leaf compositions' worst cases achieve lower cost.

The lower bound $cost(T_r) \geq \left\lceil \frac{r_{\mathcal{A}}+r_{\mathcal{B}}+r_{\mathcal{AB}}}{2} \right\rceil + r_\varepsilon$ given in Inequality (6.8) is met by all leaf compositions, except for $(1, 1, r_\varepsilon, 0)$ with $r_\varepsilon \geq 0$. For these leaf compositions there is no pair of leaves that can be covered by a long path of cost 1, as no pair of leaves share an $\mathcal{A}$-label or share a $\mathcal{B}$-label. Hence, the minimum cost are raised by 1 compared to the lower bound.

6.5 Chapter Summary

In this chapter we concentrated on finding a complete tree cover of the labelled component group tree $T_\circ(A, B)$ that has minimal cost under the cost scheme presented in the previous chapter.

After giving first upper and lower bounds to the cost of a tree cover of $T_\circ$, we were able to show that there are a certain number of homogeneous paths and an upper bound to the number of heterogeneous paths used, such that we can reduce the different subtrees that are composed of one leaf type each resulting in the so-called *residual tree* $T_r(A, B)$.

We observed how different properties of the tree(s) influence not only the composition of the cover, but also the overall cost. Among these properties are first and foremost the *leaf composition*, thus the number of leaves in the subtrees composed of the same leaf type. The arrangement of separating vertices produces different kinds of *separations* of the subtrees that are composed of the same leaf type. Furthermore, we studied the implications of a *solo* ε*-leaf* and the presence of *links* that may compensate for certain separations. We analysed all leaf compositions for the residual tree in terms of covering paths and cost for the different tree properties that exert influence thereon and provided details.

This chapter has thus successfully provided the cost of a tree cover which represents the offset of the inversion-*indel* distance to the DCJ-*indel* distance in presence of bad component groups. We will discuss this in more detail in the following chapter.

Table 6.1: Costs of optimal covers for leaf compositions of $T_r(A, B)$. Cases that are fully reducible are indicated by a '.' The '[]' signifies that only parts of that case are reducible. *lb*, *ub* and *r* indicate the relation of cover cost to the number of leaves and the upper and lower bounds (Inequalities 6.7 and 6.8). For space reasons the notation of the leaf composition $\mathcal{L}_r = (r_{\mathcal{A}}, r_{\mathcal{B}}, r_\varepsilon, r_{\mathcal{AB}})$ is condensed to $r_{\mathcal{A}}r_{\mathcal{B}}r_\varepsilon r_{\mathcal{AB}}$.

$r_{\mathcal{A}}r_{\mathcal{B}}$	$r_{\mathcal{AB}} = 0$				$r_{\mathcal{AB}} = 1$				$r_{\mathcal{AB}} = 2$				$r_{\mathcal{AB}} = 3$				$r_{\mathcal{AB}} = 4$			
00	– – 0000	1 *lb* 0010	2 *lb* 0020	3, 4 *lb*,*ub* 0030	1 *lb* 0001	2 *lb* 0011	3, 4 *lb*,*ub* 0021	· · ·	1 *lb* 0002	2, 3 *lb*, *lb*+1 0012	3, 4 *lb*, *lb*+1 0022	· · ·	[2], [2] *lb* 0003	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·
10	1 *lb* 1000	2 *lb* 1010	3, 4 *lb*,*ub* 1020	· · ·	1 *lb* 1001	2, 3 *lb*, *r* 1011	3, 4 *lb*, *r* 1021	· · ·	2 *lb* 1002	3 *lb* 1012	4, 5 *lb*, *r* 1022	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·
20	1 *lb* 2000	2, 3 *lb*, *r* 2010	3, 4 *lb*, *r* 2020	· · ·	2 *lb* 2001	3 *lb* 2011	4, 5 *lb*, *r* 2021	· · ·	2 *lb* 2002	3, 4 *lb*, *r*-1 2012	4, 5 *lb*, *r*-1 2022	· · ·	· · ·	· · ·	5, [6] *lb*, *r*-1 2023	· · ·	· · ·	· · ·	· · ·	· · ·
11	2 *lb*+1 1100	3, 4 *lb*+1,*ub* 1110	4 *lb*+1 1120	5, [6] *lb*+1,*ub* 1130	2 *lb* 1101	3 *lb* 1111	4, 5 *lb*, *r* 1121	· · ·	2 *lb* 1102	3, 4 *lb*, *r*-1 1112	4, 5 *lb*, *r*-1 1122	· · ·	· · ·	· · ·	5, [6] *lb*, *r*-1 1123	· · ·	· · ·	· · ·	· · ·	· · ·
21	2, 3 *lb*, *r* 2100	3, 4 *lb*, *r* 2110	4, 5 *lb*, *r* 2120	· · ·	2, 3 *lb*, *r*-1 2101	3, 4 *lb*, *r*-1 2111	4, 5 *lb*, *r*-1 2121	· · ·	3 *lb* 2102	4 *lb* 2112	5, 6 *lb*, *r*-1 2122	· · ·	3 *lb* 2103	4, [5] *lb*, *r*-2 2113	5, [6] *lb*, *r*-2 2123	21[3]3 =2113 +2	· · ·	· · ·	6, [7] *lb*, *r*-2 2124	· · ·
22	2, 3, 4 *lb*, *r*-1, *r* 2200	3, 4, 5 *lb*, *r*-1, *r* 2210	4, 5, 6 *lb*, *r*-1, *r* 2220	· · ·	3, 4 *lb*, *r*-1 2201	4, 5 *lb*, *r*-1 2211	5, 6 *lb*, *r*-1 2221	· · ·	3, 4 *lb*, *r*-2 2202	4, 5 *lb*, *r*-2 2212	5, 6 *lb*, *r*-2 2222	· · ·	4 *lb* 2203	5 *lb* 2213	6, [7] *lb*, *r*-2 2223	22[3]3 =2213 +2	4 *lb* 2204	5, [6] *lb*, *r*-3 2214	6, [7] *lb*, *r*-3 2224	22[3]4 =2214 +2
30	[2], [2] *lb* 3000	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·	· · ·
r_ε	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

Colour code: equal to lower bound (*lb*) – black / equal to upper bound – *ub*=*r*+1 / otherwise: *lb*+1=*r*, *lb*+1=*r*-1, *lb*+1=*r*-2, *lb*+1=*r*-3, *lb*+2=*r*.

A smooth sea never made a skillful sailor.

English proverb

Chapter 7

Inversion-*indel* Distance and Sorting

The topic of the inversion-*indel* model has been addressed in the previous two chapters and the reference, the DCJ-*indel* model, was subsumed in Chapter 4.

First, in Chapter 5, we covered several special cases. One of these was computing the inversion-*indel* distance of two unichromosomal circular genomes in absence of bad component groups. The solution to that also offered a valuable clue on how the sorting itself is performed. Subsequently, we were able to show that it is possible to perform split inversions that correspond to optimal DCJ operations while at the same time maintaining the indel-potential of each cycle in the relational diagram. Later in that chapter, we constructed a more abstract data structure that represents the component groups of the pair of genomes.

In Chapter 6 we concentrated on ridding the relational diagram of bad component groups. Resolving bad component groups (to good component groups) requires neutral or joint inversions, which correspond to suboptimal DCJ operations. The objective is to perform the minimum number of overall inversions, insertions and deletions, thus also the minimum number of extra operations when compared to the DCJ-indel distance. We will now subsume the findings and draw conclusions for further use concerning the inversion-*indel* model.

7.1 The General Inversion-*indel* Distance

In order to recap the determination of the inversion-*indel* distance, we best describe how to derive the distance value from a pair of genomes in general. A visualisation of the actions is depicted in Figures 7.1 and 7.2.

Let A and B be two unichromosomal circular genomes without duplications. We first construct a relational diagram of A and B. After determining the cycles and component groups we can directly perform all mergings of $\mathcal{AB}$-cycles such that, in the end, we have at most one cycle with $\mathcal{A}$- as well as $\mathcal{B}$-labels. Here, we take care to perform the cuts of the inversion operations between the runs. We then check the characteristics of the component groups: if we do not find any bad component group, we can directly calculate the distance as presented in Sections 5.3 and 5.4:

$$d^{id}_{\mathrm{DCJ}}(A,B) = |\mathcal{G}| - c + \sum_{C \in \mathcal{R}(A,B)} \lambda(C),$$

for which the indel-potential $\lambda(C)$ of each cycle C, the number of cycles c and the number of common markers $|\mathcal{G}|$ are required.

If we detect bad component groups, we proceed by building the labelled component group tree $T(A,B)$ and transforming it into the bad labelled component group tree $T_\circ(A,B)$. The first observed property of the tree is the leaf composition $\mathcal{L}(A,B) = (n_{\mathcal{A}}, n_{\mathcal{B}}, n_\varepsilon, n_{\mathcal{AB}})$ and ensuing the residual leaf composition $\mathcal{L}_r(A,B) = (r_{\mathcal{A}}, r_{\mathcal{B}}, r_\varepsilon, r_{\mathcal{AB}})$. Depending on the number of leaf types present, we may further reduce single values of $\mathcal{L}_r(A,B)$. Note that actually performing the reduction(s) is not necessary, as it has already been performed in the exhaustive analysis that lead to the base cases. We look up the residual leaf composition (as given in the appendix). Depending on which properties of the tree influence the cost of this leaf composition, we need to check the tree for their presence, e.g., different separations or links. Also, the existence of a solo ε-leaf might influence the total cost. The case we looked up merely provides the cost of $T_r(A,B)$ and, inserted into the formula from Lemma 5, yields:

$$cost\big(T_\circ(A,B)\big) = cost\big(T_r(A,B)\big) + \frac{n_{\mathcal{A}} - r_{\mathcal{A}}}{2} + \frac{n_{\mathcal{B}} - r_{\mathcal{B}}}{2} + (n_\varepsilon - r_\varepsilon) + \frac{n_{\mathcal{AB}} - r_{\mathcal{AB}}}{2},$$

which then gives us the cost of an optimal tree cover of $T_\circ$ and thus the additional cost evoked by the presence of all bad component groups. In total the following formulae give the minimum number of steps necessary to sort genome A into genome B using only inversions, insertions and deletions:

$$d^{id}_{\mathrm{INV}}(A,B) = d^{id}_{\mathrm{DCJ}}(A,B) + \tau^*_{\mathrm{INV}}(A,B),$$

which expands to:

$$= |\mathcal{G}| - c + \sum_{C \in \mathcal{R}(A,B)} \lambda(C) + \frac{n_{\mathcal{A}} - r_{\mathcal{A}}}{2} + \frac{n_{\mathcal{B}} - r_{\mathcal{B}}}{2} + (n_\varepsilon - r_\varepsilon) + \frac{n_{\mathcal{AB}} - r_{\mathcal{AB}}}{2} + cost\big(T_r(A,B)\big).$$

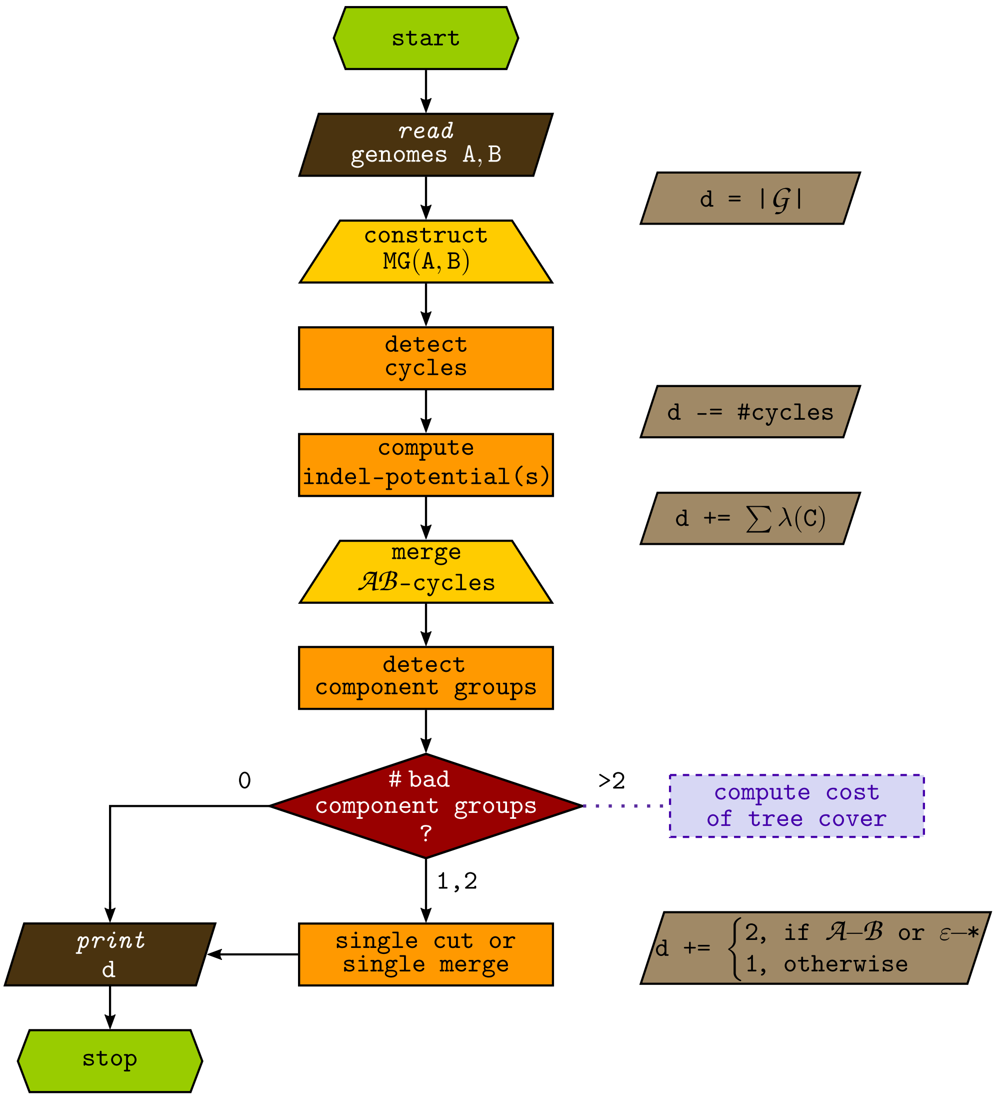


Figure 7.1: Visualisation of steps for the computation of the inversion-*indel* distance. The subroutine for computing the cost of an optimal cover is shown in Figure 7.2.

If we were to use a notation to specifically address each combination of tree properties instead of $cost(T_r)$ we could thus address the cost of the residual tree as:

$$cost(\mathcal{L}_r(A,B), \{separations\}, \{links\}, solo\ \varepsilon).$$

Then all possible combinations need to be listed in order to not take the worst case directly, when others would apply. The catalogue given in the appendix addresses the residual leaf compositions as a first key. For each leaf composition a procedure-like manner is given, that corresponds to a sequence of if-thens where the order imposes

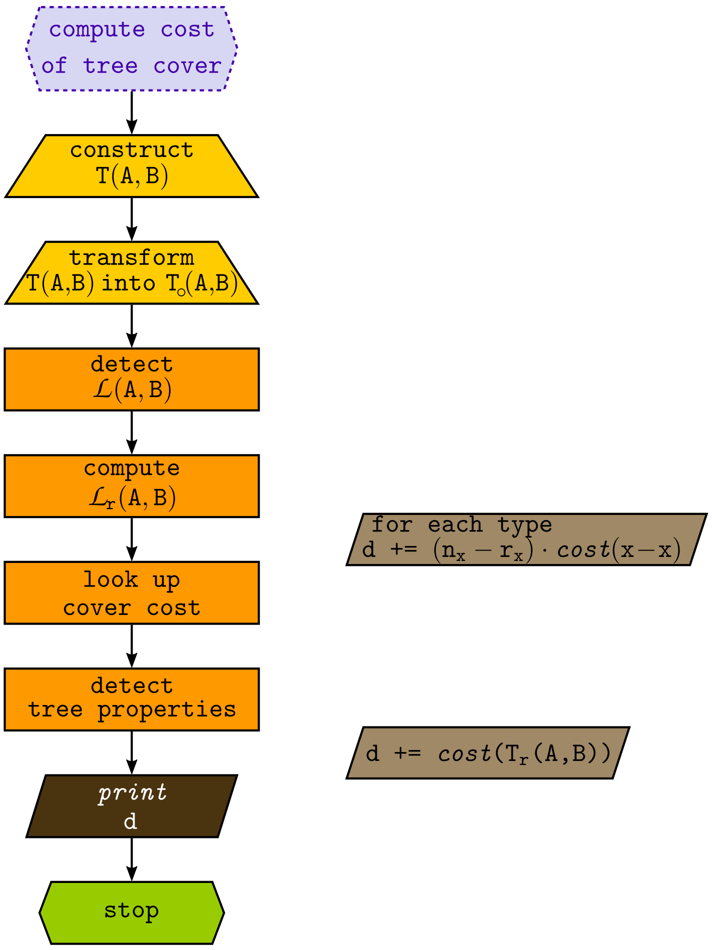

compute cost of tree cover
construct T(A, B)
transform T(A,B) into T∘(A,B)
detect $\mathcal{L}$(A, B)
compute $\mathcal{L}_r$(A, B)
look up cover cost
detect tree properties
print d
stop
for each type d += $(n_x - r_x) \cdot cost(x-x)$
d += $cost(T_r(A,B))$

Figure 7.2: This subroutine continues the case from Figure 7.1 in which there are bad component groups present and shows the steps from constructing the tree until the inversion-*indel* distance is obtained.

the use of else-if already. We thus do not directly apply such a tuple as proposed above. It should be noted that the exhaustive analysis also showed that all properties listed in Subsection 6.2.4 are indeed sufficient.

Complexity analysis. We know from [26] that the DCJ-*indel* distance can be computed in linear time with respect to the input genomes. Before we construct the tree data structure, we merge all $\mathcal{AB}$-cycles into a single $\mathcal{AB}$-cycle. If there is a single such cycle that is bad and contains four or more labels, we perform a neutral inversion fusing two pairs of runs. Furthermore, in [12,68] it was shown that the component groups of a pair of (linear) genomes can be found in $\mathcal{O}(|\mathcal{G}|)$ time.

After detecting the number of bad component groups (red lozenge of Figure 7.1) we have three options. In case we have fewer than three component groups, the distance computation is straightforward. Otherwise, we have to handle the bad component groups differently by constructing the bad labelled component group tree.

In [12,68] it was shown how a component group tree can be constructed from the component groups by walking along one of the input genomes. We showed in Chapter 5 that any linear representation of the relational diagram produces the same tree, hence, we simply choose one fixation for the detection of component groups. For each position, it is checked, in a data structure that stores the start and stop positions of component groups, if the start or end of a component was detected and a tree with corresponding nodes is built. This is possible in $\mathcal{O}(|\mathcal{G}|)$ time [12,68]. For our transformation of the labelled component group tree $T(A,B)$ into the bad labelled component group tree $T_{\circ}(A,B)$, we traverse T in a bottom-up manner and recursively remove good nodes. For each removed good node that has labels, its parent is assigned the union of both their labels. While doing this, we report the leaf composition $\mathcal{L}(A,B)$. After the traversal is finished, we compute $\mathcal{L}_r(A,B)$ and look up the cover cost.

The necessity of detecting different tree properties (which is possibly expensive) depends on whether it is requested by the base case that we are looking up. Let us assume it has to be done for all possible properties. Separations can be detected by determining subtrees and separating vertices. In a labelled component group tree, constructed as in [12,68], there exists a root. For a single leaf type, we traverse the tree in a bottom-up manner for each leaf of that type. The first leaf is followed until the root and all vertices on that path are marked. For the other leaves, we follow and mark the vertices on the path to the root until either the next marked vertex or the root is reached (marking it as the end of the path). Then, if the root is the end of exactly on such marked path, in a top-down manner, we remove the vertices that have a single marking until we reach a vertex that has more than one marking. The marked vertices determine the subtree for this leaf type. Obviously, the same is done for the other leaf types. A vertex that is not marked is then a *separating vertex.* Some base cases check for separations or non-separations of combined subtrees. In this case, a similar traversal has to be done for the concerned leaf types together. For *links*, we can check labels within the subtree during traversal. As the label could also be in the closest separating vertex of a specific subtree, we also need to check the separating vertices closest to the branching node of this subtree and the other (labelled) subtree. A leaf is a *solo ε-leaf* if its removal does not create new bad leaves or produces a new separation. We need to find at most one such ε-leaf. Any leaf on a long branch cannot be a solo ε-leaf. Detecting 'short' leaves was already described in [12] but we need to check the second requirement of the definition for solo ε-leaves. The interrelation of subtrees is intact in the following cases: for one, if the ε-subtree is separated from the other subtrees, and, for another, if the parent of the short ε-leaf has only edges that belong to the ε-subtree. Otherwise, careful analysis is necessary. A possible, but expensive, approach deletes the short ε-leaf, checks anew which separations exist (the markings from this leaf to the next branching node that has two ε-markings need to be removed) and then determines whether the treated leaf was a solo ε-leaf. Otherwise, we go back one step and check the same for any other short ε-leaf. Except for this last step, we achieve $\mathcal{O}(|\mathcal{G}|)$ time. Including the step, the worst case is $\mathcal{O}(n_\varepsilon \cdot |\mathcal{G}|)$, unfortunately. This strongly motivates to find a better way to determine solo ε-leaves.

7.2 Sorting with Inversions, Insertions and Deletions

Naturally, we not only want to know the distance of two genomes, that is, the length of a shortest sorting scenario. Rather, we also want to know the chronology and implementation of these operations on one genome in order to derive the other.

The sorting process of genomes A and B starts the same way as the computation of the distance: We first construct the relational diagram of the two genomes, determine the different cycles and perform all merges of $\mathcal{AB}$-cycles. Then the component groups are determined and, if bad component groups are present, the labelled component group tree $T(A,B)$ is constructed which is then transformed into the bad component group tree $T_\circ(A,B)$. The homogeneous safe paths are translated into their corresponding merges of cycles (and thus component groups) until the tree corresponds to the residual tree $T_r(A,B)$. The covers given in Section C of the appendix are then used to complete the resolving of all bad component groups. We are left with only good component groups. These still need to be sorted, for which only split inversions are used. However, prior to the final sorting with split inversions, we need to perform the insertions and deletions necessary. From Section 5.4 we already know how to find positions for this. In the end, we are left with a pair of sorted genomes.

7.2.1 The Residual Tree Cover as Precursor

The procedure described above is derived from the steps and proofs we undertook until we obtained the residual tree. In hindsight, there is an easier way that, at the same time, easily assures that the reduction of $T_\circ(A,B)$ to $T_r(A,B)$ is valid. In the following, we describe how the different steps can be arranged in another order and provide a small example.

When analysing the leaf composition of $T_\circ(A,B)$, we know what would be the resulting leaf composition of $T_r(A,B)$. We know an optimal cover for each T_r and thus whether we need links and/or a solo ε-leaf. The cover of $T_r(A,B)$ can be applied to $T_\circ(A,B)$ thus eliminating any separating vertices and leaving behind an even number of leaves for any type of subtree and a tree that has no separating vertices. This way, the reduction is more straightforward. After having turned all bad component groups into good component groups, like above, we sort the good component groups with split inversions and indels.

As an example, let us be given a tree with leaf composition $\mathcal{L}(A,B) = (8,6,0,1)$ and that has none of the requested features yielding lower cost. Instead of reducing the tree, we search for the reduced leaf composition $\mathcal{L}_r(A,B) = (2,2,0,1)$ in the list of base cases directly (see page 193) and find that case 'W' applies. We then apply a joint inversion to a cycle of an $\mathcal{A}$-leaf and a cycle of a $\mathcal{B}$-leaf. Another joint inversion is applied to an $\mathcal{A}$-cycle of an $\mathcal{A}$-leaf and an $\mathcal{A}$-cycle of the only $\mathcal{AB}$-leaf. The third joint inversion acts on a $\mathcal{B}$-cycle of a $\mathcal{B}$-leaf and a $\mathcal{B}$-cycle of the previously used $\mathcal{AB}$-leaf. Because we used up some leaves, by turning them into good component groups, the intermediate leaf composition is $\mathcal{L}'(A,B) = (6,4,0,0)$ and we know that after applying the cover from the residual tree, each subtree has an even number of leaves and no separation does exist. The remaining leaves in each subtree can thus be covered by long homogeneous paths until no bad leaf exists.

7.2.2 Sorting with Inversions and Deletions

Sorting with inversions and deletions was presented in [42,43]. In our representation, we can find all instances of trees with ε-leaves and $\mathcal{A}$-leaves in the leaf compositions $(1,0,1,0)$, $(1,0,2,0)$, $(2,0,1,0)$, and $(2,0,2,0)$. Let us compare the solutions by means of a small example. Leaf composition $(1,0,2,0)$ must use an $\mathcal{A}$-ε-link if both the following conditions are fulfilled: the subtrees are separated and there is no solo ε-leaf.

There are two reasons for which the theory presented in this earlier publication poses incorrect solutions. First, the definition of "short" leaf (treated as *simple hurdle* in that publication) was not altered with respect to the earlier inversion model. As we showed in a counter-example on page 123 (in Figure 6.6), covering a leaf on a seemingly short branch by a short path is not necessarily safe. The cases where we have a single ε-leaf in the residual tree that is not separated from the $\mathcal{A}$-subtree can only occur if either $n_\varepsilon = 1$ or if prior to the reduction an ε-leaf was present whose removal would create neither a new bad leaf nor a separation.

Secondly, the theory presented in [42,43] concentrated on merging *hurdles*, which are bad leaves of the bad labelled component group tree $T_\circ(A,B)$, derived by cutting all good leaves. Employing only these naturally excludes the use of internal vertices and any good leaves that might be labelled. Using only hurdles poses two flaws. For one thing, taking into account good leaves might allow us to use more labelled homogeneous paths, potentially reducing the cost of the tree cover with respect to paths ending in unlabelled vertices (this we showed in Subsection 5.6.2). The other weakness is that links do not occur in the theory of [42,43]. Neglecting the link in the example given above leads to higher cost which are obviously suboptimal.

7.3 Model Limitations and Extensions

In contrast to the DCJ and DCJ-*indel* model, that can be applied to genomes with any number of linear and/or circular chromosomes, the inversion-*indel* model as proposed in this thesis is applied to unichromosomal circular genomes only. The restrictions, that are posed on the type of genomes allowed, can be set aside in some way or another.

7.3.1 Multichromosomal Genomes

For one, the restriction to a single chromosome is due to the fact that an inversion cannot transfer markers from one chromosome to another. Another reason is that the deletion of markers from one genome and their insertion into another position is not permitted. However, if we have two multichromosomal genomes A and B without duplications and for each chromosome in the core genome $A|_{\mathcal{G}}$ there is a corresponding chromosome in $B|_{\mathcal{G}}$ with the same markers (and vice versa), no inversion needs to transfer a common marker to another chromosome. We can then regard the inversion-*indel* distance and sorting problems as several individual subproblems. Each instance then solves the inversion-*indel* distance on the pair of chromosomes that have the same set of common markers. A sorting scenario for such a pair of genomes can alter the order of inversions, insertions and deletions also across chromosomes. Genomes that contain one chromosome with common markers and one or several linear or circular singletons containing only unique markers are allowed, as the singletons require exactly one insertion or one deletion each.

7.3.2 Co-tailed Genomes

For the inversion model, results were found for linear genomes as well as for circular genomes [55,71,72]. We have shown in this thesis how inversion-*indel* distance and sorting problems are applied to unichromosomal circular genomes. If we compare a pair of linear chromosomes, the relational diagram would have two paths. From the DCJ-*indel* model [26] we know that recombination of paths is not always easily accomplished. However, we are untroubled by this, if all chromosomes of the core genomes are co-tailed (that means the telomeric $\mathcal{G}$-adjacencies in both genomes are identical, with exception of, perhaps, the label). In this case, the relational diagram has exactly two trivial paths and the other components are all cycles. We are then able to apply the theory to the relational diagram.

Capping (general linear genomes). If the chromosomes are linear but not co-tailed, we are still able to apply the inversion-*indel* model as presented in this thesis. A way to deal with this is by *capping*. Adding caps to chromosomes produces artificial telomeres for the linear chromosomes. This is modelled by adding two artificial markers, that are not present in any of the genomes, to the set of common markers. Commonly, when handling genomes that have $\mathcal{G} = \{1, \ldots, n\}$ we add 0 and $n + 1$ as common markers (regarding a single chromosome). One of each is appended to the ends of the chromosome in A and also in B. In this way the two existing paths of the relational diagram are closed into one or two cycles and also two new trivial paths are created. However, depending on which side the 0 and which side the $n + 1$ is added to each genome, the paths might be closed differently. For this, we quickly check the DCJ-*indel* distance of the two distinct possibilities and choose to continue with the one yielding lower cost. As the only unsorted components of the relational diagram are now cycles of one circular chromosome, we can apply the inversion-*indel* model as usual.

In the next and final chapter, we will subsume the contents of both parts of this thesis and give an outlook on future work.

The whole secret of life is to be interested in one thing profoundly and in a thousand things well.

Horace Walpole

Chapter 8

Final Remarks

8.1 Summary

Situated in the field of comparative genomics, this thesis covers several parts of genome rearrangements. The necessary abstractions of genomes, goals of comparative genomics and definitions of genome modification models were given in the first chapter, subsequently followed by a deeper introduction on the modelling with insertions and deletions (indels), graph data structures to represent the relation of a pair of genomes and the double cut-and-join (DCJ) model in Chapter 2.

The two main parts of the thesis then build on the DCJ model. In the first part, that is Chapter 3, the sampling of an optimal DCJ sorting scenario is studied. The method for sampling uniformly among sorting scenarios that do not recombine a pair of *AA*- and *BB*-paths was integrated into an existing software framework. The evaluation of real and artificial data shows that choosing other, more naïve sampling methods deviates notably from the expected value and our uniform sampling.

In the second part, comprised of Chapters 4–7, the DCJ model is used as a basis and extended in two different directions: first, towards the inclusion of indels (which is previous work and presented in Chapter 4), and second, towards the use of only inversions in a scenario (which is also previous work, presented in Section 5.3 and Subsection 5.6.1). The main contribution of this thesis is the combination of these two extensions, leading to the solution of the inversion-*indel* distance building on the DCJ(-*indel*) distance.

We showed that for good component groups, the indel-potential, which is achieved by using any optimal DCJ operation, is also preserved if the only rearranging operations allowed are inversions. In the next step, a data structure, the (bad) labelled component group tree, was used to represent the nesting relationship among component groups. An optimal cover of this tree with the considered cost scheme provides the minimum extra cost to handle bad component groups. Unlike simply adding the additional cost for indels to the additional cost for treating bad component groups, the modus operandi presented here seeks the minimum overall cost. Thus, we save indel operations while doing the extra inversions. Computing an optimal tree cover was studied in Chapter 6 and broken down to a finite number of base cases. In this way, we were able to provide an offset to compute the inversion-*indel* distance with respect to the DCJ-*indel* distance. Finally, we not only provide the cost for each base case, but also a cover. Along with all previously taken actions, this provides the means to finding an optimal inversion-*indel* sorting scenario.

8.2 Prospects

For both parts of this thesis, namely the sorting space of the double cut-and-join model and the sorting with inversions and indels, extensions and related problems arise.

Sampling Genome Modification Scenarios

For both the inversion [21] and the DCJ [22,23] model the sorting space has been investigated but only optimal scenarios without indels are considered.

Suboptimal sorting scenarios. Recently, Feijão [45] investigated the solution space of suboptimal DCJ scenarios, that is, scenarios with extra steps compared to the length of an optimal sorting scenario. The motivation for searching suboptimal scenarios lies in finding a maximum likelihood estimator for DCJ scenarios. Feijão found a formula for the number of scenarios that have up to one extra step. For suboptimal scenarios with more deviation from the distance no formula exists yet, but the problem could be interesting to study.

Sampling sorting scenarios with indels. We know from Section 2.3 that all components of the adjacency graph can be sorted individually using only optimal DCJ operations. The formula that gives the number of optimal scenarios also only considers individual sorting. However, as discussed in Chapter 3, only instances of the adjacency graph that do not have an *AA*-path while at the same time having a *BB*-path can be counted that way. If a recombination of an *AA*-path and a *BB*-path is possible, the derived number serves only as a lower bound to the number of optimal scenarios.

For the DCJ-*indel* model some recombinations have to be performed in order to achieve a scenario of shortest length. We could however, similarly to Section 3.3, restrict ourselves to circular genomes. In this case sorting each cycle individually is always optimal. For each run whose labels are accumulated in a single adjacency, the order of accumulating them as well as the label into which it is accumulated is subject to choice. In the same way, the merging of runs can be done at different points in the scenario or in different locations. Furthermore, we showed in Examples 11 and 12 (on page 75 f.) that even the mechanism of merging runs can be altered. In [26] DCJ-*indel* sorting was proposed where the focus was on maximising the number of DCJ operations or the number of indel operations within an optimal DCJ-*indel* scenario. Sampling a DCJ-*indel* sorting scenario is further complicated by the findings listed in Table 4.1. The case given in the last row, where an $\mathcal{A}$-run of each cycle is merged while at the same time a $\mathcal{B}$-run of each cycle is merged, maintains the DCJ-*indel* distance. This means that the merging of two cycles into a single cycle can be part of an optimal DCJ-*indel* scenario. The sorting space for the DCJ-*indel* distance thus not only needs to handle insertions and deletions but also the recombination of any pair of AB-cycles.

Since already the simple case of the DCJ-*indel* distance in which only cycles are present is sufficiently complicated, counting all optimal inversion-*indel* sorting scenarios is a very distant prospect.

Inversion-*indel* Distance and Sorting

The inversion-*indel* model has been addressed before [42,43,89] and finally we presented in this thesis the first complete treatment of distance and sorting by inversions, insertions and deletions. Although these two questions are answered, there are further problems that arise and still demand answers.

Software. The implementation of an algorithm, or rather pipeline, is still outstanding. The `UniMoG` software suite[1] already provides data structures for handling genomes and computing the DCJ-*indel* distance as well as the inversion distance [59]. And the display of a sorting scenario that includes inversions and indels is already in use for the DCJ-*indel* model. It is therefore a future endeavour to extend the existing tree

[1] `http://bibiserv.cebitec.uni-bielefeld.de/dcj/welcome.html`

data structure of the inversion model to include labels and refine the obtainment of the bad labelled component group tree. Furthermore, the base cases of the residual tree have to be included and the tree has to be searched for the properties necessary to give the optimal cost. For the sorting, a complete tree cover that gives optimal cost has to be translated into actual inversions.

Only then will the computation of the inversion-*indel* distance be automated and provided for a larger audience. Moreover, pairwise comparisons of multiple genomes will become feasible.

Co-optimal sorting scenarios. For the DCJ-*indel* model several sorting procedures were proposed [26]. One procedure maximises the number of DCJ operations in an optimal sorting scenarios. Another procedure maximises the number of indel operations in an optimal sorting scenario. A similar quest could be pursued for the inversion-*indel* model. For the case analysis usually only simple and sufficient covers were given, although for different properties several co-optimal covers are known to exist. For an analysis in maximising indels or inversions, the order in checking the properties that yield the same distance may have to be altered. Furthermore, some properties are not listed as there exist alternate co-optimal covers that do not require this specific property. It may be necessary to add further properties to the case analysis in order to minimise/maximise the number of indels in specific cases.

The merging of $\mathcal{AB}$-cycles is a simple attempt at ridding the relational diagram of bad component groups without extra cost. If the number of inversions is to be minimised, then performing an extra inversion to save indel operations should be carefully considered. For example, if there exist only good $\mathcal{AB}$-cycles and the subtree that contains all nodes with $\mathcal{AB}$-cycles, independent of their character (good/bad) or placement (internal/external), does not share any bad node with the rest of the tree, merging all $\mathcal{AB}$-cycles into a single cycle is not necessary. In this case the merging of $\mathcal{AB}$-cycles would not destroy any bad component groups or diminish the inversion-*indel* distance and could thus be neglected.

Content modifications. The computation of the double cut-and-join distance with indels under unit cost or distinct cost is well established [20,26,37]. The thesis at hand shows how to compute the inversion distance with indels under unit cost. Future work could include the computation of the inversion-*indel* distance with distinct operation cost or the computation of the inversion-*substitution* distance. We would assume an insertion to still have the same cost as a deletion, however, the inversions listed in Table 5.1 (see page 91) would then yield different overall cost. For instance, a joint inversion acting on two $\mathcal{AB}$-cycles changes the DCJ distance by +2 and reduces the overall indel-potential by −2 as it fuses two pairs of runs. This operation is optimal for the inversion-*indel* model under unit cost. With distinct operation cost this might be a strictly positive value. The idea to merge all $\mathcal{AB}$-cycles in the beginning, potentially bringing the number of bad component groups to zero (see Section 5.5), then needs to be evaluated against the extra steps necessary to handle the concerned bad component groups differently. Furthermore, the difference in indel cost changes the cost of paths in the tree and thus the composition of an optimal tree cover.

Pancakes. The unsigned prefix reversal problem that is also known as the *pancake flipping problem* [50] relates to the prefix inversion problem on unsigned genomes and was originally described like this:

> *"The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary."* [50]

Similarly, there is the sorting of *burnt pancakes* [32] that relates to the prefix inversion problem on signed genomes, in which not only the order of pancakes but the up-facing side is important.
If indels are included this could be the sorting of *burnt pancakes, crêpes and Pfannkuchen* problem. In adapting the above problem description, we could formulate the new problem as:

> *"There are two chefs in our place, one can prepare Pfannkuchen and the other can prepare pancakes and crêpes. In the process of preparing these, they get burnt on one of the sides. The chefs in our place are stubborn (they will only prepare their types of dish) and sloppy (the pancakes and Pfannkuchen come out all different sizes and the pancakes are stacked randomly with crêpes in-between). Therefore, when I deliver them to a customer, on the way to the table I have to rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom, and that their burnt sides face down) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. Unfortunately, some customers do not want crêpes, and I also have to remove them, preferably in as few moves as possible. Furthermore, some customers want Pfannkuchen, but, since they also come along in varying size, they have to be introduced to the stack and then sorted along with the pancakes from smallest to largest."*

Here, the prefix reversal also needs to take into account the grouping of insertions and the grouping of deletions. But, as the waiter cannot grab parts within the stack, the grouping has to respect that inversions can only reverse parts of the stack from the top.

Distance Computation via DCJ

A distance computation for general genome modification models can be formulated as in Equation (4.1) (from page 72):

$$d_{\mathrm{M}}(A,B) = d_{\mathrm{R}}^{\mathrm{I}}(A,B) = d_{\mathrm{DCJ}}(A,B) + \tau_{\mathrm{R}}^{\mathrm{I}}(A,B).$$

Apart from the theory presented in this thesis, also for some other models the offset $\tau_{\mathrm{R}}^{\mathrm{I}}$ has been determined. The notation for the formulae that follow is: c is the number of cycles, p is the number of paths, $|F|$ is the number of trees in forest F, n is the number of leaves in a tree or forest, $\lambda(C)$ is the indel-potential of cycle C, and P, Q, T, S, M, N are computed as given in [26]. Restrictions with regard to the pair of genomes are given on the right-hand side.

$$\tau_{\mathrm{rDCJ}} = 0,$$
circular intermediates are immediately incorporated [90]

$$\tau_{\mathrm{INV}} = 0$$
no bad component groups [10,55,72]

$$\tau_{\mathrm{DCJ}}^{id} = \sum_{C \in \mathcal{R}(A,B)} \lambda(C),$$
circular chromosomes [26]

$$\tau_{\mathrm{DCJ}}^{id} = \sum_{C \in \mathcal{R}(A,B)} \lambda(C) - 2P - 3Q - 2T - S - 2M - N,$$
[26]

$$\tau_{\mathrm{rDCJ}}^{id} = \tau_{\mathrm{DCJ}}^{id},$$
circular intermediates are immediately incorporated [27]

$$\tau_{\mathrm{INV}}^{id} = \tau_{\mathrm{DCJ}}^{id}, \qquad \text{i.e. } \tau_{\mathrm{INV}}^{*} = 0$$
no bad component groups [89]

$$\tau_{\mathrm{INV}} = \mathit{cost}(T) \quad \text{where } T \text{ has unlabelled bad leaves}$$
$$= \begin{cases} n+1, & n \text{ is odd and no solo } \varepsilon\text{-leaf exists} \\ n, & \mathit{otherwise.} \end{cases}$$
[10,55,72]

$$\tau_{\mathrm{SCJ}} = 2 \cdot c_{\text{non-trivial}}$$
[9]

$$\tau_{\mathrm{SC/J}} = d_{\mathrm{DCJ}}(A,B) + 2 \cdot c_{\text{non-trivial}} - p_{AA} - p_{BB},$$
[47]

$$\tau_{\mathrm{Trl}} = \mathit{cost}(\mathit{forest}\ F)$$
$$= \begin{cases} n+2, & n \text{ is even and } |F| = 1 \\ n, & n \text{ is even and } |F| \neq 1 \\ n+1, & n \text{ is odd.} \end{cases}$$
[11,68]

$$\tau_{\mathrm{HP}} = \mathit{cost}(T) \quad \text{where } T \text{ has unlabelled bad and semi-bad leaves}$$
[14,44]

The five cases require a complicated notation, and can be found in [44].

The list above shows solutions for some related or interesting models. However, there are still models for which the offsets have not been determined.

Distance Relations. It is also conceivable that in addition to the DCJ, restricted DCJ and single cut or join (SC/J) models, insertions and deletions will be allowed also for genome modification models such as the Hannenhalli-Pevzner (HP) model or sorting by translocations. For the HP model, a distance computation can be done via the computation of the DCJ distance and construction of a tree similar to the component group tree [14,44]. Here, the nodes have three different characters (black vertices are good, white vertices are bad and grey vertices are *semi-bad* and represent bad component groups that contain telomeres). For a prospective HP-*indel* model, additionally, labels need to be included, expanding the types of nodes from three, in the unlabelled case, to $3 \times 4 = 12$ combinations of characters and labels. In the tree, all semi-bad and bad vertices need to be covered by short or long paths. There are eight different types of short paths and 36 different long paths that use bad or semi-bad nodes. It needs to be investigated which tree properties influence the overall minimum cost of a tree cover, for example separations between not only the subtrees defined by labelling but also between subtrees defined by their character.

I have contributed an important building block to this interesting area and I am certainly curious about filling gaps for finding $\tau_{\mathrm{R}}^{\mathrm{I}}$, such as including different content modifications like substitutions and duplications.

Bibliography

[1] Max A. Alekseyev. Multi-break rearrangements and breakpoint re-uses: from circular to linear genomes. *Journal of Computational Biology*, 15(8):1117–1131, 2008.

[2] Oswald Theodore Avery, Colin Munro MacLeod, and Maclyn McCarty. Studies on the chemical nature of the substance inducing transformation of pneymococcal types: Induction of Transformation by a Desoxyribonucleic Acid Fraction Isolated from Pneumococcus Type III. *J Exp Med*, 79(2):137–158, Feb 1944.

[3] Martin Bader. Sorting by Reversals, Block Interchanges, Tandem Duplications, and Deletions. *BMC Bioinformatics*, 10 Suppl 1:S9, 2009.

[4] Martin Bader, Mohamed I. Abouelhoda, and Enno Ohlebusch. A fast algorithm for the multiple genome rearrangement problem with weighted reversals and transpositions. *BMC Bioinformatics*, 9:516, 2008.

[5] Vineet Bafna and Pavel A. Pevzner. Genome rearrangements and sorting by reversals. In *Proceedings of the 34th Annual Symposium of the Foundations of Computer Science, FOCS 1993*, pages 148–157, 1993.

[6] Vineet Bafna and Pavel A. Pevzner. Sorting Permutations by Transpositions. In *SODA '95: Proceedings of the 6th annual ACM-SIAM symposium on Discrete algorithms*, pages 614–623, 1995.

[7] Eugeni Belda, Andés Moya, and Francisco J. Silva. Genome Rearrangement Distances and Gene Order Phylogeny in γ-Proteobacteria. *Published by Oxford University Press on behalf of the Society for Molecular Biology and Evolution.*, 22(6):1456–1467, 2005.

[8] Anne Bergeron. A Very Elementary Presentation of the Hannenhalli-Pevzner

Theory. In *Proceedings of CPM 2001*, volume 2089, pages 106–117. Springer Verlag, 2001.

[9] Anne Bergeron, Paul Medvedev, and Jens Stoye. Rearrangement models and single-cut operations. *Journal of Computational Biology*, 17(9):1213–1225, Sep 2010.

[10] Anne Bergeron, Julia Mixtacki, and Jens Stoye. Reversal Distance without Hurdles and Fortresses. In S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, editors, *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching, CPM 2004*, volume 3109 of *LNCS*, pages 388–399, Berlin, 2004. Springer Verlag.

[11] Anne Bergeron, Julia Mixtacki, and Jens Stoye. On Sorting by Translocations. In Satoru Miyano, Jill P. Mesirov, Simon Kasif, Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Research in Computational Molecular Biology*, volume 3500 of *Lecture Notes in Computer Science*, pages 615–629. Springer, 2005.

[12] Anne Bergeron, Julia Mixtacki, and Jens Stoye. The Inversion Distance Problem. In O. Gascuel, editor, *Mathematics of Evolution and Phylogeny*, chapter 10, pages 262–290. Oxford University Press, Oxford, UK, 2005. [A preliminary version appeared in Proc. of CPM 2004 [10]].

[13] Anne Bergeron, Julia Mixtacki, and Jens Stoye. A Unifying View of Genome Rearrangements. In *Proceedings of the 6th International Conference on Algorithms in Bioinformatics, WABI 2006*, volume 4175 of *LNBI*, pages 163–173, 2006.

[14] Anne Bergeron, Julia Mixtacki, and Jens Stoye. A new linear time algorithm to compute the genomic distance via the double cut and join distance. *Theoretical Computer Science*, 410(51):5300–5316, 2009.

[15] Matthias Bernt, Kun-Mao Chao, Jyun-Wei Kao, Martin Middendorf, and Eric Tannier. Preserving Inversion Phylogeny Reconstruction. In Benjamin J. Raphael and Jijun Tang, editors, *Proceedings of the 12th International Workshop on Algorithms in Bioinformatics, WABI 2012*, volume 7534 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2012.

[16] Guillaume Blanc, Hiroyuki Ogata, Catherine Robert, Stéphane Audic, Karsten Suhre, Guy Vestris, Jean-Michel Claverie, and Didier Raoult. Reductive genome evolution from the mother of Rickettsia. *PLoS Genetics*, 3(1):e14, 2007.

[17] Frederick R. Blattner, Guy Plunkett III, Craig A. Bloch, Nicole T. Perna, Valerie Burland, Monica Riley, Julio Collado-Vides, Jeremy D. Glasner, Christopher K. Rode, George F. Mayhew, J. Gregor, Nelson Wayne Davis, Heather A. Kirkpatrick, Michael A. Goeden, Debra J. Rose, Bob Mau, and Ying Shao. The complete genome sequence of Escherichia coli K-12. *Science*, 277(5331):1453–1462, Sep 1997.

[18] Marília D. V. Braga. On Sorting Genomes with DCJ and Indels. In Eric Tannier, editor, *Comparative Genomics. RECOMB-CG 2010*, volume 6398 of *LNCS*, pages 62–73, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[19] Marília D. V. Braga. An Overview of Genomic Distances Modeled with Indels. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, volume 7921 of *LNCS*, pages 22–31. Springer, July 2013. 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, July 1-5, 2013. Proceedings.

[20] Marília D. V. Braga, Raphael Machado, Leonardo C. Ribeiro, and Jens Stoye. On the weight of indels in genomic distances. *BMC Bioinformatics*, 12 Suppl 9:13, 2011.

[21] Marília D. V. Braga, Marie-France Sagot, Celine Scornavacca, and Eric Tannier. Exploring the Solution Space of Sorting by Reversals, with Experiments and an Application to Evolution. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(3):348–356, 2008.

[22] Marília D. V. Braga and Jens Stoye. Counting all DCJ Sorting Scenarios. In *Proceedings of RECOMB-CG 2009*, volume 5817 of *LNBI*, pages 36–47, 2009.

[23] Marília D. V. Braga and Jens Stoye. The solution space of sorting by DCJ. *Journal of Computational Biology*, 17(9):1145–1165, 2010.

[24] Marília D. V. Braga and Jens Stoye. Restricted DCJ-Indel Model Revisited. In João C. Setubal and Nalvo F. Almeida, editors, *BSB*, volume 8213 of *LNBI*, pages 36–46. Springer, 2013.

[25] Marília D. V. Braga, Eyla Willing, and Jens Stoye. Genomic Distance with DCJ and Indels. In *Proceedings of the 10th International Workshop on Algorithms in Bioinformatics, WABI 2010*, volume 6293 of *LNBI*, pages 90–101, 2010.

[26] Marília D. V. Braga, Eyla Willing, and Jens Stoye. Double cut and join with insertions and deletions. *Journal of Computational Biology*, 18(9):1167–1184, Sep 2011.

[27] Marília D. V. Braga and Jens Stoye. Sorting linear genomes with rearrangements and indels. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(3):500–506, 2015. [A preliminary version appeared in [24]].

[28] Neil A. Campbell and Jane B. Reece. *Biologie*. Spektrum Akademischer Verlag, 6th edition, 2003. Original title: *Biology 6/e*.

[29] Sherwood Casjens, Nanette Palmer, René van Vugt, Wai Mun Huang, Brian Stevenson, Patricia Rosa, Raju Lathigra, Granger Sutton, Jeremy Peterson, Robert J. Dodson, Daniel Haft, Erin Hickey, Michelle Gwinn, Owen White, and Claire M. Fraser. A bacterial genome in flux: the twelve linear and nine circular extrachromosomal DNAs in an infectious isolate of the Lyme disease spirochete Borrelia burgdorferi. *Mol Microbiol*, 35(3):490–516, Feb 2000.

[30] Erwin Chargaff. Chemical specificity of nucleic acids and mechanism of their enzymatic degradation. *Experientia*, 6(6):201–209, Jun 1950.

[31] David A. Christie. Sorting Permutations by Block-Interchanges. *Information Processing Letters*, 60(4):165–169, 1996.

[32] David S. Cohen and Manuel Blum. On the problem of sorting burnt pancakes. *Discrete Applied Mathematics*, 61(2):105–120, 1995.

[33] Phillip E. C. Compeau. A Simplified View of DCJ-Indel Distance. In Ben Raphael and Jijun Tang, editors, *Proceedings of the 12th International Workshop on Algorithms in Bioinformatics, WABI 2012*, volume 7534 of *Lecture Notes in Computer Science*, pages 365–377. Springer Berlin Heidelberg, 2012.

[34] Phillip E. C. Compeau. DCJ-Indel sorting revisited. *Proceedings of the 13th International Workshop on Algorithms in Bioinformatics, WABI 2013*, 8(1):6, 2013.

[35] Poly H. da Silva, Marília D. V. Braga, Raphael Machado, and Simone Dantas. DCJ-indel Distance with Distinct Operation Costs. In Ben Raphael and Jijun Tang, editors, *Proceedings of the 12th International Workshop on Algorithms in Bioinformatics, WABI 2012*, volume 7534 of *Lecture Notes in Computer Science*, pages 378–390. Springer Berlin Heidelberg, 2012.

[36] Poly H. da Silva, Raphael Machado, Simone Dantas, and Marília D. V. Braga. Restricted DCJ-indel model: sorting linear genomes with DCJ and indels. *BMC Bioinformatics*, 13(S-19):S14, 2012.

[37] Poly H. da Silva, Raphael Machado, Simone Dantas, and Marília D. V. Braga. DCJ-indel and DCJ-substitution distances with distinct operation costs. *Algorithms for Molecular Biology*, 8:21, 2013. [A preliminary version appeared in Proc. of WABI 2012 [35]].

[38] Wen Deng, Shian-Ren Liou, Guy Plunkett III, George F. Mayhew, Debra J. Rose, Valerie Burland, Voula Kodoyianni, David C. Schwartz, and Frederick R. Blattner. Comparative genomics of Salmonella enterica serovar Typhi strains Ty2 and CT18. *Journal of Bacteriology*, 185(7):2330–2337, Apr 2003.

[39] Christian Baudet Ulisses Dias and Zanoni Dias. Sorting by weighted inversions considering length and symmetry. *BMC Bioinformatics*, 16 Suppl 19:S3, 2015.

[40] Zanoni Dias and João Meidanis. Genome Rearrangements Distance by Fusion, Fission, and Transposition Is Easy. In *Proceedings of SPIRE 2001*, pages 250–253, 2001.

[41] Theodosius G. Dobzhansky and Alfred H. Sturtevant. Inversions in the chromosomes of Drosophila pseudoobscura. *Genetics*, 23(1):28–64, 1938.

[42] Nadia El-Mabrouk. Genome Rearrangement by Reversals and Insertions / Deletions of Contiguous Segments. In *Proceedings of CPM'00*, LNCS, pages 222–234, 2000.

[43] Nadia El-Mabrouk. Sorting Signed Permutations by Reversals and Insertions/Deletions of Contiguous Segments. *Journal of Discrete Algorithms*, 1(1):105–122, 2001. [A preliminary version appeared in Proc. of CPM 2000 [42]].

[44] Péter L. Erdös, Lajos Soukup, and Jens Stoye. Balanced vertices in trees and a simpler algorithm to compute the genomic distance. *Applied Mathematics Letters*, 24(1):82–86, 2011.

[45] Pedro Feijão. Counting Suboptimal DCJ Sorting Scenarios with Distance $d + 1$ without Recombinations. Unpublished, correspondence with the author, 2016.

[46] Pedro Feijão. Single-Cut-or-Join with Insertions and Deletions. Unpublished, correspondence with the author, 2016.

[47] Pedro Feijão and João Meidanis. SCJ: a breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Trans Comput Biol Bioinform*, 8(5):1318–1329, 2011.

[48] Richard Friedberg, Aaron E. Darling, and Sophia Yancopoulos. *Genome rearrangement by the double cut and join operation*, volume 452, chapter 18, pages 385–416. Springer, January 2008.

[49] Milton H. Gallardo, John W. Bickham, Rodney L. Honeycutt, Ricardo A. Ojeda, and Nelida Köhler. Discovery of tetraploidy in a mammal. *Nature*, 401(6751):341, 1999.

[50] William H. Gates and Christos H. Papadimitriou. Bounds for sorting by prefix reversal. *Discrete Mathematics*, 27(1):47–57, 1979.

[51] Frederick Griffith. The Significance of Pneumococcal Types. *J Hyg (Lond)*, 27(2):113–159, Jan 1928.

[52] Anthony J. F. Griffiths, Susan R. Wessler, Richard C. Lewontin, William M. Gelbart, David T. Suzuki, and Jeffrey H. Miller. *An Introduction to Genetic Analysis*. W. H. Freeman and Co, New York, 8 edition.

[53] Sridhar Hannenhalli and Pavel A. Pevzner. Transforming Cabbage into Turnip (polynomial algorithm for sorting signed permutations by reversals). In *Journal of the ACM*, STOC '95, pages 178–189. ACM Press, 1995.

[54] Sridhar Hannenhalli and Pavel A. Pevzner. Transforming Men Into Mice (Polynomial Algorithm for Genomic Distance Problem). In *Proceedings of the 36th Annual Symposium of the Foundations of Computer Science, FOCS 1995*, pages 581–592. IEEE Press, 1995.

[55] Sridhar Hannenhalli and Pavel A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46:1–27, January 1999. [A preliminary version appeared in Proc. of STOC 1995].

[56] István Hargittai. The tetranucleotide hypothesis: a centennial. *Structural Chemistry*, 20(5):753–756, 2009.

[57] Tetsuya Hayashi, Kozo Makino, Makoto Ohnishi, Ken Kurokawa, Kazuo Ishii, Katsushi Yokoyama, Chang-Gyun Han, Eiichi Ohtsubo, Keisuke Nakayama,

Takahiro Murata, Masashi Tanaka, Toru Tobe, Tetsuya Iida, Hideto Takami, Takeshi Honda, Chihiro Sasakawa, Naotake Ogasawara, Teruo Yasunaga, Satoru Kuhara, Tadayoshi Shiba, Masahira Hattori, and Hideo Shinagawa. Complete Genome Sequence of Enterohemorrhagic Eschelichia coli O157:H7 and Genomic Comparison with a Laboratory Strain K-12. *DNA Research*, 8(1):11–22, Feb 2001.

[58] Alfred Day Hershey and Martha Chase. Independent functions of viral protein and nucleic acid in growth of bacteriophage. *Journal of General Physiology*, 36(1):39–56, May 1952.

[59] Rolf Hilker, Corinna Sickinger, Christian N. S. Pedersen, and Jens Stoye. UniMoG – a unifying framework for genomic distance calculation and sorting based on DCJ. *Bioinformatics*, 28(19):2509–2511, 2012.

[60] Peter Husemann and Jens Stoye. r2cat: synteny plots and comparative assembly. *Bioinformatics*, 26(4):570–571, Feb 2010. Available at: `http://bibiserv.techfak.uni-bielefeld.de/cgcat?id=cgcat_r2cat`.

[61] Géraldine Jean and Macha Nikolski. Genome rearrangements: a correct algorithm for optimal capping. *Information Processing Letters*, 104(1):14–20, 2007.

[62] Crystal L. Kahn, Shay Mozes, and Benjamin J. Raphael. Efficient algorithms for analyzing segmental duplications with deletions and inversions in genomes. *Algorithms Mol Biol*, 5(1):11, Jan 2010.

[63] John D. Kececioglu and David Sankoff. Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. *Algorithmica*, 13(1/2):180–210, 1995.

[64] Jakub Kováč, Robert Warren, Marília D. V. Braga, and Jens Stoye. Restricted DCJ model: rearrangement problems with chromosome reincorporation. *Journal of Computational Biology*, 18(9):1231–1241, Sep 2011.

[65] Phoebus Aaron Levene. The Structure of Yeast Nucleic Acid: IV. Ammonia Hydrolysis. *Journal of Biological Chemistry*, 40(2):415–424, 1919.

[66] Phoebus Aaron Levene and W. A. Jacobs. On Glycothionic Acid. *J Exp Med*, 10(4):557–558, Jul 1908.

[67] Zimao Li, Lusheng Wang, and Kaizhong Zhang. Algorithmic Approaches for Genome Rearrangement: A Review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 36(5):636–648, 2006.

[68] Julia Zakotnik geb. Mixtacki. *The Double Cut and Join Operation and its Application to Genome Rearrangements*. PhD thesis, Technische Fakultät, Universität Bielefeld, 2008.

[69] Michael McClelland, Kenneth E. Sanderson, John Spieth, Sandra W. Clifton, Phil Latreille, Laura Courtney, Steffen Porwollik, Johar Ali, Mike Dante, Feiyu Du, Shunfang Hou, Dan Layman, Shawn Leonard, Christine Nguyen, Kelsi Scott, Andrea Holmes, Neenu Grewal, Elizabeth Mulvaney, Ellen Ryan, Hui Sun, Liliana Florea, Webb Miller, Tamberlyn Stoneking, Michael Nhan, Robert Waterston, and Richard K. Wilson. Complete genome sequence of Salmonella enterica serovar Typhimurium LT2. *Nature*, 413(6858):852–856, Oct 2001.

[70] Michael P. McLeod, Xiang Qin, Sandor E. Karpathy, Jason Gioia, Sarah K. Highlander, George E. Fox, Thomas Z. McNeill, Huaiyang Jiang, Donna Muzny, Leni S. Jacob, Alicia C. Hawes, Erica Sodergren, Rachel Gill, Jennifer Hume, Maggie Morgan, Guangwei Fan, Anita G. Amin, Richard A. Gibbs, Chao Hong, Xue-Jie Yu, David H. Walker, and George M. Weinstock. Complete genome sequence of Rickettsia typhi and comparison with sequences of other rickettsiae. *J Bacteriol*, 186(17):5842–5855, Sep 2004.

[71] João Meidanis, Maria Emília Machado Telles Walter, and Zanoni Dias. Reversal distance of signed circular chromosomes, March 1997.

[72] João Meidanis, Maria Emília Machado Telles Walter, and Zanoni Dias. Reversal distance of signed circular chromosomes. Technical Report IC-00-23, December 2000.

[73] Friedrich Miescher. *Ueber die chemische Zusammensetzung der Eiterzellen*, chapter XLV, pages 441–460. Number 4 in Medicinisch-chemische Untersuchungen. Felix Hoppe-Seyler, 1871.

[74] István Miklós. MCMC genome rearrangement. *Bioinformatics*, 19 Suppl 2:ii130–ii137, 2003.

[75] István Miklós and Eric Tannier. Bayesian sampling of genomic rearrangement scenarios via double cut and join. *Bioinformatics*, 26:3012–3019, December 2010.

[76] István Miklós and Eric Tannier. Approximating the number of Double Cut-and-Join scenarios. *Theor. Comput. Sci.*, 439:30–40, June 2012.

[77] Aïda Ouangraoua and Anne Bergeron. Combinatorial structure of genome rearrangements scenarios. *Journal of Computational Biology*, 17(9):1129–1144, 2010.

[78] Julian Parkhill, Gordon Dougan, Keith D. James, Nicolas R. Thomson, Derek Pickard, John Wain, Carol Churcher, Karen L. Mungall, Stephen D. Bentley, Matthew T. Holden, Mohammed Sebaihia, Stehpen Baker, D. Basham, K. Brooks, Tracey Chillingworth, Phillippa Connerton, Ann Cronin, Paul Davis, R. M. Davies, Linda Dowd, Nicolas White, Jeremy Farrar, Theresa Feltwell, Nancy Hamlin, A. Haque, Tran Tinh Hien, Simon Holroyd, Kay Jagels, Anders Krogh, T. S. Larsen, S. Leather, Sharon Moule, Peadar Ó'Gaora, Christopher Parry, Michael Quail, K. Rutherford, Mark Simmonds, Jason Skelton, Kim Stevens, Sally Whitehead, and Bart G. Barrell. Complete genome sequence of a multiple drug resistant Salmonella enterica serovar Typhi CT18. *Nature*, 413(6858):848–852, Oct 2001.

[79] Sophie Pasek, Jean-Loup Risler, and Pierre Brézellec. Gene fusion/fission is a major contributor to evolution of multi-domain bacterial proteins. *Bioinformatics*, 22(12):1418–1423, Jun 2006.

[80] Nicole T. Perna, Guy Plunkett III, Valerie Burland, Bob Mau, Jeremy D. Glasner, Debra J. Rose, George F. Mayhew, Peter S. Evans, Jason Gregor, Heather A. Kirkpatrick, György Pósfai, Jeremiah Hackett, Sara Klink, Adam Boutin, Ying Shao, Leslie Miller, Erik J. Grotbeck, Nelson Wayne Davis, Alex Lim, Eileen T. Dimalanta, Konstantinos D. Potamousis, Jennifer Apodaca, Thomas S. Anantharaman, Jieyi Lin, Galex Yen, David C. Schwartz, Rodney A. Welch, and Frederick R. Blattner. Genome sequence of enterohaemorrhagic Escherichia coli O157:H7. *Nature*, 409(6819):529–533, Jan 2001.

[81] Pavel A. Pevzner and Glenn Tesler. Transforming Men into Mice: The Nadeau-Taylor Chromosomal Breakage Model Revisited. In *Proceedings of RECOMB 2003*, pages 247–256. ACM Press, 2003.

[82] João Carlos Setubal and João Meidanis. *Introduction to computational molecular biology*. PWS Publishing Company, 1997.

[83] Günter Staudt. Die Entstehung und Geschichte der großfrüchtigen Gartenerdbeeren *Fragaria* × ananassa Duch. *Theoretical and Applied Genetics*, 31(5):212–218, 1961.

[84] Jean-Nicolas Volff and Josef Altenbuchner. A new beginning with new ends: linearisation of circular chromosomes during bacterial evolution. *FEMS Microbiol Lett*, 186(2):143–150, May 2000.

[85] James Dewey Watson and Francis Harry Compton Crick. Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, Apr 1953.

[86] Rodney A. Welch, Valerie Burland, Guy Plunkett III, Peter Redford, Paula Roesch, David Rasko, Eric L. Buckles, Shian-Ren Liou, Adam Boutin, Jeremiah Hackett, David Stroud, George F. Mayhew, Debra J. Rose, Shiguo Zhou, David C. Schwartz, Nicole T. Perna, Harry L. T. Mobley, Michael S. Donnenberg, and Frederick R. Blattner. Extensive mosaic structure revealed by the complete genome sequence of uropathogenic Escherichia coli. volume 99, pages 17020–17024. National Acad Sciences, 2002.

[87] Eyla Willing. The Double Cut and Join Model with Insertions and Deletions. Master's thesis, Universität Bielefeld, 2010.

[88] Eyla Willing and Jens Stoye. Uniform Sampling of DCJ Sorting Scenarios. Poster Abstract, Poster, 2012. Conference contribution to *RECOMB-CG*.

[89] Eyla Willing, Simone Zaccaria, Marília D. V. Braga, and Jens Stoye. On the Inversion-Indel Distance. *BMC Bioinformatics*, 14(15):S3, 2013.

[90] Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by Translocation, Inversion and Block Interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

[91] Sophia Yancopoulos and Richard Friedberg. Sorting Genomes with Insertions, Deletions and Duplications by DCJ. In *Proceedings of RECOMB-CG 2008*, volume 5267 of *LNBI*, pages 170–183, 2008.

[92] Sophia Yancopoulos and Richard Friedberg. DCJ path formulation for genome transformations which include Insertions, Deletions, and Duplications. *Journal of Computational Biology*, 16(10):1311–1338, 2009.

Notation

$\circ$	Cap-symbol denoting a telomere		4
$\cdot\|_{\mathcal{G}}$	Core genome of corresponding genomes	$A\|_{\mathcal{G}}$, $B\|_{\mathcal{G}}$	17
$\mathcal{A}$	Set of unique markers of genome A	$\mathcal{A} = \mathcal{G}_A \setminus \mathcal{G}$	16
AA-path	AG-path that has both ends in genome A		24
AB-path	AG-path that has one end in both genomes		24
$\mathcal{B}$	Set of unique markers of genome B	$\mathcal{B} = \mathcal{G}_B \setminus \mathcal{G}$	16
BB-path	AG-path that has both ends in genome B		24
C	Cover, set of covering paths for $T_\circ$		116
C_r	Cover, set of covering paths for T_r		116
$d_{\mathrm{M}}, d_{\mathrm{R}}^{\mathrm{I}}$	Distance for a model M, e.g. DCJ, INV or other sets of allowed operations	$\mathrm{M} = \mathrm{R} \cup \mathrm{I}$	72, 166
$\mathcal{G}$	Set of markers common to genomes A and B	$\mathcal{G} = \mathcal{G}_A \cap \mathcal{G}_B$	16
$\mathcal{G}_A, \mathcal{G}_B$	Set of all markers from genome A (resp. B)		5
$\mathcal{I}$	Set of all content modifying operations		11, 15
I	Selection of content modifying operations	$\mathrm{I} \subseteq \mathcal{I}$	11, 15
$\mathcal{K}$	Component group, i.e. set of interleaving cycles of $R(A, B)$		92, 93
$\mathcal{L}$	Leaf composition of $T_\circ(A, B)$	$\mathcal{L}(A, B)$	114
Λ	Number of runs in a cycle C	$\Lambda(C)$	74
$\lambda, \lambda(C)$	Indel-potential of a labelled cycle C	$\lambda = \lceil \Lambda(C)/2 \rceil$	76
$\mathcal{L}_r$	Residual leaf composition of $T_r(A, B)$	$\mathcal{L}_r(A, B)$	135

MG	Mastergraph	$\mathrm{MG}(A, B)$	24
$\mathcal{M}$	Set of all genome rearrangement models	$\mathcal{M} = \mathcal{R} \times \mathcal{I}$	11
M	Specific genome modification model with a specific set of allowed operations	$\mathrm{M} = \mathrm{R} \times \mathrm{I}$	11
$\mathcal{R}$	Relational diagram of genomes A and B	$\mathcal{R}(A, B)$	24
ρ	Specific rearrangement operation ρ		42
$\mathcal{R}$	Set of all rearranging operations		11
R	Specific selection of rearranging operations	$\mathrm{R} \subseteq \mathcal{R}$	11
$T_\circ$	Bad labelled component group tree	$T_\circ(A, B)$	110, 114
T_r	Residual tree, derived from $T_\circ$	$T_r(A, B)$	135
$\mathcal{X}$	Types of labellings from $\{\mathcal{A}, \mathcal{B}, \varepsilon, \mathcal{AB}\}$		77, 119, 127
X	Adjacency graph component of arbitrary type (cycle, AA-, AB- or BB-path).		30
$\mathcal{Y}$	Types of labellings from $\{\mathcal{A}, \mathcal{B}, \varepsilon, \mathcal{AB}\}$		119
$\mathcal{D}$	Set of all distances in the adjacency graph		37
j	Split group identifier		40, 43
s_d^j	Number of sorting scenarios for the representative of $\mathcal{U}_d$ of split group j		43
s	Number of sorting scenarios for $\mathcal{U}$		36
$\mathcal{U}$	Set of all unsorted components		36
$\mathcal{U}_d$	Set of all components with distance d	$\mathcal{U}_d \subseteq \mathcal{U}$	37
w_d^j	Weight of a bucket, (representing split group j with distance d)		43, 49

The truth is rarely pure and never simple.

Oscar Wilde

Appendix

A Additional Examples

In this section we give examples that serve as a supplementary demonstration of concepts or examples given in Chapter 6.

A.1 Unsafe Covering Paths

As Example 18 (see page 114) already shows, different properties of a tree influence the optimal cost of the cover. Hence, a tree $T_\circ(A, B)$ should be reduced such that these properties are present in $T_r(A, B)$, otherwise the reduction step would not be safe.

If a new bad leaf emerges from such a procedure, the overall cost of the cover may be increased, such a path is therefore not safe. This happens if we choose paths carelessly. Figure A.1 shows a tree that has two ε-leaves and four $\mathcal{A}$-leaves. The two subtrees are non-separated. Covering ℓ_2 and ℓ_3 with a long path transforms all nodes of that path into a single good node that is labelled. This new node is then, in the process of turning the tree into a bad labelled component group tree again, removed (and the labels pushed inwards). It remains ℓ_4, a new leaf that is bad. If ℓ_4 would be a good

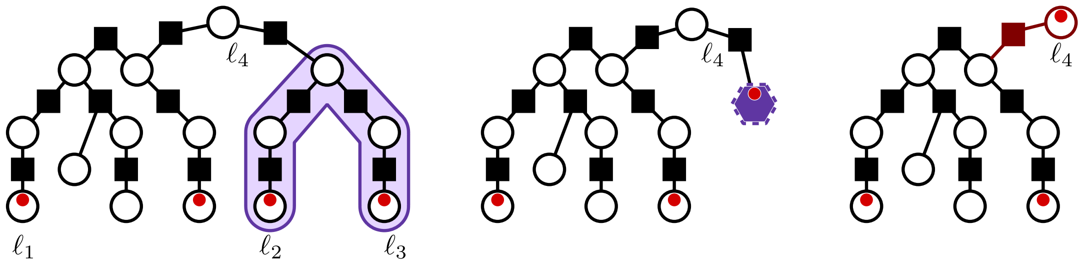


Figure A.1: Choosing leaves for homogeneous paths carelessly may result in a new bad leaf ℓ_4. The covering path is not safe.

vertex, the covering path of ℓ_2 and ℓ_3 would still become a leaf, but is erased while transforming the tree into a bad labelled component group tree again (and therefore it is safe). Furthermore, if at the branching vertex of ℓ_2 and ℓ_3 there would be another sibling, a covering path of these would also be safe, as it does not create a new bad leaf. An alternative cover is to cover ℓ_2 by a long path along with ℓ_1 instead. The covering path would not produce a new bad leaf and is therefore safe.

A.2 Non-Separated Labelled Tree

Figure A.2 (i) shows a bad component group tree with leaf composition $\mathcal{L} = (3, 0, 2, 0)$ and no separating vertex exists between the two subtrees. When further reducing the $\mathscr{A}$-subtree by covering ℓ_2 and ℓ_3 with a long homogeneous path (depicted in tree (ii)), the $\mathscr{A}$-subtree is reduced to a single leaf and the bad nodes on the branch of ℓ_1 become

(i) A tree without separation that has three $\mathscr{A}$-leaves. **(ii)** Covering two $\mathscr{A}$-leaves with a long homogeneous path. **(iii)** The covered vertices are merged into a single *meta*-vertex.

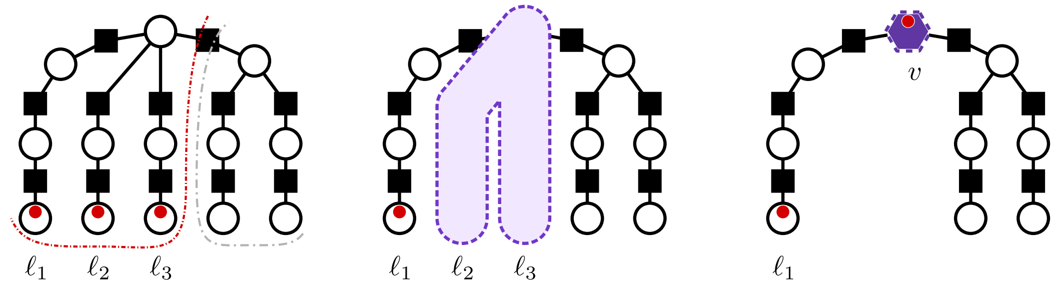


Figure A.2: A bad component group tree with two non-separated subtrees. The separation of the labelled subtree is compensated for.

separating vertices.

Fortunately, with the reduction it is guaranteed that the newly formed component group which is good is also labelled. Figure A.2 (iii) depicts this with a meta vertex v whose label serves as a link. It is hence always possible for all separating vertices, that emerged by the reduction of labelled subtrees, to be covered at the same cost with a long homogeneous path as it would cost to treat the leaf without separation.

A.3 Separations

For space reasons only a few examples were given in Subsection 6.2.1 and we seek to elaborate the individual and combined separations with some more examples. In all pictures the triangles correspond to a subtree that, if coloured, represents a certain leaf type. A sequence of separating vertices (of arbitrary length) is drawn with a dotted line that has a separating vertex. If these paths are black and solid they contain no bad vertices.

Separation of Subtrees

If we generically consider subtrees (whether or not they represent combined subtrees or single subtrees), separations of different kinds are possible. Figure A.3 shows trees with the maximum of (individual) separations.

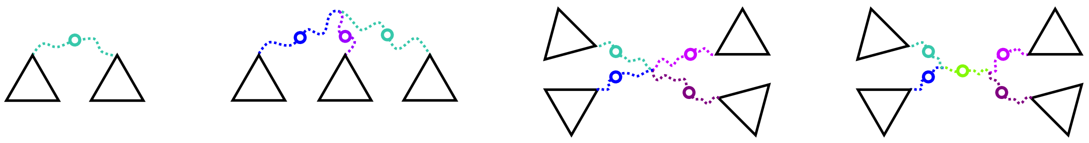


Figure A.3: A maximum of individual separations for different numbers of leaf types. Each triangle represents a subtree of a single type, and all trees have full separations. For the rightmost tree additionally two pairs of subtrees are separated.

Adding another subtree to trees (i) or (ii) can be done in several locations leading to different shapes of the extended tree. Including trees (iii) and (iv) from Figure A.3 the other shapes are depicted in Figure A.4.

(i) Three subtrees. One is non-separated. **(ii)** Four subtrees. One is non-separated. **(iii)** Four subtrees. Two are non-separated. **(iv)** Four subtrees. One is non-separated.

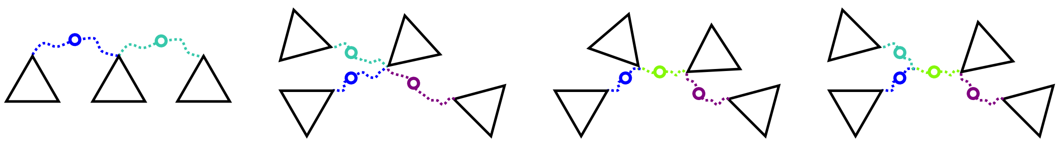


Figure A.4: Trees that have three or four subtrees of which not all are separated.

Separation of Types of Subtrees

Now we become more specific by labelling the subtrees. Figures A.5 and A.6 focus on subtrees that are non-separated. First we show three examples of trees where the $\mathcal{A}$- and $\mathcal{B}$-subtree are not separated by separating vertices but may still not share edges or even nodes.

(i) The $\mathcal{A}$- and $\mathcal{B}$-subtrees overlap. **(ii)** The $\mathcal{A}$- and $\mathcal{B}$-subtrees are "separated" only by *good* vertices. **(iii)** The separation ends in the $\mathcal{B}$-subtree.

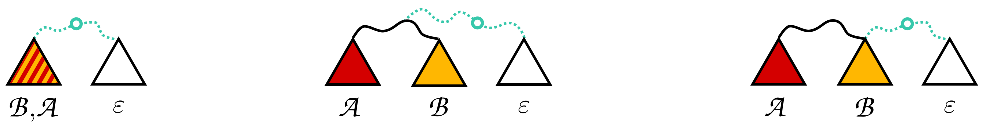


Figure A.5: Different trees whose ε- and $\{\mathcal{A}, \mathcal{B}\}$-subtrees are separated.

In Figure A.6 all three trees show $\mathcal{A}$- and ε-trees that are both individually separated. For each individual separation, at the same time the combined tree that is composed of the other leaf types is separated. Here, this means in each instance the $\mathcal{A}$-subtree is separated from the combined $\{\mathcal{B}, \varepsilon\}$-subtree and the ε-subtree is separated from the combined $\{\mathcal{A}, \mathcal{B}\}$-subtree. However, since the $\mathcal{B}$-subtree is not separated in any of the three trees, neither is the combined $\{\mathcal{A}, \varepsilon\}$-subtree separated.

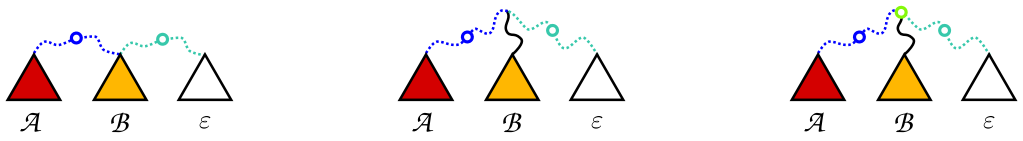


Figure A.6: The ε- and the $\mathcal{A}$-subtrees are separated, while the $\mathcal{B}$-subtrees (and thus the combined $\{\mathcal{A}, \varepsilon\}$-subtrees) are not.

We proceed with examples of four leaf types. In all four trees depicted schematically in Figure A.7 the combined $\{\mathcal{A}, \varepsilon\}$-subtree is separated from the combined $\{\mathcal{B}, \mathcal{AB}\}$-subtree. While in tree (i), $T_\circ^{\mathcal{A}}$ and $T_\circ^{\varepsilon}$ are not separated individually, in trees (ii) and (iii) additionally the ε-subtree is separated individually. In tree (iv) all subtrees are separated from another. For simplicity, an overview of all possible separations and non-separations of the trees in Figure A.7 is given in Table A.2.

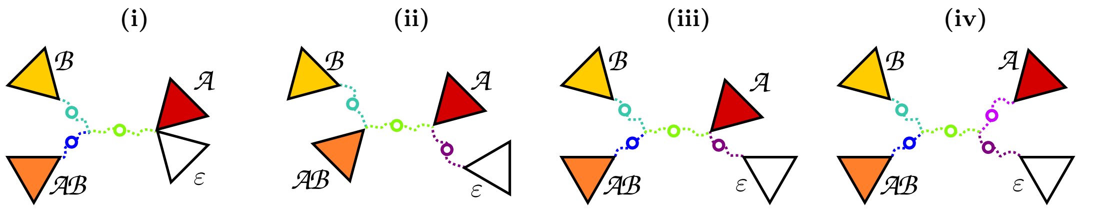


Figure A.7: Schematic display of some examples for four subtrees and their interrelation. Which subtrees or combined subtrees are separated or not is listed in Table A.2.

Table A.2: A list of separations and non-separations of the subtrees in each of the trees of Figure A.7. Subtrees or combined subtrees that are directly attached to the counterpart marked by a ✔. Otherwise, they are separated and marked by an ✗.

	non-separated			
	(i)	(ii)	(iii)	(iv)
$\mathcal{A}$ vs. $\{\mathcal{B}, \varepsilon, \mathcal{AB}\}$	✔	✔	✔	✗
$\mathcal{B}$ vs. $\{\mathcal{A}, \varepsilon, \mathcal{AB}\}$	✗	✗	✗	✗
ε vs. $\{\mathcal{A}, \mathcal{B}, \mathcal{AB}\}$	✔	✗	✗	✗
$\mathcal{AB}$ vs. $\{\mathcal{A}, \mathcal{B}, \varepsilon\}$	✗	✔	✗	✗

	non-separated			
	(i)	(ii)	(iii)	(iv)
$\{\mathcal{A}, \varepsilon\}$ vs. $\{\mathcal{B}, \mathcal{AB}\}$	✗	✗	✗	✗
$\{\mathcal{B}, \varepsilon\}$ vs. $\{\mathcal{A}, \mathcal{AB}\}$	✔	✔	✔	✔
$\{\varepsilon, \mathcal{AB}\}$ vs. $\{\mathcal{A}, \mathcal{B}\}$	✔	✔	✔	✔

A.4 Short and Long ε-Branches

Although one might detect more than one ε-leaf on a branch to which the definition of short branch applies, it is not always optimal to use all instances that were detected in the beginning. Figure A.8 shows several examples of ε-leaves with different branch lengths.

(i) All ε-leaves are on long branches.

(ii) Two ε-leaves are on long branches.

(iii) No ε-leaf is on a long branch, yet not all can be covered by short paths.

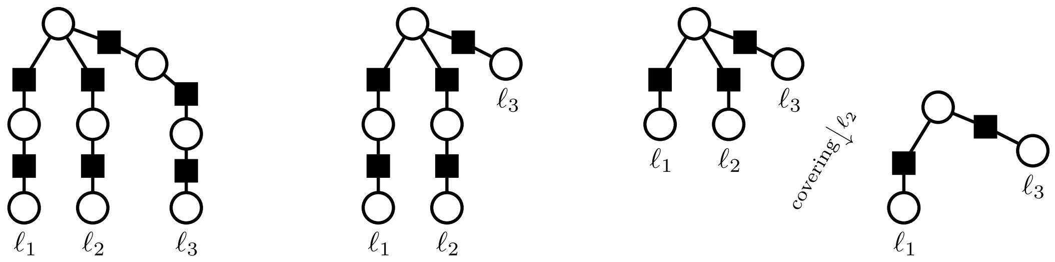


Figure A.8: Three examples of a tree with three ε-leaves on branches with various lengths.

Obviously, tree (i) can be covered only with two long homogeneous ε-paths, yielding cost 4 in total. Tree (ii) has one ε-leaf on a short branch. Covering ℓ_3 with a short path and using a long homogeneous path for ℓ_1 and ℓ_2 adds up to cost 3 and is optimal and safe. In tree (iii) all three leaves fit the definition of ε-leaves on short branches, however, once we cover ℓ_2 (or any other leaf for that matter) by a short path, the rest of the tree loses its trifurcation and ℓ_1 and ℓ_3 must now be covered by a long path that includes all three bad nodes. The total cost for this cover is 3 which is optimal and safe. If we would cover all leaves with short paths separately, a new leaf (the internal bad vertex) would arise and the total cost would amount to 4, which is neither optimal, nor safe. Thus only one of the three leaves is safe to be covered by a short path. Only if in the latter case the branching node is a good round node or a square node, a cover could consist of only short paths.

A.5 Counter-Example for Reduction of $\mathcal{AB}$-Leaves

In Figure 6.13 (on page 133) we studied a tree with leaf composition $\mathcal{L}(A, B) = (2, 2, 0, 4)$ and compared it to the same tree with only two leaves in the $\mathcal{AB}$-subtree.

Now we study an example that has all four types of leaves. In both trees of Figure A.9 the $\mathcal{A}$-, $\mathcal{B}$- and ε-subtrees as well as the separation of subtrees is the same and no links are present, merely the $\mathcal{AB}$-subtree has a single leaf in the left tree and three leaves in the tree depicted on the right.

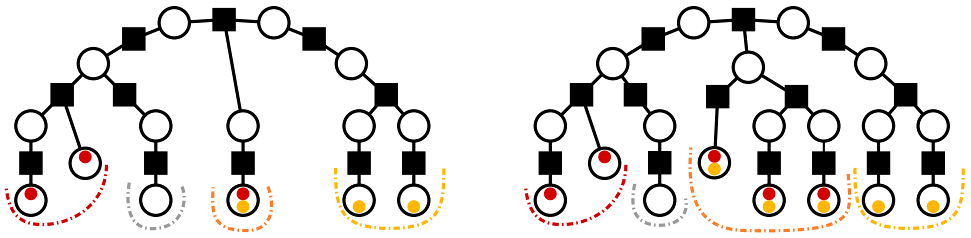


Figure A.9: Left: A tree with $\mathcal{L}(A, B) = (2, 2, 1, 1)$ whose optimal cover has cost 5 and can be achieved by using an $\mathcal{AB}$-$\mathcal{A}$-path, an $\mathcal{A}$-$\mathcal{B}$-path and a $\mathcal{B}$-ε-path. Right: A tree with the same structure, but with $\mathcal{L}(A, B) = (2, 2, 1, 3)$. Here, an optimal cover has cost 5 and can be achieved by using two $\mathcal{A}$-$\mathcal{AB}$-paths, an $\mathcal{AB}$-$\mathcal{B}$-path and a $\mathcal{B}$-ε-path.

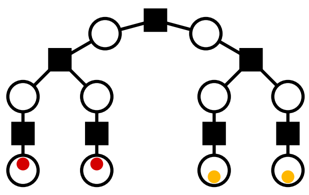
There are several co-optimal covers for the left tree that have cost 5. Obviously, using a homogeneous $\mathcal{AB}$-path in the tree on the right-hand side has cost 1 and yields the tree on the left that has cover cost 5, resulting in overall cost of 6. An alternative and optimal cover for the tree on the right uses two $\mathcal{A}$-$\mathcal{AB}$-paths, an $\mathcal{AB}$-$\mathcal{B}$-path and a $\mathcal{B}$-ε-path, yielding cost 5 in total.

Thus reducing the $\mathcal{AB}$-subtree to a single leaf is not optimal.

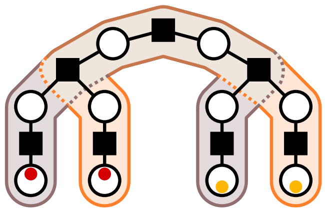
A.6 The Number of Separating Vertices Matters

The following pictures (Figures A.10 and A.11) for trees with leaf compositions $\mathcal{L} = (2, 2, 0, 0)$ and $\mathcal{L} = (2, 0, 2, 0)$ show that a strategy that yields an optimal tree cover in one case does not produce an optimal cover in another (and vice versa).

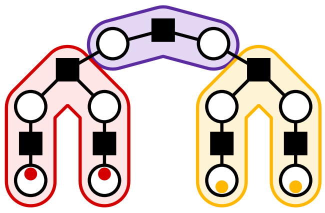
(i) More than one separating vertex.



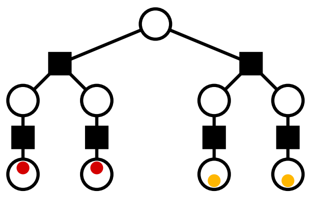
(i-a) A cover of two heterogeneous paths costs 4. ✔



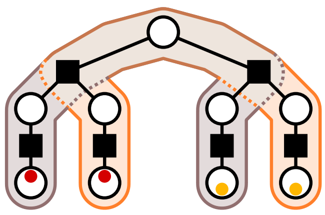
(i-b) Co-optimal cover using homogeneous paths. ✔



(ii) A single separating vertex.



(ii-a) Heterogeneous paths yield cost 4. ✗



(ii-b) An optimal cover of homogeneous paths (cost 3). ✔

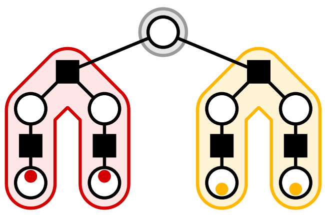
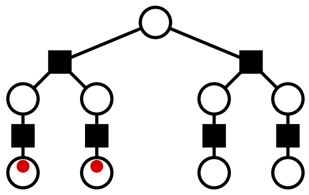
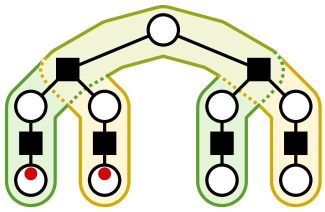


Figure A.10: Bad component group trees with leaf compositions $\mathcal{L} = (2, 2, 0, 0)$ and different homo- and heterogeneous paths.

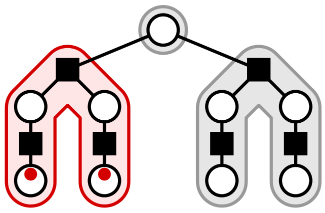
(i) A single separating vertex.



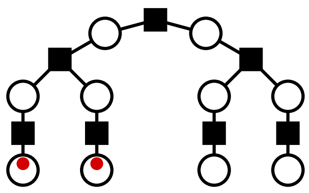
(i-a) A cover with two heterogeneous paths costs 4. ✔



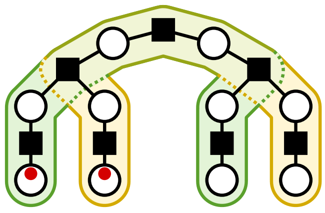
(i-b) A co-optimal cover uses two homogeneous paths. ✔



(ii) More than one separating vertex.



(ii-a) A cover with two heterogeneous paths costs 4. ✔



(ii-b) A cover with two homogeneous paths costs 5. ✗

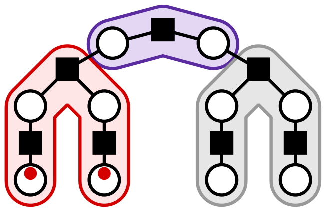


Figure A.11: Bad component group trees with leaf compositions $\mathcal{L} = (2, 0, 2, 0)$ and different homo- and heterogeneous paths.

B Bounds to an Optimal Tree Cover With Restraints

The two propositions presented here represent only a small fraction of special cases that could be studied for lower and upper bounds. When we have no separating vertex we can lower the upper bound, as the following propositions show.

Proposition 19: *Given a bad component tree $T_\circ(A,B)$ with leaf composition $\mathcal{L}(A,B) = (n_{\mathcal{A}}, n_{\mathcal{B}}, n_{\varepsilon}, 0)$ that has no separating vertex and for which n_ε is even, then we have:*

$$cost(T_\circ) = \left\lceil \frac{n_{\mathcal{A}}}{2} \right\rceil + \left\lceil \frac{n_{\mathcal{B}}}{2} \right\rceil + n_\varepsilon$$

Proof. There are no separating vertices which need to be covered. Therefore we use long homogeneous paths for covering each subtree. When $n_{\mathcal{A}}$ is odd, we use a long homogeneous path connecting the last $\mathcal{A}$-leaf with a previously used $\mathcal{A}$-leaf, as any alternative would yield the same or higher cost (similarly for the $\mathcal{B}$-subtree). It is optimal to cover an ε-subtree with even n_ε with only long homogeneous paths [55]. □

Proposition 20: *Given a bad component group tree $T_\circ(A,B)$ with leaf composition $\mathcal{L}(A,B)= (n_{\mathcal{A}}, n_{\mathcal{B}}, n_\varepsilon, n_{\mathcal{AB}})$ that has* no *separating vertex, then we have:*

$$cost(T_\circ) \leq \left\lceil \frac{n_{\mathcal{A}}}{2} \right\rceil + \left\lceil \frac{n_{\mathcal{B}}}{2} \right\rceil + (n_\varepsilon + 1) + \left\lceil \frac{n_{\mathcal{AB}}}{2} \right\rceil . \quad (1)$$

Proof. There are no separating vertices which need to be covered, therefore, covering all subtrees individually covers all bad vertices. Using long homogeneous safe paths for covering each subtree of type $\mathcal{X} \in \{\mathcal{A}, \mathcal{B}, \varepsilon, \mathcal{AB}\}$ yields the individual cost $\lfloor \frac{n_{\mathcal{X}}}{2} \rfloor \cdot cost(\mathcal{X})$ and at most one leaf in each subtree remains (this can be on a long or on a short branch). We can have two cases: (1) at most one value among $n_{\mathcal{A}}$, $n_{\mathcal{B}}$, n_ε and $n_{\mathcal{AB}}$ is odd, or (2) more than one of the subtrees has an odd number of leaves. In case (1) , the cost amounts to $cost(T_\circ) = \lceil \frac{n_{\mathcal{A}}}{2} \rceil + \lceil \frac{n_{\mathcal{B}}}{2} \rceil + cost(T_\circ^\varepsilon) + \lceil \frac{n_{\mathcal{AB}}}{2} \rceil$ and we cannot obtain lower cost by using heterogeneous paths. In case (2) it may be necessary to use heterogeneous paths in order to achieve the optimal cost. For each subtree with an odd number of leaves, a single leaf remains uncovered. If it is on a long branch, we either use a long homogeneous path (re-using a leaf) or a long heterogeneous path connecting two subtrees that have an odd number of leaves. Any combination of such paths is always with cost at most Inequality (1). Rather, if we use heterogeneous paths that share the same label, we can achieve a lower cost. If the last odd leaf of a tree is on a short branch, it may be covered by a short path that always has cost 1 and that does not violate Inequality (1). We can always achieve a cover that amounts to at most the cost given by Inequality (1) and at the same time cover all bad vertices. □

C Cover Descriptions

In this section we provide optimal covers and covering costs for trees that have two or more types of leaves. Prior to determining an optimal cover, we use reduction to obtain a residual tree. The ε-subtree has at most three and the $\mathcal{AB}$-subtree at most four leaves in a residual tree. The number of leaves in the $\mathcal{A}$- and $\mathcal{B}$-subtrees is limited to two, for which, without loss of generality, we assume $r_{\mathcal{A}} \geq r_{\mathcal{B}}$.

C.1 Nomenclature

All optimal covers presented are kept as extensive as necessary but as simple as possible. Apart from the tuple $(r_{\mathcal{A}}, r_{\mathcal{B}}, r_{\varepsilon}, r_{\mathcal{AB}})$ that is the residual leaf composition, we describe the nomenclature of the cases and covers that follow by means of some examples.

(2,1,[3],4): The [ ] enclose the leaf type whose number can be reduced. Here, the ε subtree that has $r_{\varepsilon} = 3$ leaves can be reduced to a subtree that has only a single ε-leaf. At the same time, the $\mathcal{AB}$-subtree that has four leaves cannot be reduced. The resulting base composition is then $(2, 1, 1, 4)$.

solo ε: A leaf in the ε-subtree can be covered optimally by a short path (or the whole ε-subtree, if it consists of a single vertex and is not separated from the rest of the tree).

ε not sep: There is at least one other subtree from which the ε-subtree is not separated. Similar for $\mathcal{A}$-, $\mathcal{B}$- and $\mathcal{AB}$-subtrees.

$\mathcal{A} \multimap \mathcal{B}$: There is an $\mathcal{A}$-$\mathcal{B}$-link, see Subsection 6.2.2 on page 121. $* \multimap \varepsilon$ means any label that serves as a link to the ε-subtree.

W,S,L,I: Identifiers of the covers, for example W (worst case), L (link), S (solo ε-leaf is used or, if $r_{\varepsilon} = 1$, ε-subtree is not separated), I/II (other optimal covers), SLb (combination of the above).

REDU: Reduction is possible for this set of properties. The cover in this case consists of the homogeneous path(s) specified and the set of paths given in the further reduced leaf composition for the same further tree properties.

$\mathcal{AB}-\mathcal{AB}$: A covering path using two $\mathcal{AB}$-leaves that have not been used before. Here, the covering path can use either both $\mathcal{A}$-labels or both $\mathcal{B}$-labels.

$\mathcal{A}-\mathcal{A}\mathcal{B}$: Here, the $\mathcal{A}$-label of the $\mathcal{AB}$-leaf must be used.

$\mathcal{A}-\underline{\mathcal{AB}}$: Underlining a label indicates that the respective node(s) has (have) been used previously. After each covering path (each merge) the tree is converted yet again into a bad labelled component group tree, and the labels of the now good leaves get pushed inward. This meta-vertex corresponds then to the “re-used leaf”.

$\mathcal{A}$–$\underline{\mathcal{A}/\mathcal{AB}}$: A covering path using an $\mathcal{A}$-leaf and either an $\mathcal{AB}$-leaf or an $\mathcal{A}$-leaf (both would be optimal and were used before).

$\mathcal{AB}$–$\mathcal{A}^*$: A covering path using an $\mathcal{AB}$-leaf (effectively its $\mathcal{A}$-label) with an $\mathcal{A}$-link. Where the $\mathcal{A}$-vertex is situated is specified in the case analysis, for example it could be $\mathcal{A}\multimap\mathcal{B}$ or $\mathcal{A}\multimap\varepsilon$ or $\mathcal{A}\multimap\{\mathcal{B},\varepsilon\}$.

$\cancel{\mathcal{A}\multimap\mathcal{B}}$: This property must be absent.

| 2: The cost of the cover is given to the right of the vertical bar.

| +2: A “+” means that the cost are added to those of the reduced leaf composition.

One has to be careful with $r_{\mathcal{A}}, r_{\mathcal{B}}$ or $r_{\mathcal{AB}} = 1$. Some covers derived this through reduction. If before, the subtree was not separated, but all branches were on a long branch, it will have a single leaf separated by the branch. However, as stated earlier, the new tree has the covering path as a single vertex which is internal and must contain the labels of the leaves. Thus a cover that uses a short path may in fact correspond to an $\mathcal{A}$–$\mathcal{A}^*$ cover and is looked-up in the corresponding case.

C.2 Trees with Two Leaf Types

For (residual) trees with exactly two leaf types, some cases do not need individual attention. Observation 10 provides a simple cover for all cases when there are only two leaves in total, thus for leaf compositions

$$\mathbf{(1,1,0,0),\ (1,0,1,0),\ (1,0,0,1)\ \text{and}\ (0,0,1,1).}$$

Furthermore, according to Proposition 17 the subtrees can be optimally reduced to have at most two leaves in each subtree. This renders the individual analysis of the following leaf compositions superfluous:

$$\begin{gathered}\mathbf{(0,0,[3],[3]),(0,0,[3],[4]),}\\ \mathbf{(1,0,[3],0),(2,0,[3],0),(0,0,[3],1),(0,0,[3],2),}\\ \mathbf{(1,0,0,[3]),(2,0,0,[3]),(0,0,1,[3]),(0,0,2,[3]),}\\ \mathbf{(1,0,0,[4]),(2,0,0,[4]),(0,0,1,[4]),(0,0,2,[4]).}\end{gathered}$$

Optimal covers for remaining leaf composition with two leaf types are given below. Note the similarities between leaf compositions $(r_{\mathcal{A}}, r_{\mathcal{B}}, r_{\varepsilon}, r_{\mathcal{AB}})$ and $(r_{\mathcal{AB}}, r_{\mathcal{B}}, r_{\varepsilon}, r_{\mathcal{A}})$ (swapping $r_{\mathcal{A}}$ and $r_{\mathcal{AB}}$).

(1,0,0,2)

any	I	$\mathcal{A}-\mathcal{AB}$, $\mathcal{AB}-\underline{\mathcal{A}/\mathcal{AB}}$	2

(2,0,0,1)

any	I	$\mathcal{A}-\mathcal{AB}$, $\mathcal{A}-\underline{\mathcal{A}/\mathcal{AB}}$	2

(2,0,0,2)

any	I	$2\times(\mathcal{A}-\mathcal{AB})$	2

cover $2\times(\mathcal{A}-\mathcal{AB})$ is optimal for all three leaf compositions (re-using a leaf, if necessary).

(2,0,1,0)

no separation	I	ε, $\mathcal{A}-\mathcal{A}$	2
else	W	$\varepsilon-\mathcal{A}$, $\mathcal{A}-\underline{\mathcal{A}}$	3

(0,0,1,2)

no separation	I	ε, $\mathcal{AB}-\mathcal{AB}$	2
else	W	$\varepsilon-\mathcal{AB}$, $\mathcal{AB}-\underline{\mathcal{AB}}$	3

(2,0,2,0)

no separation	I	$\varepsilon-\varepsilon$, $\mathcal{A}-\mathcal{A}$	3
else	W	$2\times(\varepsilon-\mathcal{A})$	4

(0,0,2,2)

no separation	I	$\varepsilon-\varepsilon$, $\mathcal{AB}-\mathcal{AB}$	3
else	W	$2\times(\varepsilon-\mathcal{AB})$	4

(1,0,2,0)

no separation		I	$\varepsilon-\varepsilon$, $\mathcal{A}$	3
else	$\mathcal{A}\multimap\varepsilon$	L	$\varepsilon-\varepsilon$, $\mathcal{A}-\mathcal{A}^*$	3
	solo ε-leaf	S	ε, $\varepsilon-\mathcal{A}$	3
	else	W	$\varepsilon-\mathcal{A}$, $\varepsilon-\underline{\varepsilon}$	4

(0,0,2,1)

no separation		I	$\varepsilon-\varepsilon$, $\mathcal{AB}$	3
else	$\mathcal{A}\multimap\varepsilon$	La	$\varepsilon-\varepsilon$, $\mathcal{AB}-\mathcal{A}^*$	3
	$\mathcal{B}\multimap\varepsilon$	Lb	$\varepsilon-\varepsilon$, $\mathcal{AB}-\mathcal{B}^*$	3
	solo ε-leaf	S	ε, $\varepsilon-\mathcal{AB}$	3
	else	W	$\varepsilon-\mathcal{AB}$, $\varepsilon-\underline{\varepsilon/\mathcal{AB}}$	4

(2,1,0,0)

no separation		I	$\mathcal{A}-\mathcal{A}$, $\mathcal{B}$	2
else	$\mathcal{B}\multimap\mathcal{A}$	Lb	$\mathcal{A}-\mathcal{A}$, $\mathcal{B}-\mathcal{B}^*$	2
	else	W	$\mathcal{A}-\mathcal{A}$, $\mathcal{B}-\underline{\mathcal{A}}$	3

(2,2,0,0)

no separation		I	$\mathcal{A}$–$\mathcal{A}$, $\mathcal{B}$–$\mathcal{B}$	2
1 separating vertex		II	$\mathcal{A}$–$\mathcal{A}$, $\mathcal{B}$–$\mathcal{B}$, sepVert	3
else	$\mathcal{A}\multimap\mathcal{B}$	La	$\mathcal{A}$–$\mathcal{A}$, $\mathcal{B}$–$\mathcal{B}$, $\underline{\mathcal{B}}$–$\mathcal{B}^*$	3
	$\mathcal{B}\multimap\mathcal{A}$	Lb	$\mathcal{A}$–$\mathcal{A}$, $\mathcal{B}$–$\mathcal{B}$, $\underline{\mathcal{A}}$–$\mathcal{A}^*$	3
	else	W	2×($\mathcal{A}$–$\mathcal{B}$)	4

C.3 Trees with Three Leaf Types

For residual leaf compositions with three leaf types we start analysing trees that have at most two leaves in each subtree.

(1,0,1,1)

ε not sep	I	ε, $\mathcal{A}$–$\mathcal{AB}$	2
else	W	$\mathcal{AB}$–$\mathcal{A}$, $\underline{\mathcal{A}}$–ε	3

(1,0,1,2)

any	I	$\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–ε	3

(1,0,2,1)

ε not sep	I	$\mathcal{A}$–$\mathcal{AB}$, ε–ε	3
else	W	$\mathcal{A}$–ε, ε–$\mathcal{AB}$	4

(1,0,2,2)

ε not sep		I	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\underline{\mathcal{A}/\mathcal{AB}}$	4
else	$\mathcal{A}\multimap\varepsilon$	La	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{A}^*$	4
	$\mathcal{B}\multimap\varepsilon$	Lb	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}^*$	4
	solo ε	S	ε, ε–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{A}$	4
	else	W	$\mathcal{A}$–ε, ε–$\mathcal{AB}$, $\mathcal{AB}$–$\underline{\mathcal{A}/\mathcal{AB}}$	5

(2,0,1,1)

any	I	ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$	3

(2,0,1,2)

ε not sep	S	ε, 2×($\mathcal{A}$–$\mathcal{AB}$)	3
else	W	ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\underline{\mathcal{AB}/\mathcal{A}}$	4

(2,0,2,1)

ε not sep		I	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{A}$–$\underline{\mathcal{A}/\mathcal{AB}}$	4
else	solo ε	S	ε, ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$	4
	$\mathcal{A}\multimap\varepsilon$	La	ε–ε, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{A}^*$	4
	$\mathcal{B}\multimap\varepsilon$ and $\mathcal{A}$ not sep	Lb	ε–ε, $\mathcal{A}$–$\mathcal{A}$, $\mathcal{AB}$–$\mathcal{B}^*$	4
	else	W	$\mathcal{AB}$–ε, ε–$\mathcal{A}$, $\mathcal{A}$–$\underline{\mathcal{A}/\mathcal{AB}}$	5

(2,0,2,2)

ε not sep	I	$\varepsilon{-}\varepsilon$, $2\times(\mathcal{A}{-}\mathcal{AB})$	4
else	W	$\mathcal{A}{-}\mathcal{AB}$, $\mathcal{AB}{-}\varepsilon$, $\varepsilon{-}\mathcal{A}$	5

(1,1,0,1)

any	I	$\mathcal{A}{-}\mathcal{AB}$, $\underline{\mathcal{AB}}{-}\mathcal{B}$	2

(1,1,0,2)

any	I	$\mathcal{A}{-}\mathcal{AB}$, $\mathcal{AB}{-}\mathcal{B}$	2

(2,1,0,1)

$\mathcal{A}$ not sep	I	$\mathcal{A}{-}\mathcal{A}$, $\mathcal{AB}{-}\mathcal{B}$	2
else	W	$\mathcal{AB}{-}\mathcal{A}$, $\mathcal{A}{-}\mathcal{B}$	3

(2,1,0,2)

any	I	$\mathcal{B}{-}\mathcal{AB}$, $\mathcal{AB}{-}\mathcal{A}$, $\mathcal{A}{-}\underline{\mathcal{A}}$	3

(2,2,0,1)

$\mathcal{A}$ not sep		Ia	$\mathcal{A}{-}\mathcal{A}$, $\mathcal{B}{-}\mathcal{AB}$, $\mathcal{B}{-}\underline{\mathcal{AB}/\mathcal{B}}$	3
$\mathcal{B}$ not sep		Ib	$\mathcal{B}{-}\mathcal{B}$, $\mathcal{A}{-}\mathcal{AB}$, $\mathcal{A}{-}\underline{\mathcal{AB}/\mathcal{A}}$	3
else	$\mathcal{A}{\multimap}\mathcal{B}$	La	$\mathcal{A}{-}\mathcal{A}^*$, $\mathcal{A}{-}\mathcal{AB}$, $\mathcal{B}{-}\mathcal{B}$	3
	$\mathcal{B}{\multimap}\mathcal{A}$	Lb	$\mathcal{B}{-}\mathcal{B}^*$, $\mathcal{B}{-}\mathcal{AB}$, $\mathcal{A}{-}\mathcal{A}$	3
	else	W	$\mathcal{A}{-}\mathcal{AB}$, $\underline{\mathcal{AB}}{-}\mathcal{B}$, $\mathcal{B}{-}\mathcal{A}$	4

(2,2,0,2)

$\mathcal{A}$ not sep	Ia	$\mathcal{A}{-}\mathcal{A}$, $2\times(\mathcal{B}{-}\mathcal{AB})$	3
$\mathcal{B}$ not sep	Ib	$\mathcal{B}{-}\mathcal{B}$, $2\times(\mathcal{A}{-}\mathcal{AB})$	3
else	W[†]	$\mathcal{A}{-}\mathcal{AB}$, $\mathcal{AB}{-}\mathcal{B}$, $\mathcal{B}{-}\mathcal{A}$	4

[†]An alternative optimal cover for W re-uses two $\mathcal{AB}$-leaves: $2\times(\mathcal{A}{-}\mathcal{AB})$, $2\times(\mathcal{B}{-}\underline{\mathcal{AB}})$.

(1,1,1,0)

ε not sep			S	ε, $\mathcal{A}{-}\mathcal{B}$	3
else	$\mathcal{A}$ not sep		Ia	$\mathcal{A}$, $\mathcal{B}{-}\varepsilon$	3
	$\mathcal{B}$ not sep		Ib	$\mathcal{B}$, $\mathcal{A}{-}\varepsilon$	3
	else	$\mathcal{A}{\multimap}\{\mathcal{B},\varepsilon\}$	La	$\mathcal{A}{-}\mathcal{A}^*$, $\mathcal{B}{-}\varepsilon$	3
		$\mathcal{B}{\multimap}\{\mathcal{A},\varepsilon\}$	Lb	$\mathcal{B}{-}\mathcal{B}^*$, $\mathcal{A}{-}\varepsilon$	3
		else	W	$\mathcal{A}{-}\varepsilon$, $\varepsilon{-}\mathcal{B}$	4

(1,1,2,0)

any	I	$\mathcal{A}{-}\varepsilon$, $\varepsilon{-}\mathcal{B}$	4

(2,1,1,0)

$\mathcal{A}$ not sep		I	$\mathcal{A}–\mathcal{A}$, $\mathcal{B}–\varepsilon$	3
else	ε not sep, $\mathcal{B}\multimap\mathcal{A}$	SLb	$\mathcal{A}–\mathcal{A}$, $\mathcal{B}–\mathcal{B}^*$, ε	3
	else	W	$\mathcal{B}–\mathcal{A}$, $\mathcal{A}–\varepsilon$	4

(2,1,2,0)

no separation			I	$\mathcal{B}$, $\varepsilon–\varepsilon$, $\mathcal{A}–\mathcal{A}$	4
$\mathcal{A}$ not sep	solo ε		S	ε, $\varepsilon–\mathcal{B}$, $\mathcal{A}–\mathcal{A}$	4
	ε separated	$\mathcal{B}\multimap\varepsilon$			
	$\{\mathcal{A},\varepsilon\}$ separated	$\mathcal{B}\multimap\{\mathcal{A},\varepsilon\}$	L	$\mathcal{B}–\mathcal{B}^*$, $\mathcal{A}–\mathcal{A}$, $\varepsilon–\varepsilon$	4
$\mathcal{A}$ separated	ε not sep	$\mathcal{B}\multimap\mathcal{A}$			
else			W	$\mathcal{B}–\varepsilon$, $\varepsilon–\mathcal{A}$, $\mathcal{A}–\underline{\mathcal{A}}$	5

(2,2,1,0)

no separation		I	$\mathcal{A}–\mathcal{A}$, $\mathcal{B}–\mathcal{B}$, ε	3
$\mathcal{A}$ not sep		IIa	$\varepsilon–\mathcal{B}$, $\mathcal{B}–\underline{\mathcal{B}}$, $\mathcal{A}–\mathcal{A}$	4
$\mathcal{B}$ not sep		IIb	$\varepsilon–\mathcal{A}$, $\mathcal{A}–\underline{\mathcal{A}}$, $\mathcal{B}–\mathcal{B}$	4
else	$\mathcal{A}\multimap\mathcal{B}$	La	$\varepsilon–\mathcal{A}$, $\mathcal{A}–\mathcal{A}^*$, $\mathcal{B}–\mathcal{B}$	4
	$\mathcal{B}\multimap\mathcal{A}$	Lb	$\varepsilon–\mathcal{B}$, $\mathcal{B}–\mathcal{B}^*$, $\mathcal{A}–\mathcal{A}$	4
	else	W	$\varepsilon–\mathcal{A}$, $\mathcal{A}–\mathcal{B}$, $\mathcal{B}–\underline{\mathcal{B}}$	5

(2,2,2,0)

no separation			I	$\mathcal{A}–\mathcal{A}$, $\mathcal{B}–\mathcal{B}$, $\varepsilon–\varepsilon$	4
$\mathcal{A}$ not sep			IIa	$\mathcal{A}–\mathcal{A}$, $2\times(\mathcal{B}–\varepsilon)$	5
$\mathcal{B}$ not sep			IIb	$\mathcal{B}–\mathcal{B}$, $2\times(\mathcal{A}–\varepsilon)$	5
ε not sep	$\mathcal{A}\multimap\mathcal{B}$		La	$\varepsilon–\varepsilon$, $\mathcal{B}–\mathcal{B}$, $\mathcal{A}–\mathcal{A}$, $\underline{\mathcal{A}}–\mathcal{A}^*$	5
	$\mathcal{B}\multimap\mathcal{A}$		Lb	$\varepsilon–\varepsilon$, $\mathcal{A}–\mathcal{A}$, $\mathcal{B}–\mathcal{B}$, $\underline{\mathcal{B}}–\mathcal{B}^*$	5
full separation	$\mathcal{A}\multimap\mathcal{B}$	solo ε	SLa	ε, $\varepsilon–\mathcal{A}$, $\mathcal{A}–\mathcal{A}^*$, $\mathcal{B}–\mathcal{B}$	5
		$\mathcal{A}\multimap\varepsilon$	LLa	$2\times(\mathcal{A}–\mathcal{A}^*)$, $\varepsilon–\varepsilon$, $\mathcal{B}–\mathcal{B}$	5
	$\mathcal{B}\multimap\mathcal{A}$	solo ε	SLb	ε, $\varepsilon–\mathcal{B}$, $\mathcal{B}–\mathcal{B}^*$, $\mathcal{A}–\mathcal{A}$	5
		$\mathcal{B}\multimap\varepsilon$	LLb	$2\times(\mathcal{B}–\mathcal{B}^*)$, $\varepsilon–\varepsilon$, $\mathcal{A}–\mathcal{A}$	5
	else		W	$\mathcal{A}–\mathcal{B}$, $\mathcal{B}–\varepsilon$, $\varepsilon–\mathcal{A}$	6

Further Cases

We now consider leaf compositions whose ε- and/or $\mathcal{AB}$-subtrees may have more than two leaves. The instances in which an optimal cover of these has a different cost or composition than we would obtain by reducing the subtrees further, are considered *non-reducible*, other instances are *reducible*. We can divide the leaf compositions into *fully reducible*, that means for all combination of tree properties we can always find an optimal cover that uses at least one homogeneous path and thus reduces the respective ε- or $\mathcal{AB}$-subtree(s) by two leaves. In other cases we can find a reduction that is optimal only for specific tree properties but have to use a cover with a higher number of heterogeneous paths for other tree properties.

Fully Reducible Cases

Except for five leaf compositions (that we discuss later) all other leaf compositions that have ε- or $\mathcal{AB}$-subtrees with more than two leaves can be optimally reduced to contain at most two leaves in each subtree. In order to provide obvious repetition of covers, we simply state the paths used to reduce the trees, the offset of costs such that the remainder of the tree can then be looked up in the derived further reduced leaf composition. Obviously, an ε–ε path costs 2 and an $\mathcal{AB}$–$\mathcal{AB}$-path costs 1 (if both are used the reduction cost amount to 3). For example for $(2, 1, 3, 0)$ the set of covering paths of $(2, 1, 1, 0)$ is taken for the exact same properties and extended by an ε–ε-path (which adds 2 to the overall cost of $(2, 1, 1, 0)$ thus deriving 5 or 6 depending on the tree properties).

Bear in mind that if a single ε-leaf remains, depending on whether it is solo or the ε-subtree was separated before, the look-up case changes in the further reduced leaf composition.

Initial $\mathcal{L}_r$	**New base $\mathcal{L}'_r$**	**Reduction**	*cost*
$(1, 0, 2, 3)$	$(1, 0, 2, 1)$	$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(1, 0, 2, 4)$	$(1, 0, 2, 2)$	$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(1, 0, 1, 3)$		$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(1, 0, 3, 1)$	$(1, 0, 1, 1)$	ε–ε	$+2$
$(1, 0, 3, 3)$		ε–ε, $\mathcal{AB}$–$\mathcal{AB}$	$+3$
$(1, 0, 1, 4)$		$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(1, 0, 3, 2)$	$(1, 0, 1, 2)$	ε–ε	$+2$
$(1, 0, 3, 4)$		ε–ε, $\mathcal{AB}$–$\mathcal{AB}$	$+3$
$(1, 1, 0, 3)$	$(1, 1, 0, 1)$	$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(1, 1, 0, 4)$	$(1, 1, 0, 2)$	$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(2, 0, 1, 3)$		$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(2, 0, 3, 1)$	$(2, 0, 1, 1)$	ε–ε	$+2$
$(2, 0, 3, 3)$		ε–ε, $\mathcal{AB}$–$\mathcal{AB}$	$+3$

Initial $\mathcal{L}_r$	**New base $\mathcal{L}'_r$**	**Reduction**	*cost*
$(2,0,1,4)$		$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(2,0,3,2)$	$(2,0,1,2)$	ε–ε	$+2$
$(2,0,3,4)$		ε–ε, $\mathcal{AB}$–$\mathcal{AB}$	$+3$
$(2,0,2,4)$	$(2,0,2,2)$	$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(2,1,0,4)$	$(2,1,0,2)$	$\mathcal{AB}$–$\mathcal{AB}$	$+1$
$(2,1,3,0)$	$(2,1,1,0)$	ε–ε	$+2$
$(2,2,3,0)$	$(2,2,1,0)$	ε–ε	$+2$

Not Generally Reducible Cases

The instances in which an optimal cover of these has a different cost or composition than we would obtain by reducing the subtrees further, are considered non-reducible. Other instances are reduced as indicated (this prevents repetition of sets of covering paths). For three leaf types we find only five leaf compositions that are not fully reducible. These are listed below.

(1,1,3,0)

full separation **and** solo ε-leaf **and** ~~$\mathcal{A}\multimap\{\mathcal{B},\varepsilon\}$~~ **and** ~~$\mathcal{B}\multimap\{\mathcal{A},\varepsilon\}$~~	S2	$\mathcal{A}$–ε, ε, ε–$\mathcal{B}$	5
otherwise	REDU	ε–$\varepsilon \cup \mathcal{L}'_r = (1,1,1,0)$	$+2$

(2,0,2,3)

ε and $\mathcal{A}$ separated **and** $\mathcal{B}\multimap\varepsilon$ **and** ~~$\mathcal{A}\multimap\varepsilon$~~ **and** ~~solo ε-leaf~~	L2	ε–ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{AB}$–$\mathcal{B}^*$	5
otherwise	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,0,2,1)$	$+1$

(2,1,0,3)

$\mathcal{A}$ separated	I	$\mathcal{B}$–$\mathcal{AB}$, $2\times(\mathcal{A}$–$\mathcal{AB})$	3
otherwise	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,1,0,1)$	$+1$

(2,2,0,3)

$\mathcal{A}$ separated **and** $\mathcal{B}$ separated **and** ~~$\mathcal{A}\multimap\mathcal{B}$~~ **and** ~~$\mathcal{B}\multimap\mathcal{A}$~~	W	$2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{B}$–$\mathcal{AB}$, $\mathcal{B}$–$\underline{\mathcal{B}/\mathcal{AB}}$	4
otherwise	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,0,1)$	$+1$

(2,2,0,4)

$\mathcal{A}$ separated **and** $\mathcal{B}$ separated	W	$2\times(\mathcal{A}$–$\mathcal{AB})$, $2\times(\mathcal{B}$–$\mathcal{AB})$	4
otherwise	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,0,2)$	$+1$

C.4 Trees with Four Leaf Types

Again we start with the base cases that have at most two leaves in each subtree.

(1,1,1,1)

$\{\mathcal{A}, \mathcal{AB}\}$ not sep from $\{\mathcal{B}, \varepsilon\}$	Ia	$\mathcal{A}$–$\mathcal{AB}$, $\mathcal{B}$–ε	3
$\{\mathcal{B}, \mathcal{AB}\}$ not sep from $\{\mathcal{A}, \varepsilon\}$	Ib	$\mathcal{B}$–$\mathcal{AB}$, $\mathcal{A}$–ε	3

(1,1,1,2)

ε not sep	S	$\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$, ε	3
else	W	$\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$, ε–$\underline{\mathcal{AB}/\mathcal{A}/\mathcal{B}}$	4

(1,1,2,1)

ε not sep			I	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\underline{\mathcal{AB}}$–$\mathcal{B}$	4
else	$\{\mathcal{A}, \varepsilon\}$ not sep from $\{\mathcal{B}, \mathcal{AB}\}$	$\mathcal{A} \multimap \varepsilon$	La	ε–ε, $\mathcal{A}$–$\mathcal{A}^*$, $\mathcal{B}$–$\mathcal{AB}$	4
		solo ε	Sa	ε, ε–$\mathcal{A}$, $\mathcal{B}$–$\mathcal{AB}$	4
	$\{\mathcal{B}, \varepsilon\}$ not sep from $\{\mathcal{A}, \mathcal{AB}\}$	$\mathcal{B} \multimap \varepsilon$	Lb	ε–ε, $\mathcal{B}$–$\mathcal{B}^*$, $\mathcal{A}$–$\mathcal{AB}$	4
		solo ε	Sb	ε, ε–$\mathcal{B}$, $\mathcal{A}$–$\mathcal{AB}$	4
	else		W	$\mathcal{A}$–ε, ε–$\mathcal{B}$, $\underline{\mathcal{AB}}$–$\mathcal{B}$	5

(1,1,2,2)

ε not sep	I	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$	4
else	W	$\mathcal{A}$–ε, ε–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$	5

(2,1,1,1)

ε not sep, $\mathcal{A}$ not sep, $\{\mathcal{A}, \varepsilon\}$ not sep	S	ε, $\mathcal{A}$–$\mathcal{A}$, $\mathcal{AB}$–$\mathcal{B}$	3
else	W	ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$, $\underline{\mathcal{AB}}$–$\mathcal{B}$	4

(2,1,2,1)

ε not sep, $\mathcal{A}$ not sep, $\{\varepsilon, \mathcal{A}\}$ not sep	I	ε–ε, $\mathcal{A}$–$\mathcal{A}$, $\mathcal{AB}$–$\mathcal{B}$	4
else	W	$\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–ε, ε–$\mathcal{B}$	5

(2,1,1,2)

any	I	$\mathcal{B}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–ε	4

(2,1,2,2)

ε not sep		I	ε–ε, $\mathcal{B}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–$\underline{\mathcal{A}/\mathcal{AB}}$	5
else	solo ε	S	ε, ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$	5
	$\mathcal{A}\multimap\varepsilon$	La	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{A}$–$\mathcal{A}^*$	5
	$\mathcal{B}\multimap\varepsilon$, $\mathcal{A}$ not sep	Lb2	ε–ε, $\mathcal{A}$–$\mathcal{A}$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{AB}$–$\mathcal{B}^*$	5
	$\mathcal{B}\multimap\varepsilon$, $\{\mathcal{B},\varepsilon\}$ not sep	Lb1	ε–ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{B}$–$\mathcal{B}^*$	5
	else	W	$\mathcal{B}$–ε, ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\underline{\mathcal{A}/\mathcal{B}/\mathcal{AB}}$	6

(2,2,1,1)

$\mathcal{A}$ not sep		Ia	$\mathcal{A}$–$\mathcal{A}$, ε–$\mathcal{B}$, $\mathcal{B}$–$\mathcal{AB}$	4
$\mathcal{B}$ not sep		Ib	$\mathcal{B}$–$\mathcal{B}$, ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$	4
ε not sep	$\mathcal{A}\multimap\mathcal{B}$	SLa	ε, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{A}^*$, $\mathcal{B}$–$\mathcal{B}$	4
	$\mathcal{B}\multimap\mathcal{A}$	SLb	ε, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{B}$–$\mathcal{B}^*$, $\mathcal{A}$–$\mathcal{A}$	4
else		W	$\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{B}$, $\mathcal{B}$–ε	5

(2,2,2,1)

$\mathcal{A}$ not sep	solo ε		Sa	ε, ε–$\mathcal{B}$, $\mathcal{B}$–$\mathcal{AB}$, $\mathcal{A}$–$\mathcal{A}$	5
	ε not sep	$\{\mathcal{A},\varepsilon\}$ not sep	Ia	ε–ε, $\mathcal{A}$–$\mathcal{A}$, $\mathcal{B}$–$\mathcal{AB}$, $\mathcal{B}$–$\underline{\mathcal{B}/\mathcal{AB}}$	5
	$\mathcal{B}\multimap\varepsilon$		Lb	ε–ε, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{B}$–$\mathcal{B}^*$, $\mathcal{A}$–$\mathcal{A}$	5
ε not sep	$\mathcal{B}\multimap\mathcal{A}$		Lb		
	$\mathcal{A}\multimap\mathcal{B}$		La	ε–ε, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{A}^*$, $\mathcal{B}$–$\mathcal{B}$	5
$\mathcal{B}$ not sep	$\mathcal{A}\multimap\varepsilon$		La		
	ε not sep	$\{\mathcal{B},\varepsilon\}$ not sep	Ib	ε–ε, $\mathcal{B}$–$\mathcal{B}$, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–$\underline{\mathcal{A}/\mathcal{AB}}$	5
	solo ε		Sb	ε, ε–$\mathcal{B}$, $\mathcal{B}$–$\mathcal{AB}$, $\mathcal{A}$–$\mathcal{A}$	5
else			W	$\mathcal{AB}$–$\mathcal{B}$, $\mathcal{B}$–ε, ε–$\mathcal{A}$, $\mathcal{A}$–$\underline{\mathcal{A}/\mathcal{AB}}$	6

(2,2,1,2)

ε not sep	$\mathcal{A}$, $\{\varepsilon,\mathcal{A}\}$ not sep	Ia	ε, $\mathcal{A}$–$\mathcal{A}$, $2\times(\mathcal{B}$–$\mathcal{AB})$	4
	$\mathcal{B}$, $\{\varepsilon,\mathcal{B}\}$ not sep	Ib	ε, $\mathcal{B}$–$\mathcal{B}$, $2\times(\mathcal{A}$–$\mathcal{AB})$	4
else		W	ε–$\mathcal{A}$, $\mathcal{A}$–$\underline{\mathcal{AB}}$, $2\times(\mathcal{B}$–$\mathcal{AB})$	5

(2,2,2,2)

ε not sep	$\mathcal{A}$ not sep	$\{\mathcal{A},\varepsilon\}$ not sep	Ia	ε–ε, $\mathcal{A}$–$\mathcal{A}$, $2\times(\mathcal{B}$–$\mathcal{AB})$	5
	$\mathcal{B}$ not sep	$\{\mathcal{B},\varepsilon\}$ not sep	Ib	ε–ε, $\mathcal{B}$–$\mathcal{B}$, $2\times(\mathcal{A}$–$\mathcal{AB})$	5
else			W	ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{B}$–ε	6

Further Cases

It follows the analysis for leaf compositions whose ε- and/or $\mathcal{AB}$-subtrees may have more than two leaves. This time, apart from *fully reducible* and *non-reducible*, we also find cases in which the ε-subtree is fully reducible, but the $\mathcal{AB}$-subtree is not.

Fully Reducible Cases

Reduction is always possible for the following leaf compositions:

Initial $\mathcal{L}_r$	**New base** $\mathcal{L}'_r$	**Reduction**	*cost*
$(1,1,1,3)$	$(1,1,1,1)$	$\mathcal{AB}$–$\mathcal{AB}$	+1
$(1,1,3,1)$		ε–ε	+2
$(1,1,3,3)$		ε–ε, $\mathcal{AB}$–$\mathcal{AB}$	+3
$(1,1,1,4)$	$(1,1,1,2)$	$\mathcal{AB}$–$\mathcal{AB}$	+1
$(1,1,3,2)$		ε–ε	+2
$(1,1,3,4)$		ε–ε, $\mathcal{AB}$–$\mathcal{AB}$	+3
$(2,1,1,4)$	$(2,1,1,2)$	$\mathcal{AB}$–$\mathcal{AB}$	+1
$(2,1,3,2)$		ε–ε	+2
$(2,1,3,4)$		ε–ε, $\mathcal{AB}$–$\mathcal{AB}$	+3
$(1,1,2,4)$	$(1,1,2,2)$	$\mathcal{AB}$–$\mathcal{AB}$	+1
$(2,1,3,1)$	$(2,1,1,1)$	ε–ε	+2
$(2,2,3,1)$	$(2,2,1,1)$	ε–ε	+2
$(2,2,3,2)$	$(2,2,1,2)$	ε–ε	+2

ε-reducible Cases

In the following cases the ε-subtree is fully reducible while the $\mathcal{AB}$-subtree is not. For the look-up case $\mathcal{L}'_r$, thus only the number of ε-leaves but not the number of leaves in the $\mathcal{AB}$-subtree are reduced. This concerns the following three leaf compositions:

Initial $\mathcal{L}_r$	**New base** $\mathcal{L}'_r$	**Part Reduction**	*cost*
$(2,1,[3],3)$	$(2,1,1,3)$	ε–ε	+2
$(2,2,[3],3)$	$(2,2,1,3)$	ε–ε	+2
$(2,2,[3],4)$	$(2,2,1,4)$	ε–ε	+2

Not Generally Reducible Cases

The following eight leaf compositions are not generally reducible. This means for some properties they are reducible, but not for others.

(1,1,2,3)

ε not sep			REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (1,1,2,1)$	+1
else	$\{\mathcal{A},\varepsilon\}$ not sep	solo ε	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (1,1,2,1)$	+1
		$\mathcal{A}\multimap\varepsilon$	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (1,1,2,1)$	+1
	$\{\mathcal{B},\varepsilon\}$ not sep	solo ε	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (1,1,2,1)$	+1
		$\mathcal{B}\multimap\varepsilon$	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (1,1,2,1)$	+1
	$\mathcal{A}\multimap\varepsilon$		L2a	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{AB}$–$\mathcal{A}^*$	5
	$\mathcal{B}\multimap\varepsilon$		L2b	ε–ε, $\mathcal{A}$–$\mathcal{AB}$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{AB}$–$\mathcal{B}^*$	5
	else		REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (1,1,2,1)$	+1

(2,1,1,3)

ε not sep **and**			
$\mathcal{A}$ not sep **and** $\{\mathcal{A},\varepsilon\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,1,1,1)$	+1
$\mathcal{A}$ not sep **and** $\{\mathcal{A},\varepsilon\}$ sep $\mathcal{A}$ sep	S2	ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{AB}$–$\mathcal{B}$	4
else	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,1,1,1)$	+1

(2,2,1,3)

$\mathcal{A}$ sep **and** $\mathcal{B}$ sep **and** $\cancel{\mathcal{A}\multimap\mathcal{B}}$ **and** $\cancel{\mathcal{B}\multimap\mathcal{A}}$	W2	$2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{B}$–ε	5
else	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,1,1)$	+1

(2,1,2,3)

ε not sep **and**			
$\mathcal{A}$ not sep **and** $\{\mathcal{A},\varepsilon\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,1,2,1)$	+1
$\mathcal{A}$ not sep **and** $\{\mathcal{A},\varepsilon\}$ sep $\mathcal{A}$ sep	II	$2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{AB}$–$\mathcal{B}$, ε–ε	5
else	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,1,2,1)$	+1

(2,1,2,4)

ε sep **and** $\mathcal{A}$ sep **and** $\{\mathcal{B},\varepsilon\}$ sep **and** ~~solo ε~~ **and** $\cancel{\mathcal{A}\multimap\varepsilon}$ **and** $\mathcal{B}\multimap\varepsilon$	Lb3	ε–ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{AB}$–$\mathcal{B}^*$	6
else	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,1,2,2)$	+1

(2,2,1,4)

ε not sep **and**				
	$\mathcal{A}$ not sep **and** $\{\varepsilon,\mathcal{A}\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,1,2)$	+1
	$\mathcal{B}$ not sep **and** $\{\varepsilon,\mathcal{B}\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,1,2)$	+1
	else	S2	ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $2\times(\mathcal{B}$–$\mathcal{AB})$	5
else		REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,1,2)$	+1

(2,2,2,3)

$\mathcal{A}$ not sep	solo ε	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,1)$	+1
	ε not sep and $\{\mathcal{A},\varepsilon\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,1)$	+1
	$\mathcal{B}\multimap\varepsilon$	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,1)$	+1
ε not sep	$\mathcal{B}\multimap\mathcal{A}$	REDU		
	$\mathcal{A}\multimap\mathcal{B}$	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,1)$	+1
$\mathcal{B}$ not sep	$\mathcal{A}\multimap\varepsilon$	REDU		
	ε not sep $\{\mathcal{B},\varepsilon\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,1)$	+1
	solo ε	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,1)$	+1
solo ε		S2	ε, ε–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{AB}$, $2\times(\mathcal{AB}$–$\mathcal{B})$	6
$\mathcal{A}\multimap\varepsilon$		L2a	ε–ε, $2\times(\mathcal{B}$–$\mathcal{AB})$, $\mathcal{AB}$–$\mathcal{A}$, $\mathcal{A}$–$\mathcal{A}^*$	6
$\mathcal{B}\multimap\varepsilon$		L2b	ε–ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $\mathcal{AB}$–$\mathcal{B}$, $\mathcal{B}$–$\mathcal{B}^*$	6
ε not sep		III	ε–ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $2\times(\mathcal{AB}$–$\mathcal{B})$	6
else		REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,1)$	+1

(2,2,2,4)

ε not sep **and**				
	$\mathcal{A}$ not sep **and** $\{\mathcal{A},\varepsilon\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,2)$	+1
	$\mathcal{B}$ not sep **and** $\{\mathcal{B},\varepsilon\}$ not sep	REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,2)$	+1
	else	II	ε–ε, $2\times(\mathcal{A}$–$\mathcal{AB})$, $2\times(\mathcal{B}$–$\mathcal{AB})$	6
else		REDU	$\mathcal{AB}$–$\mathcal{AB} \cup \mathcal{L}'_r = (2,2,2,2)$	+1

D Detailed Sampling Results for sty-stm Comparison of γ-Proteobacteria

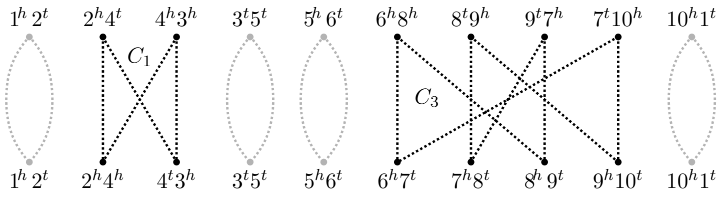


Figure D.12: Adjacency graph for `sty-stm` comparison.

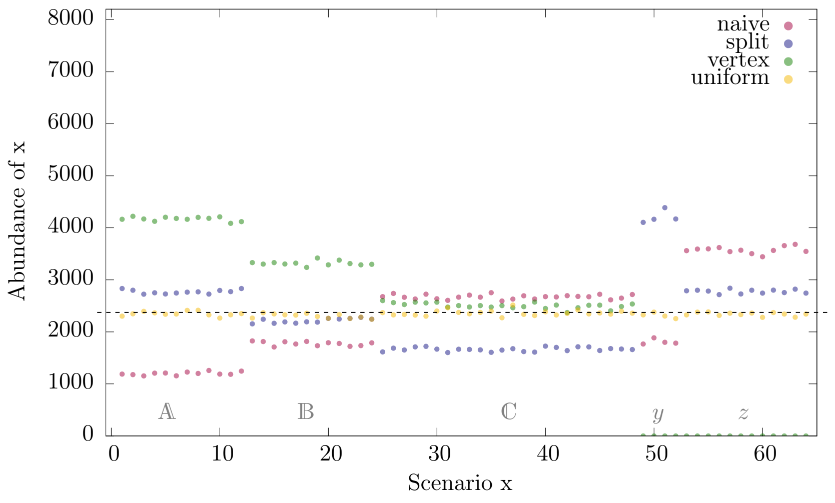


Figure D.13: Sampling results for all four methods for the `sty-stm` comparison. Scenarios ordered by category as listed below.

In the standard notation of `UniMoG` input/output: the `")"` signifies a circular chromosome (all markers to the left are on it in that order).

1#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 7 9 8 10 1)	2 -4 -3 5 6 7 8 10 1) 9)	A
2#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -7 10 1) 9 8)	2 -4 -3 5 6 -9 -8 -7 10 1)	A
3#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -8 7 9 10 1)	2 -4 -3 5 6 -8 -7 9 10 1)	A
4#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -7 10 1) 9 8)	2 -4 -3 5 6 7 10 1) 9 8)	A
5#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 7 9 8 10 1)	2 -4 -3 5 6 7 10 1) 9 8)	A
6#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -8 7 9 10 1)	2 -4 -3 5 6 -7 8 9 10 1)	A
7#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -8 -7 10 1) 9)	2 -4 -3 5 6 -9 -8 -7 10 1)	A
8#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -8 -7 10 1) 9)	2 -4 -3 5 6 7 8 10 1) 9)	A
9#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 7 9 8 10 1)	2 -4 -3 5 6 7 9 10 1) 8)	A
10#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -8 7 9 10 1)	2 -4 -3 5 6 7 9 10 1) 8)	A
11#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -7 10 1) 9 8)	2 -4 -3 5 6 -7 8 9 10 1)	A
12#	1 2 -4 -3 5 6 -8 -9 -7 10 1)	2 -4 -3 5 6 -8 -7 10 1) 9)	2 -4 -3 5 6 -8 -7 9 10 1)	A
13#	1 2 4 -3 5 6 -8 -7 10 1) 9)	2 -4 -3 5 6 -8 -7 10 1) 9)	2 -4 -3 5 6 -9 -8 -7 10 1)	B
14#	1 2 4 -3 5 6 -7 10 1) 9 8)	2 -4 -3 5 6 -7 10 1) 9 8)	2 -4 -3 5 6 7 10 1) 9 8)	B
15#	1 2 4 -3 5 6 7 9 8 10 1)	2 -4 -3 5 6 7 9 8 10 1)	2 -4 -3 5 6 7 8 10 1) 9)	B
16#	1 2 4 -3 5 6 -8 7 9 10 1)	2 -4 -3 5 6 -8 7 9 10 1)	2 -4 -3 5 6 -7 8 9 10 1)	B

```
17#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 -4 -3 5 6 -7 10 1 ) 9 8 )     2 -4 -3 5 6 -9 -8 -7 10 1 )   B
18#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 -4 -3 5 6 -8 -7 10 1 ) 9 )    2 -4 -3 5 6 -8 -7 9 10 1 )    B
19#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 -4 -3 5 6 -8 -7 10 1 ) 9 )    2 -4 -3 5 6 7 8 10 1 ) 9 )    B
20#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 -4 -3 5 6 -8 7 9 10 1 )       2 -4 -3 5 6 7 9 10 1 ) 8 )    B
21#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 -4 -3 5 6 -8 7 9 10 1 )       2 -4 -3 5 6 -8 -7 9 10 1 )    B
22#  1 2 4 -3 5 6 7 9 8 10 1 )       2 -4 -3 5 6 7 9 8 10 1 )        2 -4 -3 5 6 7 10 1 ) 9 8 )    B
23#  1 2 4 -3 5 6 7 9 8 10 1 )       2 -4 -3 5 6 7 9 8 10 1 )        2 -4 -3 5 6 7 9 10 1 ) 8 )    B
24#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 -4 -3 5 6 -7 10 1 ) 9 8 )     2 -4 -3 5 6 -7 8 9 10 1 )     B
25#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 4 -3 5 6 -7 8 9 10 1 )        2 -4 -3 5 6 -7 8 9 10 1 )     C
26#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 4 -3 5 6 -9 -8 -7 10 1 )      2 4 -3 5 6 7 8 9 10 1 )       C
27#  1 2 4 -3 5 6 7 9 8 10 1 )       2 4 -3 5 6 7 10 1 ) 9 8 )       2 4 -3 5 6 7 8 9 10 1 )       C
28#  1 2 4 -3 5 6 7 9 8 10 1 )       2 4 -3 5 6 7 8 10 1 ) 9 )       2 4 -3 5 6 7 8 9 10 1 )       C
29#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 4 -3 5 6 7 10 1 ) 9 8 )       2 4 -3 5 6 7 8 9 10 1 )       C
30#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 4 -3 5 6 -8 -7 9 10 1 )       2 -4 -3 5 6 -8 -7 9 10 1 )    C
31#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 4 -3 5 6 -8 -7 9 10 1 )       2 -4 -3 5 6 -8 -7 9 10 1 )    C
32#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 4 -3 5 6 7 10 1 ) 9 8 )       2 -4 -3 5 6 7 10 1 ) 9 8 )    C
33#  1 2 4 -3 5 6 7 9 8 10 1 )       2 4 -3 5 6 7 9 10 1 ) 8 )       2 4 -3 5 6 7 8 9 10 1 )       C
34#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 4 -3 5 6 7 9 10 1 ) 8 )       2 4 -3 5 6 7 8 9 10 1 )       C
35#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 4 -3 5 6 -7 8 9 10 1 )        2 4 -3 5 6 7 8 9 10 1 )       C
36#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 4 -3 5 6 -9 -8 -7 10 1 )      2 4 -3 5 6 7 8 9 10 1 )       C
37#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 4 -3 5 6 -8 -7 9 10 1 )       2 4 -3 5 6 7 8 9 10 1 )       C
38#  1 2 4 -3 5 6 -7 10 1 ) 9 8 )    2 4 -3 5 6 -9 -8 -7 10 1 )      2 -4 -3 5 6 -9 -8 -7 10 1 )   C
39#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 4 -3 5 6 -7 8 9 10 1 )        2 -4 -3 5 6 -7 8 9 10 1 )     C
40#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 4 -3 5 6 7 8 10 1 ) 9 )       2 -4 -3 5 6 7 8 10 1 ) 9 )    C
41#  1 2 4 -3 5 6 7 9 8 10 1 )       2 4 -3 5 6 7 8 10 1 ) 9 )       2 -4 -3 5 6 7 8 10 1 ) 9 )    C
42#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 4 -3 5 6 -8 -7 9 10 1 )       2 4 -3 5 6 7 8 9 10 1 )       C
43#  1 2 4 -3 5 6 7 9 8 10 1 )       2 4 -3 5 6 7 10 1 ) 9 8 )       2 -4 -3 5 6 7 10 1 ) 9 8 )    C
44#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 4 -3 5 6 -9 -8 -7 10 1 )      2 -4 -3 5 6 -9 -8 -7 10 1 )   C
45#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 4 -3 5 6 -7 8 9 10 1 )        2 4 -3 5 6 7 8 9 10 1 )       C
46#  1 2 4 -3 5 6 7 9 8 10 1 )       2 4 -3 5 6 7 9 10 1 ) 8 )       2 -4 -3 5 6 7 9 10 1 ) 8 )    C
47#  1 2 4 -3 5 6 -8 -7 10 1 ) 9 )   2 4 -3 5 6 7 8 10 1 ) 9 )       2 4 -3 5 6 7 8 9 10 1 )       C
48#  1 2 4 -3 5 6 -8 7 9 10 1 )      2 4 -3 5 6 7 9 10 1 ) 8 )       2 -4 -3 5 6 7 9 10 1 ) 8 )    C

49#  1 2 -4 -3 5 6 -8 -9 -7 10 1 )   2 -4 -3 5 6 -9 -7 10 1 ) 8 )    2 -4 -3 5 6 -9 -8 -7 10 1 )   y
50#  1 2 -4 -3 5 6 -8 -9 -7 10 1 )   2 -4 -3 5 6 -8 -9 7 10 1 )      2 -4 -3 5 6 -8 -7 9 10 1 )    y
51#  1 2 -4 -3 5 6 -8 -9 -7 10 1 )   2 -4 -3 5 6 -9 -7 10 1 ) 8 )    2 -4 -3 5 6 7 9 10 1 ) 8 )    y
52#  1 2 -4 -3 5 6 -8 -9 -7 10 1 )   2 -4 -3 5 6 -8 -9 7 10 1 )      2 -4 -3 5 6 7 10 1 ) 9 8 )    y
53#  1 2 4 -3 5 6 -9 -7 10 1 ) 8 )   2 4 -3 5 6 -9 -8 -7 10 1 )      2 -4 -3 5 6 -9 -8 -7 10 1 )   z
54#  1 2 4 -3 5 6 -8 -9 7 10 1 )     2 4 -3 5 6 7 10 1 ) 9 8 )       2 4 -3 5 6 7 8 9 10 1 )       z
55#  1 2 4 -3 5 6 -8 -9 7 10 1 )     2 4 -3 5 6 -8 -7 9 10 1 )       2 -4 -3 5 6 -8 -7 9 10 1 )    z
56#  1 2 4 -3 5 6 -8 -9 7 10 1 )     2 -4 -3 5 6 -8 -9 7 10 1 )      2 -4 -3 5 6 -8 -7 9 10 1 )    z
57#  1 2 4 -3 5 6 -9 -7 10 1 ) 8 )   2 4 -3 5 6 7 9 10 1 ) 8 )       2 -4 -3 5 6 7 9 10 1 ) 8 )    z
58#  1 2 4 -3 5 6 -8 -9 7 10 1 )     2 4 -3 5 6 -8 -7 9 10 1 )       2 4 -3 5 6 7 8 9 10 1 )       z
59#  1 2 4 -3 5 6 -8 -9 7 10 1 )     2 4 -3 5 6 7 10 1 ) 9 8 )       2 -4 -3 5 6 7 10 1 ) 9 8 )    z
60#  1 2 4 -3 5 6 -9 -7 10 1 ) 8 )   2 4 -3 5 6 -9 -8 -7 10 1 )      2 4 -3 5 6 7 8 9 10 1 )       z
61#  1 2 4 -3 5 6 -9 -7 10 1 ) 8 )   2 4 -3 5 6 7 9 10 1 ) 8 )       2 4 -3 5 6 7 8 9 10 1 )       z
62#  1 2 4 -3 5 6 -8 -9 7 10 1 )     2 -4 -3 5 6 -8 -9 7 10 1 )      2 -4 -3 5 6 7 10 1 ) 9 8 )    z
63#  1 2 4 -3 5 6 -9 -7 10 1 ) 8 )   2 -4 -3 5 6 -9 -7 10 1 ) 8 )    2 -4 -3 5 6 7 9 10 1 ) 8 )    z
64#  1 2 4 -3 5 6 -9 -7 10 1 ) 8 )   2 -4 -3 5 6 -9 -7 10 1 ) 8 )    2 -4 -3 5 6 -9 -8 -7 10 1 )   z

 A   B   C   y   z
12  12  24   4  12
```