AUTOMATIC EXTRACTION OF LOGICALLY CONSISTENT ONTOLOGIES FROM TEXT CORPORA

John Philip M^{c}Crae

DOCTOR OF
PHILIOSOPHY

Department of Informatics,
School of Multidisciplinary Sciences
The Graduate Univesity of Advanced Studies (SOKENDAI)

2009 (School Year)

September 2009

A dissertation submitted to the Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University of Advanced Studies (SOKENDAI)
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Advisory Committee:

Advisor	Nigel Collier, *Assoc. Prof.*	National Institute of Informatics (NII), SOKENDAI
Members	Nobuhiro Furuyama, *Assoc. Prof.*	NII,SOKENDAI
	Katsumi Inoue, *Prof.*	NII, SOKENDAI
	Ken Satoh, *Prof.*	NII, SOKENDAI
	Hideaki Takeda, *Prof.*	NII, SOKENDAI
	Hong Yu, *Assoc. Prof.*	University of Wisconsin, Milwaukee

(In alphabetic order)

Abstract

Ontologies provide a structured description of the concepts and terminology used in a particular domain and provide valuable knowledge for a range of natural language processing applications. However, for many domains and languages ontologies do not exist and manual creation is a difficult and resource-intensive process. As such, automatic methods to extract, expand or aid the construction of these resources is of significant interest.

There are a number of methods for extracting semantic information about how terms are related from raw text, most notably the approach of Hearst [1992], who used *patterns* to extract hypernym information. This method was manual and it is not clear how to automatically generate patterns, which are specific to a given relationship and domain. I present a novel method for developing patterns based on the use of alignments between patterns. Alignment works well as it is closely related to the concept of a *join-set* of patterns, which minimally generalise over-fitting patterns. I show that join-sets can be viewed as an reduction on the search space of patterns, while resulting in no loss of accuracy. I then show the results can be combined by a *support vector machine* to a obtain a classifier, which can decide if a pair of terms are related. I applied this to several data sets and conclude that this method produces a precise result, with reasonable recall.

The system I developed, like many semantic relation systems, produces only a binary decision of whether a term pair is related. Ontologies have a structure, that limits the forms of networks they represent. As the relation extraction is generally noisy and incomplete, it is unlikely that the extracted relations will match the structure of the ontology. As such I represent the structure of ontology as a set of logical statements, and form a consistent ontology by finding the network closest to the relation extraction system's output, which is consistent with these restrictions. This gives a novel *NP-hard* optimisation problem, for which I develop several algorithms. I present simple greedy approaches, and branch and bound approaches, which my results show are not sufficient for this problem. I then use resolution to show how this problem can be stated as an *integer programming problem*, which can be efficiently solved by relaxing it to a *linear programming problem*. I show that this result can efficiently solve the problem, and furthermore when applied to the result of the relation extraction system, this improves the quality of the extraction as well as converting it to an ontological structure.

Acknowledgement

I would like to thank my supervisor, Nigel Collier, for his assistance throughout the course of my studies as well as all the people at the NII who assisted me in many ways. I would also like to thank my parents, Mary and Robert for the support they provided throughout the duration of my course and Katrin Gengenbach for her help in checking the text of this thesis.

Contents

List of Figures

Chapter 1

Introduction

Ontologies are a formal representation of concepts in a domain and are intrinsically linked with the natural language and as such generally can be thought of as structured linguistic databases, and have numerous applications in NLP. Ontologies often need to be designed specifically for different domains and the terminology they contain will naturally be different for different languages. The task of constructing an ontology requires a large amount of manual effort and automatic procedures to aid or replace this process are highly desirable. There are several methods of extracting these ontologies from raw text, however these methods often do not create an extraction that is consistent with the structure of the ontology. This structure can be specified as a set of logical restrictions and by making the result of an automatic extraction system consistent with these restrictions it is possible to extract a sound ontology.

Ontologies provide a hugely valuable resource for all kinds of natural language processing work. Possibly the most well-known example of a ontology[1] is WordNet which is simply a list of terms that are related by several principles such as

- hypernym: X is a hypernym of Y if Y is a kind of X (e.g., "animal" is a hypernym of "cat")
- hyponym: X is a hyponym of Y if X is a kind of Y
- holonym: X is a holonym of Y if Y is a part of X (e.g., "car" is a holonym of "steering wheel")
- meronym: X is a meronym of Y if X is a part of Y
- synonym: X is a synonym of Y if X is the same as Y (e.g, "animal" is the same as "creature")

[1] Many authors differentiate between an *ontology* and a *semantic network*. In this paper I will not make the distinction, but instead use "ontology" to refer to any structured semantic database

WordNet then lists for about 150,000 words the relations between these words. This can be used for a large number of applications, for example for co-reference resolution, which is the process of working out anaphoric references in the text. In the quote:

> Hoplophoneus is another type of cat. During the Oligocene (e.g., about 20 million years older than Smilodon), this creature hunted...

The objective is to deduce that the term "creature" in the second sentence refers to the "cat" in the first sentence, and this is much easier if the system can refer to knowledge that "cat" is a hyponym of "creature", as it is much more likely for a reference to be true if the two terms have a hypernym relationship, because this knowledge indicates that the "cat" in the first sentence could also be the "creature" in the second sentence. Another important application of ontologies is in information retrieval (IR), this is the process of retrieving the documents from a set of documents given some criteria such as a set of keywords, which are most relevant to the specific criteria. A simple information retrieval system trying to extract information for the keyword "cat" might simply return all documents, where the word "cat" appears, however by using a ontology it is possible to improve this result by also including hypernym terms such as "feline". This means that the IR system can now retrieve more potentially relevant documents.

In specific domains such as disease control, it is often the case that a general ontology such as WordNet is not effective for the purposes as it may not include specific terms that needed and its limited set of relations may not be sufficient for the applications. For example in disease control the system may come across a term like "EV-71", which is an abbreviation for the virus "Enterovirus 71" and is unlikely to be in many general-purpose ontologies. Furthermore, it is desirable to know that occurrences of the term "Enterovirus 71" or "EV-71" indicate the disease "hand, foot and mouth disease", this requires that the ontology contains the relation "causative agent". In response to these issues a lot of research has been invested in adding concepts such as logical restrictions, valued attributes, rules and axioms, in addition to the simple terms and relations. This allows ontologies to better represent the terminology of specific domains, however creating an ontology is a much more complex process.

One further problem with many ontologies is their lack of availability and completeness. Even for English many terms are not included in WordNet, especially those for specialised domains such as bio-medicine. For other languages, especially non-European languages, such as Vietnamese or Thai, resources of the size and completeness of WordNet simply do not exist.

For these reasons, it is necessary to investigate the topic of automatic extraction of ontologies, as this can be used for great benefit in a wide range of domains. This involves several tasks: firstly it is necessary to identify the terms from a document which are of most interest, for example for disease control, this means identify diseases, virii, bacteria, symptoms etc. Then once these terms have been extracted, assuming there is a relations to be extract, the goal is to work out which of these terms are related and how. There are a number of ways

to do this, first of all simply looking at the terms themselves, for example it is often possible to tell that two terms such as "AIDS" and "acquired immunodeficiency syndrome" are related as one is simply abbreviations of the other. It is also possible to use the contexts that terms appear close to each other, for example seeing the phrase

> AIDS is a disease of the human immune system...

Is a clear sign that "AIDS" is a hyponym of "disease". We can also used shared contexts: for example, the terms "dog" and "terrier" are frequently followed by the verb "bark", which suggest some degree of similarity between those two terms. Although all of these methods are useful, only pattern based methodologies can differentiate between different relations, however it is not clear how to manually extract patterns, which are indicative of a particular relation, and generalise them sufficiently.

Once the terms and the relations between them have been extracted it is necessary to create an ontology for them, or incorporate them into an existing ontology. Ontologies generally have a strong structure, however if the system is not 100% accurate its extraction may not fit the structure of the network, so the result must be altered to fit into the structure of the semantic network. A common example of a structure is the *synonym/hypernym taxonomy*, this is the basic structure used by WordNet and requires that terms are grouped into synonyms sets (or *synsets* in WordNet's terminology) and that these synonym sets are arranged hierarchically in a tree graph. This is quite different from the simple network form where two terms are simply connected or not connected, for example if we know that "cat" is a hyponym of "mammal" and that "mammal" is a hypernym of "animal", then it is only possible to add this into a synonym/hypernym taxonomy if it is also known that "cat" is a hyponym of "animal". So to add this data into the structure the system needs to be able to decide if it is more likely that "cat" is a hyponym of "animal" or one of the two extracted facts was erroneous.

This idea of structures is further extended by ontologies, which allow multiple relations to be defined and allows for restrictions to be placed on these relations. For example the ontology language OWL 2 uses the following restrictions on relations:

- **Transitive:** If a relation is transitive and x is related to y and y is related to z, then x is also related to z.
- **Symmetric:** If a relation is symmetric then it means if x is related to y then y is also related to x.
- **Asymmetric:** If a relation is asymmetric then it cannot be true that both x is related to y and y is related to x.
- **Reflexive:** If a relation is reflexive then every element must be related to itself.

- **Irreflexive:** If a relation is irreflexive then no element can be related to itself.
- **Cardinality:** A relation may have a minimum cardinality of n and a maximum cardinality of m, these means each element x must be related to at least n other elements and cannot be related to more than m.
- **Disjoint:** Two relations can be disjoint meaning that a pair of elements cannot be related by both relations.
- **Range/Domain Limits:** A relation may have a limit to what it can state a relation exists between, for example the relation "causative agent" might be limited to be between "pathogens" and "diseases".
- **Sub-property:** A relation may be a more specific version of another relation hence it can only apply to a pair of elements if the super-relation also applies.

In order to handle such restrictions it is necessary to develop an algorithm, which can handle these complex restrictions and the way their interactions. I model the set of elements and the relations between them as a set of graphs, where there is a link between each element if the system extracted that the relationship holds between these elements. From this model I can also say that only some of these possible networks are correct with respect to the relationships and then I say that the goal is to find the network which is correct and closest to the extracted relations. This is in general quite difficult, as even if there are restrictions that can be stated in just propositional logic form then the problem of finding a single consistent network is *NP-complete*, as it is the *Boolean satisfiability problem*, which means that attempting to find the optimal solution will be an *NP-hard* problem. Furthermore if I wish to deal with the kind of rules found in ontology languages such as OWL, I need to be able to deal with *first-order logic* restrictions and this can lead to further problems such as undecidability. This means that robust and specialised algorithms will be needed to be able to quickly find solutions to these problems.

In Summary, there are two key problems I am concerned with in the paper. Firstly the problem of extracting relationships from text, which I will handle through the use of patterns. It is not in general clear how to form these patterns in such a way that they are sufficiently generalised. I will present a methodology that can generalise this pattern, and cuts out the majority of the search space, without losing any potentially valuable patterns. The second issue is the task of forming these binary relations into a structured ontology. I assume that the structure of the ontology can be represented by a set of logical statements and show that this gives a novel problem, for which new algorithms need to be developed. I then develop that into an algorithm which is more efficient than naïve approaches or algorithms adapted from related problems.

Chapter 2

Related Work

2.1 Ontologies

2.1.1 WordNet and related networks

WordNet is considered by many to be one of the most important ontologies currently available. It was initiated at Princeton University in 1985, and was intended to be a resource that was useful to both humans and machines, as a knowledge source about the English language. It achieves this by acting as a combined thesaurus/dictionary, combining words with the same meaning into *synsets* ("synonymy sets") and then organising them into a hierarchical structure. The database focuses on general English and contains about 150,000 terms in 115,000 synsets. One of the main failings of WordNet is its comparatively low coverage of terms from specialised domains, for example Bodenreider et al. [2003] attempted to match terms from a biomedical resource on genes called the Gene Ontology[Ashburner et al., 2000], which contains very specific terminology related to genes and genetic disorders. They found that WordNet gave very poor coverage of such terminology, for example they found that only 47 out of 1903 (2.5%) of "genetic disease names" could be mapped to terms in WordNet. While, this is not surprising, as the kind of terminology used in a specific domain such as Genetics is naturally very different from that used in the general domain that WordNet is aimed at, it does show that it is necessary to create ontologies that are applicable to specific domains.

One example of an attempt to extend WordNet to a new domain is Medical WordNet ([Smith and Fellbaum, 2004]); this project added not only a large amount of medical terminology, such as disease names and genetic terminologies but also two additional networks that they call the MedicalFactNet and MedicalBeliefNet. These two networks are intended to describe medical facts such as

> Aspirin doesn't help in the case of a hangover

And split these facts into two groups, one for those facts, which are well estab-

lished and agreed upon by the majority of medical professionals (MedicalFactNet) and a second set, which do not have a consensus of agreement, but may be useful for making diagnoses (MedicalBeliefNet).

As WordNet is a resource that contains knowledge only in English this means that a new network must be made for nearly every other language, as was seen in the project called EuroWordNet [Vossen, 1998], which attempted to create word nets for Dutch, Italian, Spanish, German, French, Czech, and Estonian. As well as creating new ontologies for these languages, they also created a *inter-lingual-index*, which created a link between "rocket" in the English WordNet, "Rakete" in the German part of EuroWordNet and "fusée" in the French part; this was intended to aid automatic translation. In addition they added new relationships not found in the original WordNet such as role_patient which indicated the usual subject of a verb, for example "student" is often the subject of "teaches".

WordNet is very effective for nouns, however as in the previous example, the network lacks a lot of information, which would be useful for verbs. For this reason, FrameNet [Baker et al., 1998] was created, which encodes semantic information about its terminology in what it calls "frames". These frames state a large amount of information about how a verb interacts with other terminology.

> frame(DRIVING)
> inherits(TRANSPORTATION)
> frame_elements(DRIVER (=MOVER), VEHICLE (=MEANS), RIDER(s) (=MOVER(s)), CARGO (=MOVER(s)))
> scene(DRIVER starts VEHICLE, DRIVER controls VEHICLE, DRIVER stops VEHICLE)

This frame for example concerns the action of "driving" and shows that this frame has several other elements such as "driver" and "vehicle" and "cargo", which are implied by the action. This allows for a system to have a more complete understanding of the action and how it can be interpreted.

2.1.2 Logic and Ontologies

It is clear that for many different domains and ontologies with different purposes, a large number of different structures and relationships need to be defined, for this reason it is necessary to standardise the form of these databases in a logically consistent manner. This leads to the development of ontology languages, and in particular the ontology language OWL (Web Ontology Language)[1]. One of the first papers to claim the need for a unified design for ontologies was Gruber [1995], who claimed that an ontology was required to have the following properties

- **Clarity:** All terms should be uniquely and unambiguously defined

[1] Acronym is deliberately incorrect: "Why not be inconsistent in at least one aspect of a language, which is all about consistency?" - Guus Schreiber

- **Coherence:** The logical foundation must be clear and "at least" logically consistent.
- **Extendability:** The ontology should be designed so that it can be extended in new directions without the need to redefine base definitions.
- **Minimal Encoding Bias:** The system should be minimally dependent on its encoding.
- **Minimal Ontological Commitment:** The system should make as few claims about the world as is necessary for its purposes.

This leads to the creation of several ontology languages such as KIF, DAML+OIL and CycL, however I shall focus in this paper on OWL (Web Ontology Language)[2], which has grown popular based on its usage in the "semantic web". OWL defines several key concepts

- **Classes:** Hierarchically organised groups of concepts
- **Instances:** Particular instances of a class
- **Properties:** Relationships between classes/instances and other classes/instances, or *valued properties* which take numeric/text values.
- **Restrictions:** A list of axiomatic restrictions on a particular property.

While OWL and other languages are useful for writing sound ontologies, there is a key problem in how to construct the ontology and what to include and not to include. A very interesting work is that of DOLCE [Gangemi et al., 2003], a "top-level ontology", this does not attempt to provide the terminology for a specific domain but instead provide a basis for all ontological discussion. They did this by defining a set of classes, which could be logically defined, namely "Endurant" (things that are), "Perdurant" (things that happen), "Quality" (something that describes something else) and "Abstract" (something that is not a thing). More pertinently they also dealt with the definition of a taxonomic relation in Guarino and Welty [2000] where they state and define four key properties that relationships should be defined by

- **Identity:** This is "the problem of distinguishing a specific instance of a certain class from other instances of that class".
- **Unity:** This is the problem of "distinguishing the parts of an instance from the rest of the world".
- **Essence/Rigidity:** This defines whether a property is "essential" to that instance or if it is mutable, for example a "person" is *rigidly* a person, as this fact cannot change, however a "student" may cease to be a student

[2]Formally defined at http://www.w3.org/TR/owl-features/ , a solid introduction is given in Antoniou and van Harmelen [2004]

Name	Definition
`#$TransporationDevice`	The device used for transportation
`#$Conveyance`	The collection objects used for moving things
`#$Vehicle`	The device, which provides motive force
`#$transportees`	The objects being transported
`#$passengers`	The persons being transported
`#$fromLocation`	The origin of the transportation event
`#$toLocation`	The destination of the transportation event

Figure 2.1: Some examples relations of Cyc

- **Dependence:** This is the difference between those problems which are extrinsic/intrinsic, that is those that are defined by the object itself, and those that are dependent on other objects.

By giving these definitions it makes it easier for to add strong logical principles to the relationships and make an ontology a more well-defined object.

One of the most complete ontologies created was the Cyc ontology of Lenat [1995], which attempted to encode a comprehensive database of "common sense knowledge". This means that in addition to the having hypernymy/synonymy information like most ontology, Cyc also contains approximately 6000 other concepts, including for example semantic relations like "capital city of" and facts like "capable of flight". To demonstrate the scope of the database figure 2.1 shows some of the relations that are used to describe a transportation event.

2.1.3 Applications of Ontologies

The literature on applications of ontologies is too broad to fully consider within the scope of this thesis, but I review here several of the more pertinent applications, which highlight the need for good ontologies. I first return to WordNet as it is the most widely used semantic network and I will discuss some of the practical applications, which WordNet has been used for. *Text retrieval* is the task of retrieving from a set of document a subset, which is relevant to some set of terms commonly referred to as *key-words*. This should be familiar to most readers as the task performed by Internet search engines such as Google. It should be clear that the use of synonym information from an ontology such as WordNet should be able to enable the system to find documents, which do not contain the key-words and disambiguate documents, which contain a key-word but are not appropriate to the given query. Indexing is the procedure of creating an index from the terms to the documents, which contain these terms and in Gonzalo et al. [1998], they used WordNet's synsets to index their documents, rather than the key-words themselves. They found that by doing this and enabling querying by a WordNet synset instead of a key-word, they improved the precision of their results from 48.0% to 62.0% against a baseline system. A similar experiment was done in Voorhees [1993], where she turned to the problem of using WordNet information to disambiguate key-words with

multiple meanings. For example, the term "base" has many meanings, however if it occurs alongside words such as "bat", "glove" and "hit", then it can be assumed it is the baseball term meaning

> A place that the runner must touch before scoring[3]

They found mixed results for the ability to improve the quality of retrieved documents, however they claim that using disambiguation is essential for future systems.

Another key use of ontologies is for *co-reference resolution* and similar tasks. Co-reference resolution is the problem of resolving terms which refer to terms mentioned earlier in the same discourse. There are many forms of co-reference and ontologies can be useful when the referent term is a common noun such as in the following example:

> Use of Scoring High is widespread in South Carolina and common in Greenville County... Experts say there isn't another state in the country where... [4]

This is sometimes referred to as *bridging* and relies on knowing semantic knowledge about South Carolina, namely that South Carolina is a state. In Markert and Nissim [2005] they found that in a selection of articles from the Wall Street Journal, 45.7% of all anaphoras used a common noun, they further investigated and found that WordNet as a knowledge source could help in 43.0% of these cases.

One of the uses of ontologies, is the one with which ontologies are now frequently associated: the *Semantic Web*, first defined by Berners-Lee et al. [2001]. This project attempts to encode into the world-wide web *meta-data* describing the objects on the page in a unified way that a computer can handle. This offers advantages such as the ability to create *agents*, which can visit a website and automatically extract information from it. For example, a user looking for a camera would currently have to visit many electronics sites and look at cameras with different technical specifications and compare prices, however if there was meta-data then the agent could automatically visit these sites and extract price and specifications such as maximum zoom, sensor mega-pixels etc. The key underpinning of semantic web is a single, well-defined vocabulary. For this reason, the semantic web requires the definition of many ontologies, which define the terminology used in these specific domains. In the camera example there would be an ontology including elements for such things as "maximum zoom length" and "sensor mega-pixels". The semantic web further requires a single consistent way to access any data in a single ontology or for cross-ontology reference, for this reason they use a single ontological language with a fixed structure. The language designed for the semantic web is OWL, which has since become a popular ontology language for other applications.

[3] WordNet definition

[4] Quote from Markert and Nissim [2005]

Finally, ontologies are often used for the organisation of information. This is especially important in medical applications as the amount of data generated on a topic such as Genetics makes it difficult for it all to be correctly categorised. For this reason, the Gene Ontology (GO) [Ashburner et al., 2000] was created with the goal of creating a single unified system for referencing genes. They performed this by creating a record for each identified gene and giving it a unique ID. This project forms part of the Open Biomedical Ontologies project, which attempts to develop a set of ontologies, based on an agreed set of design principles, in order to facilitate integration between ontologies in different domains of bio-medicine. A similar project was that of the Medical Subject Headings (MeSH)[5] database which was created to list all terms related to medical domains and give them a unique ID, which could be used for indexing documents and articles by their subject. Another project to organise clinical information consistently is Snomed-CT [Stearns et al., 2001], which was designed using to allow consistent sharing of computer records between health care providers. This database was later incorporated into a meta-thesaurus called UMLS, which has increased range and coverage.

Summary

WordNet is one of the most successful examples of an ontology, however it only covers general English and as such there is a significant need to expand it to new domains such as medical terminology and new languages. WordNet only contains a limited number of relationships between terms and for different domains and applications it may be necessary to include new relationships. Many ontologies expand on the structure of WordNet to give a logical framework in which new relationships can be added. Ontologies see a wide range of uses, such as for *co-reference resolution* a key stage in understanding texts or in *information retrieval* as knowing how terms are related is useful to improving both the precision and recall of the system. Finally, the logical information encoded in many ontologies makes them ideal for a number of reasoning tasks such as those in the Semantic Web.

2.2 Extracting Semantic Relations

Although a lot of research on automatic techniques for ontology extraction exists, most of the ontologies I have looked at so far have been entirely or mostly manually created. As such, integrating these automatic techniques into the real-world work-flow of ontology creation is an important issue. For example in order to recreate a general ontology of the same scope as WordNet for a new language it would require producing a similar number of terms which would probably take tens of thousands of man-hours. For this reason there has been a lot of interest in the automatic construction of networks and ontologies and I will now present a survey of the methodologies, which have been used to do this.

[5] Open access and download at http://www.nlm.nih.gov/mesh/

1. NP_h *such as* {*NP, NP ... (or | and)*} *NP*
2. *such* NP_h *as* {*NP ,*}* {*(or | and)*} *NP*
3. *NP* {*, NP*}* {*,*}*? or other* NP_h
4. *NP* {*, NP*}* {*,*}*? and other* NP_h
5. NP_h {*,*} *including* {*NP ,*}* {*(or | and)*} *NP*
6. NP_h {*,*} *especially* {*NP ,*}* {*(or | and)*} *NP*

Figure 2.2: Patterns for extracting hypernym relations from free text as given in Hearst [1992] (NP_h denotes the hypernymous term)

In the discussion that follows I group the existing methods into three groups: *pattern-based extraction* relies on the occurrence of term pairs in the same contexts and uses the words in the context to identify the relation; *distributional clustering* uses the contexts that terms occur in individually and attempts to group semantically related elements based on similarities of these contexts; *term variation* is based on the form of the term and uses similarities between terms to identify, which are semantically related.

2.2.1 Pattern-based extraction

A seminal paper on the topic of relation extraction is that of Hearst [1992], here she dealt with the problem of extracting terms that exhibit the hypernymy relationship, for the purpose of expanding *machine readable dictionaries* (i.e., WordNet). Her approach involved noting that such terms often occurred near each other in stereotypical patterns such as

> The bow lute, such as the Bambara ndang, is plucked and has an individual curved neck for each string

This leads her to conclude that if there is a noun phrase followed by the text "such as" and then another noun phrase she could assume that there existed a hypernymy relationship of the form

> hyponym("Bambara ndang", "bow lute")

She proposed the following method for developing these patterns

1. Decide on a lexical relationship.
2. Collect a set of term pairs known to have this relationship and a corpus, which contains these pairs.
3. Find the places where these terms co-occur.
4. Find commonalities and hypothesise a pattern.

5. Use this pattern to find more term pairs and repeat the process.

Using this method, she found the patterns in figure 2.2 and applied them to encyclopedic text, finding found that 20.4% of results were new to WordNet.[6] Perhaps, the greatest weakness of this work was that there was no way to automate the process or provide a strong comparison between the effectiveness of the different patterns, which perhaps lead to the inclusion of a relatively "weak" pattern (number 6 in figure 2.2), while excluding possibly the strongest result as show in the results of Snow et al. [2004]

NP-like NP_h

A more detailed exploration of Hearst's methodology was done in Cimiano and Staab [2004], in which they used one of the largest corpora currently available: the set of websites indexed by Google. Then by applying Hearst's patterns and a few patterns of their own invention they found they could get a precision of 62.3% and a recall of 45.1%, showing that given enough data this approach can be very effective and complete. Another attempt to reapply Hearst's methodology was performed in Yu et al. [2002], where they worked on extracting relations in a very specific domain, namely that of gene/protein names, and attempted to extract synonyms from the data. Their approach lead them to developing several patterns. Especially they found that lists separated by commas or slashes were often indicative of synonymy as well as more obvious patterns using the phrases "also called", "known as" and "also known as". After applying several filters designed to filter terms based on the structure of their documents and domains they managed to extract synonyms with a precision of 71%. This demonstrates that this methodology is applicable to different relations and domains, however one of the key issues is the ability to properly evaluate the effectiveness. One approach for this was performed in Yang and Su [2007], they used a simple methodology for creating their patterns, by including only the few words that came between the two seed terms, so for example from the seed terms "Bill Clinton" and "president" and the context

Bill Clinton is the elected President of the United States

They would extract the pattern

<#t1#> is the elected <#t2#>

This method allows for the quick extraction of a large number of patterns, but these patterns are far too numerous to be easily applied so they used a metric called *point-wise mutual information* defined as

$$pmi(x, y) = log \frac{P(x, y)}{P(x)P(y)}$$

[6]Given the date of the experiment, the size of WordNet is much smaller at the time of Hearst's experiment than the valued stated in this paper, in fact following the work of Snow et al. [2004], WordNet has been extended to include terms found by a methodology which incorporated that of Hearst [1992], so it seems unlikely as large a value would be obtained, when compared against WordNet 2.0

So if (E_i, E_j) is the event that a pair of terms E_i and E_j occur in a text and p as the event that a pair of terms is separated by the text p then it follows that

$$pmi(p, (E_i, E_j)) = log \frac{occurrences(E_i, p, E_j)}{occurrences(E_i, E_j) \times occurrences(p)}$$

They then evaluated a *reliability* for these patterns as the average PMI for each pattern across all seed pairs. This data was extracted and used to improve the effectiveness of a co-reference resolution system in a similar way to Markert and Nissim [2005], as already mentioned. Another approach was taken in Snow et al. [2004], where they used a dependency grammar to extract their patterns. This has the advantage that the patterns are not affected by small lexical variations as the pattern is based on the syntactic structure of the text, however the text needs to be parsed before it can be used. For parsing they used a specialised dependency parser called MINIPAR [Lin, 1998]. The dependency grammar is also useful as it shows which parts of the sentence are related to other parts so the sentence

I have a brown dog

Is parsed as

Modifier	Category	Head	Type
I	N	< have	subj
have	V		
a	Det	< dog	spec
brown	Adj	< dog	adjn
dog	N	> have	comp

This is also represented in figure 2.3. They then simply chose all of these dependency paths between their seed pairs which occurred between at least 5 pairs of nouns. This method created a large amount of patterns, many of which may be far to general and lead to a lot of false positives, so they then formed their results into vectors and used a statistical classification algorithm to determine if a particular pair is a hypernym or not based on the dependency paths that the pair occurs in (this method is fully described in section 3.4). They found that by doing this they could improve the F-measure from 14.2% using just the patterns of Hearst [1992] to 27.1% with their dependency paths based patterns.

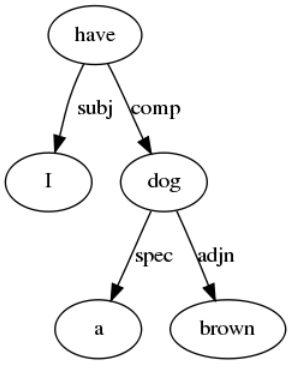

Figure 2.3: Dependency paths of "I have a brown dog"

Parsing the corpus to gain patterns is potentially powerful however it takes a large amount of computational overhead, and for many languages parsers do not exist. Furthermore, using a parser can lead to the phenomenon of *error carried forward*, which means that if the parser makes a mistake, then this will impact the ability of the pattern system to develop patterns correctly. For this reason I shall consider extracting patterns based on the lexical structure of text. One system for extracting information from free text is the WHISK system [Soderland, 1999]. This method starts from a base expression with *wild-cards* and *capturers*; for example to extract two *slots*, the pattern may start as

* (*slot1*) * (*slot2*) *

This rule is essentially meaningless as it matches everything. The WHISK system then proceeds by "adding" terms, i.e., it replaces the wild-cards, "*", with actual matches from the text. So for example consider the text

the price is $ 20

Where "price" is labelled as *slot1* and "20" as *slot2*, then from this example any of the other terms "the" "is" and "$" could be added. This means that the following pattern could be obtained

* (*slot1*) * "$" (*slot2*) *

This process can be repeated by adding more terms or removing wild-cards to gain a very large variety of patterns. For this reason a metric is necessary to decide if the patterns being generated are useful. For this Soderland uses the *Laplacian expected error* defined as

$$Laplacian = \frac{e+1}{n+1}$$

Where e is the number of incorrect matches to the pattern and n is the total number of matches. The system then adds terms one by one until the rule has no errors on the training data or there is no improvement in the Laplacian expected error. This can be seen as a sort of *hill-climbing search*, and as such cannot be expected to produce the true optimum values, but can be effective at creating applicable rules.

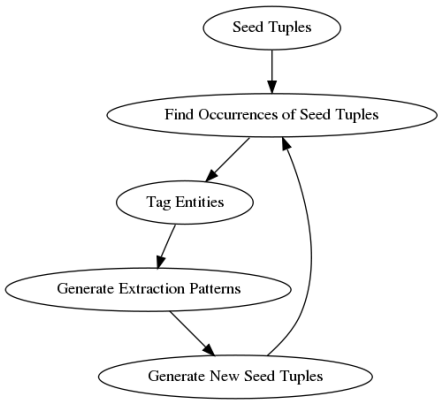

Figure 2.4: Snowball system work flow

Another system which has been used for extracting relations from text is Snowball [Agichtein and Gravano, 2000]. Snowball uses a much simpler basis for extracting patterns, in that

it only takes the text occurring between the pair of terms (similar to the approach of Yang and Su [2007]), but their methodology uses the patterns to extract more seed term pairs, and hence generate more patterns, illustrated in figure 2.4. *Seed term pairs* are a set of term pairs that are used initially as the training data for the process and may be obtained from an existing ontology or by manual input. Snowball takes the seed term pairs and uses those to make patterns, which it then applies to create more term pairs and so forth until some criteria is met. To control this loop it assigns a confidence to each pattern defined as

$$conf(P) = \frac{positives}{positives + negatives}$$

It then defines the confidence in each generated tuple, T, as

$$conf(T) = 1 - \Pi^{P}(1 - Conf(P_i))$$

Where P is the set of patterns which have context which match to the tuple T. The confidences are then iteratively updated as

$$conf(P) = conf_{new}(P) \times W + conf_{old}(P) \times (1 - W)$$

W is a weight value which controls how quickly the confidence value can change. This system is designed to effectively extract a large number of patterns from very few initial examples and was successfully applied to the problem of extracting synonyms from free text by Yu and Agichtein [2003].

One potential method to extract patterns and generalise them is to use a technique called *sequence alignment*, which is widely used in molecular bioinformatics to extract data from sequences of DNA and protein. The classic algorithm for this method is the Needleman-Wunsch algorithm [Needleman and Wunsch, 1970], which is used to compute an alignment between two sequences. For the purpose of this algorithm assume there is a set Σ of symbols (e.g., for DNA analysis $\Sigma = \{A, G, C, T\}$), and there are two sequences A, B of different length, that is $A \in \Sigma^n$,$B \in \Sigma^m$. In addition, there is a special *gap* character denoted $-$, then an alignment of A, B is defined as two sequences A', B', of length k which are identical to A, B respectively except that a number of gaps have been added. For example the sequences

$A =$ `GGCATACTGT`
$B =$ `GGACTATAGT`

Give the following possible alignment

$A' =$ `GGCATACT---GT`
$B' =$ `GG---ACTATAGT`

It is then possible to define a *similarity function*, $S : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{R}$, which says how similar to characters are to one another.

$$similarity(A', B') = \sum_{i=1\ldots k} S(a'_i, b'_i)$$

It is now possible to state that an *optimal global alignment* of A and B is the alignment, A',B' which is maximal with respect $similarity(A', B')$. Needleman and Wunsch [1970] presented a *dynamic programming* algorithm capable of solving this problem in polynomial time, which is still widely used for molecular biology. In Chang and Lui [2001] they use this technique to generate patterns suitable for extracting facts from HTML documents and found alignments to generalise these patterns and improve their extraction. The form of their patterns is in terms of *suffix trees*, which I will not describe here, however their method to stop these patterns from over-fitting to the data is through the use of alignment. For example if there are three patterns in a *multiple alignment* (a generalisation of sequence alignment to more than two sequences), e.g.,

```
a d c w b d
a d c x b -
a d c x b d
```

Where the symbols represent matching elements, then there is a generalised pattern from this alignment: "adc[w|x]b[d]?", that is the 4^{th} elements matches either to w or x and the last match to d is optional. They applied this to a task of extracting document matching a key, and found that by applying alignment to their patterns they improved their retrieval rate from 86% to 97% and accuracy from 86% to 94%. Another approach was taken in Barzilay and Lee [2003], where they used sequence alignment to paraphrase texts. They formed their sentences into, *word lattices*[7], from which they use multiple alignments to find similarities and replace non-matching elements with slots, to create a "slotted lattice". They then found by using these slotted lattices and replacing terms from the slots of one lattice to another they could create paraphrased sentences.

An interesting take on the problem of extracting relationships and the terms between them was taken in Davidov et al. [2007]. They were not interested in extracting a given relationship such as hypernymy, but instead find a variety of binary lexical relations. For example, from a seed set of a list of countries they managed to discover several relationships, which were strongly related to countries, such as *capital-of*, *language-spoken-in* and *president-of*. Their methodology for doing this was:

1. Using a seed of two (or more) examples, automatically obtain other examples belonging to the same class. This can be done by looking for lists e.g., "France, Britain, America" or common words such as "and" or "or".
2. For each concept word, find those terms, which occur close to the concept word, e.g., "Paris" may often occur close to "France".
3. Develop patterns linking these context words.
4. Group patterns based on similarity of their form (e.g., they contain a common low-frequency word), or the similarity of their output (i.e., they produce similar pairs)

[7]This is not to be confused with *lattice* as used in this paper, it is actually similar to the dependancy trees as discussed above

By using this methodology they can extract ontologies on a particular topic, with reasonably high precision and as a bonus the relationships chosen by this methodology are those, which are most commonly used in the chosen text. Hence, ontologies of multiple relations can be developed in an unsupervised manner.

2.2.2 Distributional Clustering

Another group of methods that are often used for identifying terms and forming them into ontologies is known as *Distributional clustering*, which consider the context that a term tends to occur in and then apply clustering to work out, which terms are most "similar". One of the earlier works on this methodology is Pereira et al. [1993], who considered the connection between nouns and verbs. In particular they calculated for each noun/verb pair a probability that that noun could be the direct object of the verb. They first tabulate from the corpus all the occurrences of a particular noun, n, with a specific verb, v, denoting this f_{vn}. They then attempt to find the function

$$p_n(v) = \frac{f_{vn}}{\sum_{v' \in V} f_{v'n}}$$

This function can be thought of as the probability that n is the direct object of v. While this seems readily computable, in fact, given the number of combinations of v and n, the size of the corpus that would be needed is to great to allow for easy computation. They get around this problem by splitting the set of verbs into a set of clusters $\mathcal{C}$, and defining the probability as

$$\hat{p}_n(v) = \sum_{c \in \mathcal{C}} p(c|n) p_c(v)$$

Where $p(c|n)$ is the membership probability of n in c and $p_c(v)$ is v's conditional probability given by the distribution for cluster c. They then use a clustering approach to find these sets of clusters and the probability functions on them. After calculating these values, they gain a better estimate of the function $p_n(v)$ they can then calculate the similarity of two nouns by the use of the *Kullback-Leibner distance*, defined as

$$D(n||m) = \sum_{v \in V} p_n(v) log \frac{p_n(v)}{p_m(v)}$$

By using this methodology they could find class of words that are similar in meaning, for example by using just the verb "fire", they found the following class of nouns

- Gun, Missile, Weapon
- Shot, Bullet, Rocket, Missile
- Officer, Aide, Chief Manager

And also found that the first two classes were closer to each other than the second.

This approach was also taken by Bean and Riloff [2004] and applied to the problem of co-reference resolution. They used the same idea however they did use a pattern development system called Autoslog [Riloff, 1996] to develop patterns for denoting the context of the nouns. They also combined this by using a *Dempster-Shafer decision model* instead of a clustering approach to decide co-references and applying this to the MUC-6 tasks they found significant improvement in result from 57% to 63%.

A closely related method is known as *Latent Semantic Analysis*, which uses the documents which terms occur in to work out which terms are most closely semantically related. Assume there are a set of documents, D, and these documents consist of a set of words, W, then a matrix, M of size $|D| \times |W|$ can be created with

$$m_{ij} = \text{the number of occurrences of word } w_j \text{ in documents } d_i.$$

In order to compare terms by which documents they occur, LSA decomposes the matrix using a technique known as *singular value decomposition*. This is defined (for real matrices) as finding three matrices such that

$$M = U\Sigma V^T$$

Where

- U is orthogonal, that is $UU^T = U^TU = I$
- Σ is diagonal, that is $\sigma_{ij} = 0$ if $i \neq j$
- V is orthogonal

The key is that as U is orthogonal, which means that its columns are *linearly independent*, this is formally defined as if the columns of U are $U = \{u_1 \dots u_n\}$ there does not exist a set of values $\{c_1, \dots c_n\}$, which are not all zero such that

$$0 = c_1u_1 + \dots + c_nu_n$$

For the purpose of this analysis, it is possiblt to think of this method as rearranging the space of document/term co-occurrences into a number of different axis, where each axis represents an independent "aspect" of the data. The values on the diagonal of Σ then express the "importance" of each axis. These values can be used to represent each term based on how it is represented in each axis, in fact the values for each term are given by the rows of $U\Sigma$. These vectors can then be used to compute the similarity of two terms using *cosine similarity* defined as

$$sim(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}.\mathbf{y}}{||\mathbf{x}||||\mathbf{y}||}$$

This method was used in Cederberg and Widdows [2003], where they attempted to extract hypernym relationships using the patterns of Hearst [1992] and then

filtering the extracted term pairs by selecting only those, which have a high similarity according to the LSA method. They reported that by performing this they could increase the precision of their method from 46% to 64%.

Another method for distributional clustering was applied in Widdows and Dorow [2002], they first found all the lists in a corpus, given by the appearance of terms separated by commas and "and"/"or". This allows them to construct a graph where each node corresponds to a term and the nodes are linked if the terms occur somewhere in a list. They then formed "classes" of words from a single term, by adding terms from a "neighbourhood" by the following criteria: Let A be a set of nodes, and $N(A)$ the neighbours of A, that is those terms, which are connected. For each $u \in (N(A) - A)$ define the *affinity* as

$$affinity(u, A) = \frac{|N(u) \cap N(A)|}{|N(u)|}$$

The algorithm then proceeds by adding a node with the highest affinity to A from $N(A) - A$ until the highest affinity is under a certain threshold. They applied this method to a number of classes and found that they got 82% accuracy, for example from the seed word "murder" they extracted {"crime", "theft", "arson", "incest", "fraud", "larceny",... }.

2.2.3 Term Variation

Another method for looking at the similarity of terms is called *term variation*, which works by looking at the form of the actual term and using the similarity of the words in it to deduce if the terms are related. For example it is clear that "cancer of the mouth" and "mouth cancer" are semantically related as they use the same words. In Jacquemin [1999], he defines three main ways that term variation occurs

- **Syntactic Variations:** This is when the content words are the same but the organisation of the terms is different, this includes the addition of *modifications* and/or *coordinate* terms, for example the addition of "hospital-acquired" to "hospital-acquired MRSA" or the coordination in "fresh or dried fruit" versus "fresh fruit". It also includes *synapsies*, which describe small changes to the form of the words, for example the pluralisation of a term.

- **Morpho-syntactic Variations:** Here the words may have the same root, but they may be in a different form, or the syntactical structure of the term may be different. An example of this is the change from *N of N* to *NN* as in the "mouth cancer" example above, or change between the use of a noun and its adjectival equivalent, for example "saline solution" versus "salt solution".

- **Semantic Variations:** These cover the case where one of the content terms has been changed, for example "maize flour" and "wheat flour" are related as there is a relation between "maize" and "wheat".

In addition to this there are also compound variations where several variations may link one term to another. This methodology was applied to the extractions of hypernym relationships in Morin and Jacquemin [2004], where they created a set of transformational patterns to describe the common form of variations and then applied them to a corpus. They found that most variations gave a very high precision in identifying the relationships between specific terms and reported precision values of 93.9% for syntactic variations, 71.2% for morpho-syntactic variations and 80.5% for semantic variations. A similar attempt was used to apply this method to synonym link discovery in Hamon et al. [1998], for this they used a much stricter requirement that the content terms must be the same or known to be synonymous (they used WordNet to decide this), and they found a much lower result of 37%, but they point out that they had problem differentiating between synonymous term pairs and those which have other relationships, such as hypernymy, meronymy, antonymy etc. Another attempt to apply this methodology was explored in Bodenreider et al. [2001], where they looked at only modification by a single adjective, and applied this to the medical thesaurus UMLS. They found that the relations split by modification were about 4% indicative of synonymy, 43% indicative of hypernymy, 24% indicative of "sibling" relation (both direct hypernyms of other terms), about 1% some other relation, and 27% unrelated. They also found that for most terms, the split between the different relations was quite high, which suggest that term variation is not great for differentiating kinds of relationship.

The methods used in here require a large amount of manual analysis to create patterns, which describe the transformations relating one term to another. A simpler metric for relating terms was described in Nenadić et al. [2002], here they viewed terms as a bag of words, so for example the term "orphan nuclear receptor" is viewed as a set $t_1 = \{$ "orphan", "nuclear", "receptor", "orphan nuclear", "nuclear receptor", "orphan nuclear receptor" $\}$. They then describe the similarity of two terms by

$$LS(t_1, t_2) = \frac{|t_1 \cap t_2|}{|t_1| + |t_2|}$$

They found this to be useful in combination with metrics based on extraction patterns (à la Hearst [1992]) and distributional clustering. However this simple approach cannot be expected to represent many of the variations seen in Jacquemin [1999]. A more complex automatic method was presented in Ibekwe-SanJuan [1998], here she grouped these transformations into a small number of operations

- **M-Sub(x,y)** The replacement of a single modifier x with a new modifier y.
- **H-Sub(x,y)** The replacement of the head of a term, x, with a new head y.
- **L-Exp(x)** The insertion of a term x at the beginning of the term.

- **R-Exp(x)** The insertion of a term x at the beginning of the term.
- **Ins(x)** The insertion of a term x in the middle of the term.

By using these transformations, she could describe a path that transforms one term into another, then she takes a set of terms and then forms these into a graph based on these terms and the relationships given between them. These can be clustered to create sets of nodes, which have similar terms.

2.2.4 Term extraction

One of the methods I have not really mentioned yet is the identification of the terms of interest from free text. Most of the papers already mentioned achieve this by parsing the sentence and identifying the term as the noun phrase in the parse tree, in English often dropping articles if present. This method is effective and several systems have been developed that allow for doing this such as GATE [Cunningham, 2004], however there are some alternative approaches, which are worthy of consideration. In Bikel et al. [1997], they present a method based on using a Hidden Markov Model to extract terms that would be of interest to a specific task, and they found that their system scored a a high performance of 90-93%. This methodology is particularly useful when attempting to identify certain types of terms in a domain, for example Collier et al. [2000] applied this to the specific problem of extracting gene names, for which a tagger may fail as it would not have many of the terms in its dictionary and their system scored 72.8% F-Measure on their corpus. Another method for selecting the terms that are most relevant is that of Frantzi et al. [2000], which they call *C-value* this selects terms based on their number of occurrences and also their number of occurrences in a nested form. Firstly they tag the corpus with parts of speech and then they identify the terms by using the following patterns

- Noun$^+$ Noun
- (Adj | Noun)$^+$ Noun
- ((Adj | Noun)$^+$ | ((Adj | Noun)* (NounPrep)?) (Adj | Noun)*) Noun

Where NounPrep refers to a class of prepositions occurring in terms, such as "composition of tears". Once they have done this they select terms by the *C-value* defined as

$$C\text{-}value(a) = log_2|a|(f(a) - \frac{1}{|T_a|}\sum_{b \in T_a} f(b))$$

Where a is the term, $|a|$ the length of the term, $f(a)$ the number of occurrences of the term in the corpus and T_a the set of terms of which a is a sub-string, e.g., if $a =$ "real time" then "real time systems" $\in T_a$. The C-value is useful as it increases the importance of terms which occur frequently in nested forms and are likely to be of more importance than others.

Summary

Extracting relationships between terms is the task of deciding if a particular pairs of terms are related. There are three main approaches to doing this: *Patterns*, *Distributional Clustering* and *Term Variation*. *Patterns* were first suggested by Hearst [1992] and involve finding the terms in the same sentence and in some "pattern" that is suggestive of a particular relation. These patterns can be found either manually or automatically from a given a set of seed term pairs. However as patterns rely on finding the two terms in the same context, this limits the recall and ambiguity in the text can cause errors in the extractions. On the other hand, patterns have the advantage that they can be specialised for different relationships. *Distributional clustering* is the process of seeing how a pair of terms are related by finding similarities in the contexts they occur in. This method then requires a mathematical approach to determine the clusters of terms which have a similar distribution of contexts. Distributional clustering does not require that the terms occur in the same sentence or even in the same document, hence it generally has a higher recall than pattern based methods, however it is very difficult from distributional clustering to work out the nature of the relationship between the terms, meaning that distributional clustering is not suitable for extracting specific relationships such as if "X is a causal agent of Y". Finally *Term variation*, is a method that examines the structure of the term and uses this to determine new variations of the term, of which there are a number of such possible variations, from simple acronyms to more complex modification. Term variation often has very high precision, however as it relies on the structure of the terms being similar it cannot help to identify relationships between terms with no similarity, for example, the synonymous terms "tuberculosis" and "consumption". Term variation is also strongest for finding if two terms are hypernymous however it can prove useful for some other cases as well, e.g., consider the name of a virus like "HIV", then there is normally a causal agent relationship to the term made by adding "infection", i.e., "HIV Infection"

Method	Precision	Recall	Applicability
Patterns	OK	Limited	Produces specific results for any relationships
Distributional Clustering	OK	Good	Only produces a concept of "semantic relatedness"
Term Variation	Good	Poor	Strongest for hypernymy, some use elsewhere

I also briefly looked at the problem of extracting terms from free texts, mostly this is done by identifying the noun phrase through the use of a chunking methodology. If it is required to identify which terms are more important the C-Value of Frantzi et al. [2000] or the HMM based approach of Bikel et al. [1997] can be useful.

2.3 Forming Ontological Structures

I have so far looked at methods capable of extracting relationships from text or determining the semantic similarity of terms, however this information would generally be better if they could be stored in some kind of organised structure. A clear example of the need for this can be seen if we consider the results of hypernym relationship extraction by the use of Hearst's pattern [Hearst, 1992] and the organisation of terms in WordNet. Hearst's patterns extract a binary decision of whether a term pair is or is not hypernymous, where as WordNet has sets of synonyms organised in a hierarchical structure. Naïvely adding the binary extractions from Hearst's patterns into the structure would cause a number of problems, in that there may be missing links that would be suggested by the transitivity of the hypernymy relation (for example the system might find that "cat" is a type of "mammal" and "mammal" is a kind of "animal" but not that "cat" is a type of "animal") or other similar problems occurring from noise in the extraction procedure. For this reason I will examine some methods that can either extract the form of the structure in a single attempt or form the extracted result into a given structure

In Cimiano et al. [2004], they use a process called *Formal Concept Analysis* as a basis to form hierarchical structures, which starts by parsing sentences and associating specific terms with verbs, which the terms occur as the subject or object of. This gives a set of concepts, which are associated with a given term, so for example the term "car" might have the concept set { "driveable", "rideable", "bookable" }, they then form this into a lattice structure[8], such as the structure shown in 2.5. A hierarchy can generally be reliably formed, giving a taxonomy. This method has exponential time but the advantage that it is possible to trace the reasoning for the inclusion of each term in a particular class. In Cimiano et al. [2005] the same authors compared their work to a more standard approach of hierarchical clustering, which can be used to form a hierarchical structure. This approach relies on calculating a similarity function between different terms, this can be done in many ways, but they used a cosine

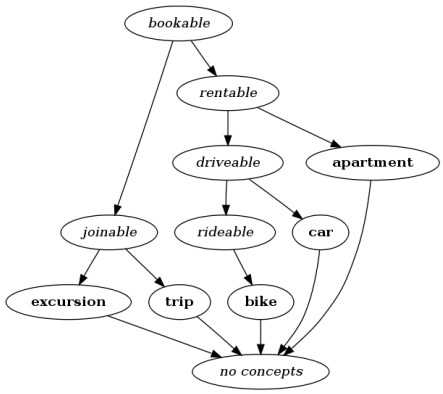

Figure 2.5: A formal concept analysis lattice. Terms in **bold** and concepts in *italics*

[8] A *lattice* is a set, S, with a *partial order* on the set and the set contains a unique *supremum* and *infimum*, that is two elements $\top$,$\bot$ such that $\forall s \in S$, $\bot \leq s \leq \top$.

metric, which when applied to concept sets becomes

$$sim(a,b) = \frac{|C_a \cap C_b|}{|C_a||C_b|}$$

Where C_a,C_b are the concept sets of a and b respectively.Then for a set of terms P, define a *split* of this as the set as a pair of sets Q, R, such that $P = Q \cup R$ and $\emptyset = Q \cap R$ and that Q, R are minimal with respect to

$$\frac{1}{|Q||R|} \sum_{q \in Q, r \in R} sim(q,r)$$

By repeatedly applying this methodology a hierarchy can be obtained like in the example of figure 2.6. These clusters can then be used to define a taxonomy by including the terms, which have the same set of concepts into the appropriate place in the hierarchy. This method has the advantage that it is much more computable, in fact its complexity is $O(n^2 log(n))$, however it doesn't produce an outcome that is as easily traced.

These methods are certainly suitable for extracting hierarchical structures, however for more general combinations of multiple relations, something that can be more readily generalised is required. In Snow et al. [2006] they use an approach which forms the problem into a logical framework. Initially they used the methodology of Snow et al. [2004], which I have already discussed to extract a large number of hypernym pairs from a corpus. They then stated that a taxonomy should have the following property known as *transitivity*

$$H_{ij}^m \wedge H_{jk}^n \rightarrow H_{ik}^{m+n}$$

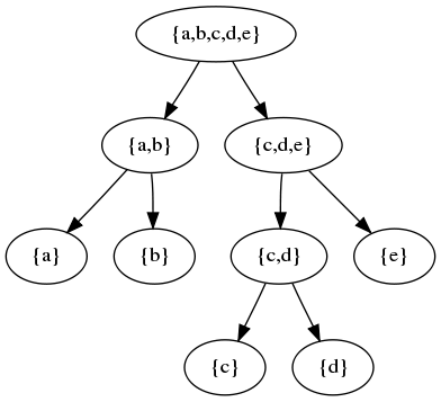

Figure 2.6: Hierarchically clustering a set by applying splits

Which states that if term i is a the n^{th} ancestor of j and k is the n^{th} ancestor of k, then i is the $(n+m)^{th}$ ancestor of k. They then define a taxonomy, T, as a set of pairs on a domain of objects, and state that based on the information extracted from their pattern extraction system they have a probability value, $P(R_{ij} \in T | E_{ij}^R)$, which denotes the probability that a relation, R_{ij}, is in the taxonomy based on the linguistic evidence E_{ij}^R. By applying some independence assumptions and using Bayes Rule they state their problem as that of finding the optimal network $\hat{T}$ given by

$$\hat{T} = argmax_T P(E|T)$$

Where E denotes all evidence and $P(E|T)$ is given by

$$P(E|T) = \prod_{R_{ij} \in T} \frac{P(R_{ij} \in T|E^R_{ij})P(E^R_{ij})}{P(R_{ij}) \in T} \prod_{R_{ij} \notin T} \frac{P(R_{ij} \notin T|E^R_{ij})P(E^R_{ij})}{P(R_{ij}) \notin T}$$

The value of $P(R_{ij} \in T)$ and $P(R_{ij} \notin T)$ can be set to make the model more or less likely to include relations, however it is safest to set them to 0.5, unless there is a strong reason that the model is over or under-fitting. They then defined the change by adding a relation, R_{ij} to a taxonomy T, as

$$\Delta_T(R_{ij}) = \frac{P(R_{ij} \in T|E^R_{ij})(1 - P(R_{ij} \in T)}{(1 - P(R_{ij} \in T|E^R_{ij}))P(R_{ij} \in T)}$$

They also define the *implied set* $I(R_{ij})$ as the set of terms given by applying the transitivity rule, that is that

$$\begin{array}{ll} R_{jk} \in I(R_{jk}) & \\ R_{im} \in I(R_{jk}) & \text{if } R_{ij}, R_{jm} \in T \cup I(R_{jk}) \end{array}$$

The algorithm[9] starts with $T_0 = \emptyset$ and then finds the next step as $T_{n+1} = T_n \cup I(R_{ij})$, where $R_{ij} = argmax \prod_{R \in I(R_{ij})} \Delta_T(R)$. The algorithm continues generating the next T_n, until it finds that $P(E|T_{n+1}) < P(E|T_n)$. This algorithm may not find an optimal solution, however its complexity is polynomial. The authors found that by applying this methodology they could improve their precision from 46% to 64% on a set of 10000 extracted links. This methodology was also applied in Yates and Etzioni [2007] to the problem of extracting sets of synonyms, here they used data extracted by the use of string similarity metrics combined with distributional similarity data, to determine the probability of a single relation R_{ij}. As synonyms are naturally grouped into sets, instead of relations one by one, they merge synonym sets. Merging is gaining a new set from S from S_1 and S_2, by $S = S_1 \cup S_2$, which can be considered identical to adding all relations from the set $\{R_{ij}, R_{ji} | i \in S_1, j \in S_2\}$. Again they show an increase in F-measure from 57% without merging synonym sets to 68%.

These systems are good for dealing with transitivity restrictions, which are commonly found in ontology construction, but these methods are not suitable for more complicated restrictions (for more discussion of this see sections 4.4 and 4.5). A model which is capable of dealing with a much wider range of logical restrictions was used in the SOFIE system of Suchanek et al. [2009], where they attempt to extract ontologies with multiple relations from free text. Their system is based on pattern extraction, and they turn each of their pattern extractions into a fact of the form:

$$patternOcc(\text{“X went to school in Y”}, Einstein, Switzerland)$$

They then presented a rule that allowed them to understand the usage of each term

$$patternOcc(P, X, Y) \wedge R(X, Y) \rightarrow expresses(P, R)$$

[9]This algorithm is in fact a special case of Algorithm 4.3.2.3

That is if pattern P appears between X and Y and we know there is a relation R between X and Y, then it is possible to conclude that pattern P expresses a relationship R. Similarly they have a reverse rule that says if they see X and Y in pattern P and it is known that pattern P expresses R they conclude there is a relationship between X and Y. These statements can all be formed and then treated as a *MAX-SAT problem*: if there are a set of statements S then the maximum satisfaction of them is an assignment of true/false to each term such that the maximum number of statements in S are true given the assignment. This problem is well-known and there exist many algorithms to solve it. An advantage of this approach is that it is relatively easy to add more restrictions, in their paper they present one such restriction

$$R(X,Y) \wedge functional(R) \wedge different(X,Z) \rightarrow \neg R(X,Z)$$

Which states that if R is a *functional* relation, i.e., it is a many-to-one mapping , for example if $R = bornIn$ it is clear that it is not possible to have two objects as a person can only be born in one place. It is also trivial to see that other such restrictions could easily be added to the statement problem such as the transitive axiom of Snow et al. [2006] which would translate to

$$R(X,Y) \wedge R(Y,Z) \rightarrow R(X,Z)$$

The disadvantage with this methodology is that MAX-SAT is a NP-hard problem and as such robust algorithms are required to solve the problem. Furthermore, the MAX-SAT problem can only deal with restrictions, which can be grounded to prepositional logic, and as such this methodology can't handle many of the first-order logic statements used in an ontology language such as OWL.

2.3.1 MAX-SAT

As demonstrated in the works of Snow et al. [2006] and Suchanek et al. [2009], it is possible to consider the problem of forming some arbitrary relation extractions into a consistent network as one of finding a satisfaction of the constraints on the network. In general these problems are called *constraint satisfaction problems* and a survey of some of the approaches that can be taken is given in Kumar [1992]. He associates most of the solution methods to these problems as a form of tree search, as each constraint has several possible assignments that satisfy it, hence each possible assignment represents a different branch. However as choosing one assignment can lead to contradictions with other constraints, it is necessary to backtrack frequently to solve most problems. He states that this causes many problems to be intractable and as such a way to guide the search towards a better solution is required. This is a wide subject and I shall discuss one of the most general frameworks for solving this type of problem.

One formalisation, which I have already discussed is MAX-SAT, we shall now formally define it.

Definition: Suppose there is a set of variable $P = \{p_1, \ldots p_n\}$, call a mapping from P to $\{0, 1\}$ an *assignment*. Define a *clause* as a set $\{q_1, \ldots q_n, \neg r_1, \ldots \neg r_m\}$ where $q_i \in P$ and $r_i \in P$ for all i, and say a clause is *satisfied* by an assignment a, if there is some i such that $a(q_i) = 1$ or $a(r_i) = 0$. Suppose there is a set of clauses C, a *maximal satisfaction* of C is a set $C' \subseteq C$ such that there exists some assignment, a, for which all clauses of C' are satisfied and C' is the largest subset of C for which such an assignment exists.

This problem is *NP-hard* in most cases and several methods have been suggested to solve. There is also a variation of this problem called *Weighted MAX-SAT*, which assigns a weight to each clause and defines the optimal solution as the one, which has largest sum of the weights of its satisfied clauses. A simple approach to both problems is *branch and bound*, which attempts to find the assignment by choosing an assignment on a specific variable, and then by branching on each assignment to each variable a search space is formed. It is also possible to split the clauses into three groups by a partial assignment, that is those which are satisfied by the partial assignment, those which are not satisfied but may be, that is still have unassigned variable, and those which cannot be satisfied by the partial assignment, and it should be clear that this forms a bounding condition to the search space. This means that once a solution has been found that satisfies n clauses, it is possible to prune all branches of the search space, which cannot satisfy n clauses. This algorithm can be further improved by the variable selection of Davis and Putnam [1960], where they choose the variable at each branch by choosing a variable from the unsatisfied clause with the least unassigned variables.

A popular approximation method for MAX-SAT is the method of Selman et al. [1992], known as *GSAT* or *WalkSAT*, this is an approximation method and works as follows

1. Start with a random assignment.
2. For each variable calculate the "gain" of changing the assignment to that variable, that is the number of clauses that would be satisfied by flipping that variable minus those that would no long be satisfied.
3. Go to step 2, unless this is a local maximum (no flip has positive gain).
4. Repeat with a different random assignment.

This algorithm does not guarantee being able to find the maximum satisfaction, however it is robust and efficient at finding a solution [10]. A survey of algorithms for MAX-SAT is given in Gu et al. [1996].

Summary

Relationship extraction systems generally give a binary decision of whether two particular terms are related, however ontologies have a specific structure. As the

[10] Algorithm 4.3.7.9 is an adaption of GSAT

relationship extraction system generally produces noisy and incomplete data, it is highly unlikely that the result will be consistent with the structure of the ontology. One approach is to design the system to extract an entire ontology by the use of clustering, this can be done however the structures produce can only be exact covers or hierarchies and as such are only suitable for a few relationships such as hypernymy and synonymy. To deal with more complex relationships and the interactions of multiple relationships, it is necessary to rely on logical axioms that describe the structure of the ontology. This can be done with special algorithms or by the use of an existing constraint satisfaction problem algorithm such as those developed for *MAX-SAT*. In both cases this leads to an *NP-hard* problem and hence strong algorithms are needed to produce optimal/good solutions.

Chapter 3

Extracting relations

My first goal is to be able to extract a variety of different relations and to do this I choose to base my method on that of Hearst [1992], as this methodology can extract any kind of relationship from a text. However I do not wish to be restricted to a manual definition of patterns, as these patterns may vary widely for different domains and relationships. I make a key assumption throughout this section that a "term" is a set of words that uniquely identifies a single concept. This is not in general true as many terms are *polysemous*, i.e., they have several different meanings. Depending on the domain, for which related term pairs need to be extracted, it may be the case that nearly all terms uniquely identify a single concept or there is some need to incorporate a sense identification method into the process.

3.1 Patterns & Generalisation

I shall start with a simple problem that needs to be attacked before I approach the problem of extracting information from text and that is the problem of tokenization. This is the methodology of splitting English text up from a sequence of characters into a sequence of words. For the most part in English it is possible to use the simple methodology of splitting words by using white-space and punctuation with exceptions for the following cases

- Apostrophe in proper names, i.e., "O'Reilly"
- Acronyms separated by periods, i.e., "I.B.M."
- Numbers using comma and period separators, i.e., "1,000.00"
- Email addresses

This leads to the ability to write a very simple tokenizer, that splits by punctuation and white-space unless it sees one of the special cases. I now define patterns based on the words, which are treated as *symbols*.

3.1.1: Definition Say there is a set of symbols Σ', then define a special symbols $*$. I define a set $\Sigma = \Sigma' \cup \{*\}$, and a pattern as a sequence from the set Σ^*.

3.1.2: Definition Suppose there is a pattern $p \in \Sigma^*$ and a sequence of characters s of length n, with $s_i \in \Sigma'$. It is said that p *matches* the sequence s, if there exists some i, with $1 \leq i \leq n$, such that for all j, with $0 \leq j < |p|$, either $p_j = s_{i+j}$ or $p_j = *$.

These definitions give a basic description of a pattern and how it matches, normally I assume Σ' is the set of all non-white-space sequences of characters, and as such I can make patterns that match sequences of words. For example the pattern

* is a * , which

Can match several contexts such as "a cat is a mammal, which", and "a mammal is a animal, which". Assuming there is a given set of *seed term pairs* that are examples of the given relationship then patterns can be simply extracted from text by the following methodology

3.1.3: Definition If there is a sequence of symbols s, and a symbol σ, then define the *(m,n)-base set* of patterns as

$$(m,n)\text{-}base\text{-}set(\sigma) = \{s_{i-m} \dots s_{i-1} * s_{i+1} \dots s_{i+n} | s_i = \sigma\}$$

Similarly define the *(l,m,n)-base set* of a pair of symbols σ_1, σ_2 as

$$(l,m,n)\text{-}base\text{-}set(\sigma_1,\sigma_2) = \{ \quad s_{i-m} \dots s_{i-1} * s_{i+1} \dots s_{i+m} * s_{j+1} \dots s_{j+n} \\ | s_i = \sigma_1, s_j = \sigma_2, j - i = m + 1\}$$

This is actually a simple definition that states that a base set of a pair of symbols is in fact just found by finding the places in the text where the occurrences are exactly m apart, and then creating a pattern by replacing the chosen symbols with wild-cards elements, for example from the context

...types of felines such as cheetahs and lynxes...

If we chose as our pair of symbols (words), "feline" and "cheetahs", then we have the (2,2,2)-base pattern as

types of * such as * and lynxes

Then extracting these patterns gives the first approach for extracting information from raw text. Figure 3.1 shows example of some of the patterns that can be extracted from text, using parameters of $(2, 8 \leq, 2)$, however these patterns are often very specific to the context that they occurred in, including terms such as "Ehlers" and "MTB". This means that these patterns are very unlikely to be reusable.

- an established * and clear clinical symptoms , such as * , Ehlers
- an underlying * , such as an * or an
- forms of * (TB) such as * (MTB
- forms of * of unknown cause, such as * and juvenile
- tuberculosis (*) such as * (MTB
- types of * , such as * (HD

Figure 3.1: Examples of base pattern extractions

3.1.4: Definition Define the *match set* of a pattern p as the set

$$match\text{-}set(p) = \{s \in \Sigma'^* | p \text{ matches } s\}$$

Furthermore define a partial order on patterns by

$$p \leq p' \leftrightarrow match\text{-}set(p) \subseteq match\text{-}set(p')$$

3.1.5: Lemma If p and p' are two patterns then $p \leq p'$ if and only if $|p| \geq |p'|$ and there is some value i such that for all $1 \leq j \leq |p'|$, $p'_j = p_{i+j}$ or $p'_j = *$.
Proof: The condition of the lemma states that "p' is equal to a sub-sequence of p, except any number of elements have been replaced by *", it should be clear that any such pattern matches at least the same sequences. If we assume that this condition is not true, i.e., for some i, j $p'_j \neq *$ and $p'_j \neq p_{i+j}$, then if $s \in \Sigma'^*$ is a sequence that matches p and p', which must exist as Σ'^* contains all possible patterns and hence $match\text{-}set(p) \neq \emptyset$, we can conclude that $p_{i+j} = *$. Hence by replacing the element of s corresponding to p'_j truncating the match we can get a sequence s' which matches p but not p', hence $p \not\leq p'$.□

This states that some patterns can be said to be "more general" or "less general" than other patterns, by the criteria of how many matches they could make on all possible inputs, and that this is actually just the result of replacing symbols from one expression by the wild-card *. This allows for simply generalising patterns by replacing terms with *, hence from a base pattern 2^n generalised patterns can be developed, where n is the number of non-wild-card symbols in the pattern. As this is clearly a very large number a way to select the patterns, which are most useful, is desirable. I will now define the patterns in they way I wish to use them

3.1.6: Definition An *extraction pattern* is a pattern with at least two wild-cards, * and I shall denote two of these wild-cards, $*_1$ and $*_2$. These are called the *extraction wild-cards*

This definition allows it to be stated, which two of the wild-cards, are used to extract any relevant terms. I now present a method, which can be used to extract results, this method starts not from the base patterns but instead starts from a most general pattern

3.1.7: Definition The $(l, m, n)_{1,2}$-*supremum* pattern is of the form

$$\underbrace{*\ldots*}_{\text{l times}} *_1 \underbrace{*\ldots*}_{\text{m times}} *_2 \underbrace{*\ldots*}_{\text{n times}}$$

Similarly the $(l, m, n)_{2,1} - supremum$ pattern is the same except the extraction wild-cards are switched. Call the (l, m, n)-*supremum pattern set* the set given by

$$\{(i, j, k)_{1,2}\text{-}supremum, (i, j, k)_{2,1}\text{-}supremum | 0 \leq i \leq l, 0 \leq j \leq m, 0 \leq k \leq n\}$$

I now present a pattern generation algorithm.

3.1.8: Algorithm
Input: A corpus, that is a sequence of symbols s, an evaluation function e, and pattern boundary numbers (i, j, k), and an iteration limit n.
Output: A set of patterns.

1. Initialise P as the (i, j, k) supremum pattern set.
2. While P is not empty and the iteration limit n has not been reached
   (a) Select $p \in P$ which is maximal in P relative to $e(p)$, and has not already been selected.
   (b) Find all matches in s to p, and for each case where a wild-card in p matches to a symbol σ add a new pattern p', which is identical to p except the wild-card has been replaced by σ. Remove p from P.

This algorithm covers the search space as illustrated in figure 3.2, and can generate most of the high-quality patterns given a good evaluation methodology for the patterns, however as this system covers every possible variation of the patterns the the search space is far too large to be tractable. So it is necessary to find a way to cover this search space more efficiently by either prioritising "better" patterns or by skipping those patterns which are too similar to existing patterns.

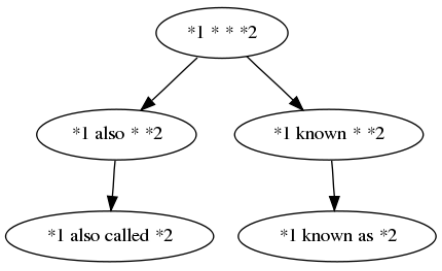

Figure 3.2: Example search space of algorithm 3.1.8

3.1.1 Evaluation Functions

One of the key issues with pattern generation is that it is easy to generate a very large number of patterns, however there is no clear way of evaluating their effectiveness, for this some kind of metric is needed. There are a number of things which can be measured:

- The total number of sub-sequences a pattern matches
- The number of these matches which correspond to known term pairs
- The number of term pairs matched by some sub-sequence.

I will call these the *total matches*, *correct matches* and *omitted terms*. One of the simplest metrics is to simply choose patterns, which have the highest percentage of correct matches to omitted matches, call this the precision of the pattern

$$e_{precision} = \frac{\#\text{Correct Matches}}{\#\text{Total matches}}$$

This metric is good but it favours patterns, which are too specific, for example a pattern that matches a single correct subsequence would have 100% precision. As such I will also define the recall of the pattern as

$$e_{recall} = \frac{\#\text{Term Pairs Matched}}{\#\text{Term Pairs}}$$

This metric is also flawed as in this case supremum patterns must have the highest recall. As such it is desirable to find a metric that gives a balance between the two results, I use the commonly used metric called *F-Measure*, which is a *weighted harmonic mean*, that is

$$e_{F\text{-}Measure} = \frac{1+\alpha}{\frac{1}{e_{recall}} + \frac{\alpha}{e_{precision}}} = \frac{(1+\alpha)e_{recall}e_{precision}}{\alpha e_{recall} + e_{precision}}$$

This metric needs a parameter α which sets how much to value precision over recall (or vica versa if $\alpha < 1$). This metric also has the property of only giving 100% if both recall and precision are 100% and 0% if either are 0%.

While F-Measure is often useful, the problem of finding a suitable value for α is difficult and as such it would often be preferable to have a metric that gives a clearer guarantee of finding strong patterns. Define the *support* of a pattern by the number of term pairs it matches, the support threshold as a value n, and then the minimum-support metric is given as

$$e_{min-support} = \begin{cases} e_{precision} & \text{if } support > n \\ 0 & \text{otherwise} \end{cases}$$

This metric has the advantage of being simpler, and choosing only patterns that are sufficiently general, while still valuing those patterns which are most accurate.

Summary

Patterns can easily be defined on tokenized texts and by introducing a wild-card symbol which matches any single token, these patterns can be generalised in a number of ways. Furthermore, by looking at the set of all possible sequences of tokens a pattern can match it is possible to define a match set, which leads to a *partial order* on patterns. I then present an algorithm, which starts with the most general pattern, that is the pattern consisting of only wild cards, and develops a more specific pattern by replacing wild cards with terms from some corpus. I then presented two metrics, which are capable of evaluating the effectiveness of a pattern, taking into account both its accuracy and its coverage.

3.2 Rules and Join-sets

3.2.1 Rules

One of the problems with the patterns so far has been the inability to handle multi-word terms; to handle this effectively it is required that there are identified boundaries of the terms. In addition a more complex rule language is required to handle these terms. I approach the goal of finding terms as that of finding noun phrases, and this requires a *part-of-speech tagging* of the corpus. My method starts by using the well-known statistical tagger BRILL [Brill, 1992], then following the approach of Frantzi et al. [2000] I use the following simple pattern to identify the areas of interest, henceforth known as *entities*

$$entity = (NN|JJ|NNS|NNP|FW|NNPS|JJR) * (NN|NNS|NNP|NNPS)$$

For reference these tags are

- NN: A singular noun
- NNS: A plural noun
- NNP: A proper noun
- NNPS: A pluralised proper noun
- JJ: An adjective
- FW: A prefix, such as “pre-” or “non-”
- JJR: An adjective in comparative form

Note I include JJR as terms such as “lower” or “upper”, are often found in the dictionary form of terms, but not JJS, which means an adjective in superlative form, such as “lowest”, as this rarely occurs in dictionary forms. I then define a rule language based on a subset of the SRL language[1]

[1] Available at http://srl-editor.googlecode.com/

3.2.1: Definition Define a rule as a sequence of matchers where the matchers are

- A literal: Denote this as `"`σ`"`. This matches only the symbol σ.
- A word region: Denote this as `words(`n`,`m`)`, where $n, m \in \mathcal{N}$, this matches a sequence of symbols, s, if $n \leq |s| \leq m$
- An entity: Denote this as `name()`, this matches a sequence of symbols which constitute an entity.

As before say a rule matches a sequence of symbols if each of its matchers match the sequence when applied in order.

For example taking the pattern

`*`$_1$ `*` such as `*`$_2$

Gives us a rule

```
:- name() words(1,1) "such" "as" name()
```

I quickly make a few definitions to simplify the operations with these rules.

3.2.2: Definition Denote the i^{th} matcher of a rule as r_i, and the *length* of the rule as the number of matchers it contains, denoted $|r|$. Two rules are equal if their *match-set* as in definition 3.1.3 is the same. A rule is *well-limited*, if its first and last expressions are an entity, a literal or a `words(m,n)` element with $m = n$ and $n \neq 0$.

3.2.3: Lemma For every rule r there is an infinite set $eq(r) = \{r' | match\text{-}set(r) = match\text{-}set(r')\}$ and there is a unique well-limited element $r_s \in eq(r)$ such that $|r_s| < |r'| \, \forall r' \in eq(r)$, this rule is the *simplified form* of r.
Proof: This is based on the fact that an element `:- words(`n, m`)` has the same match-set as `:- words(`n', m'`) words(`n'', m''`)` if $n' + n'' = n$ and $m' + m'' = m$. From this it is clear that two rules have the same match-set if their literals and entities are in the same order and the sum of all the consecutive word regions are equal. The simplified form is then the one which has no consecutive word regions and no `words(0,0)`, and the set $eq(r)$ is infinite as it is possible to include any number of `words(0,0)`, without affecting the match-set.□

These two statements allow us to talk about rules as being unique elements based on their match-set, which helps cut down the amount of work necessary. For example, the rule

```
:- words(1,2) name() words(0,1) words(2,3) "literal" name()
```

Can be simplified to the following form

```
:- words(1,1) name() words(2,4) "literal" name()
```

Without affecting the set of sentences it could match.

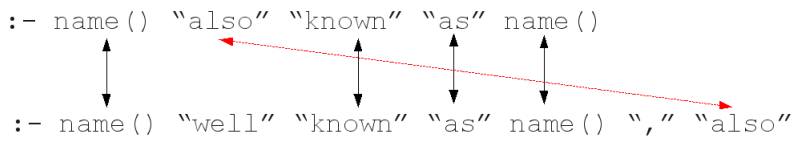

Figure 3.3: Example of an alignment

3.2.2 Join-set

These more complicated rules could be used with the methodology I have already discussed, however I will present a more complex but efficient method based on *join-sets*.

3.2.2.1: Definition If there are two rules, r_1 and r_2 and there is a partial order on these elements as in definition 3.1.3, then a rule, r_j, is a *join* of r_1 and r_2 if it holds that $r_1 \leq r_j$ and $r_2 \leq r_j$ and there does not exist some rule r', with $r_1 \leq r'$ and $r_2 \leq r'$ and $r' < r_j$. Call the set of all joins of r_1 and r_2 the *join-set* of r_1 and r_2.

It is now necessary to find an easy way to calculate the join-sets of a pair of rules. Fortunately for the form of rules I presented before, this is not exceptionally hard, if the method focuses on the literals of the expression.

3.2.2.2: Definition Define an *alignment* of two rules, r and r' as a set of pairs $A = \{(i, j)\}$, with $1 \leq i \leq |r|$ and $1 \leq j \leq |r'|$, and there is no two pairs $(i, j) \in A$ and $(i', j') \in A$ with $i = i'$ or $j = j'$, or $i > i'$ and $j < j'$, and for all $(i, j) \in A$ r_i is the same as r_j. Furthermore if r_i is an entity expression, then $(i, j) \in A$ for some j and similarly if r_j is an entity expression, then $(i, j) \in A$ for some i.

This states that an *alignment* is a mapping of the expressions such that the entities match up, and this match does not contain any "crossing" matches, that is if the third element of the rule matches to the fifth element of the other rule, then the fourth element must match to an element after the fifth of the other rule. In figure 3.3 there is an example of an incorrect matching between the two "also"s as this matching crosses other matches. I can now derive a result about the relationship between joins and alignments

3.2.2.3: Definition Define the *span* of a rule, r or sub-sequence of a rule as a pair of integers (m, n), given by $m = min_{s \in match\text{-}set(r)}|s|$ and $n = max_{s \in match\text{-}set(r)}|s|$. Denote that as

$$(m, n) = span(r)$$

For rules and

$$(m, n) = span(r, i, j)$$

Sometimes I need the span of the sub-sequence of r, from the i^{th} element to the j^{th}, excluding the i^{th} and j^{th} element. Hence, define the *co-span* of two rules as

$$\begin{aligned} \textit{co-span}(r, r') = \quad & (min_{\sigma \in \textit{match-set}(r) \cup \textit{match-set}(r')} |\sigma|, \\ & max_{\sigma \in \textit{match-set}(r) \cup \textit{match-set}(r')} |\sigma|) \end{aligned}$$

And for convenience define the co-span of two sub-sequences of r between the i^{th} and j^{th} element and r' between the $i^{'th}$ and $j^{'th}$ element as a function *co-span*(r, i, j, r', i', j').

It should be noted that for the most part span is very easy to calculate as for `words` it is given by the two parameters and for literals it is $(1, 1)$, for entities it is defined as $(1, \infty)$, and the span of a rule or sub-sequence is simply the sum of all elements involved. The co-span is useful as it states the maximum and minimum number of terms that can be covered by both rule or rule sub-sequences

3.2.2.4: Definition Assume A is an alignment between two rules r,r'

$$A = \{(i_1, j_1), \ldots (i_n, j_n)\} \text{ where } i_1 < \ldots < i_n$$

Define the *alignment-to-join conversion* of A as a rule r^A is a rule of length $2n + 1$ with

$$r^A_m = \begin{cases} \texttt{words}(\textit{co-span}(r, 0, i_1, r', 0, j_1)) & m = 0 \\ \texttt{words}(\textit{co-span}(r, i_n, |r|, r', j_n, |r|)) & m = 2n + 1 \\ \texttt{words}(\textit{co-span}(r, i_{(m-1)/2}, i_{(m+1)/2}, r', j_{(m-1)/2}, j_{(m+1)/2})) & m \text{ is odd} \\ \texttt{r}_{\texttt{i}_{m/2}} & m \text{ is even} \end{cases}$$

This is actually quite easy to perform, as r^A is in fact the rule with all aligned elements preserved and in between each element a `words` element is introduced that matches the same possible spans as the combination of the two other rules for example if we have rules

```
:- "a" name() "b" "c" "d" name()
:- words(,1) name() words(2,3) "c" name()
```

And we have the following alignment on these rules

$$\{(2, 2), (4, 4), (6, 5)\}$$

Then we can get the alignment-to-join conversion as

```
:- words(,1) name() words(2,3) "c" words(0,1) name() words(0,0)
```

Which in simplified form is

```
:- name() words(2,3) "c" words(0,1) name()
```

3.2.2.5: Theorem The alignment-to-join conversion of two rules r,r' by alignment A, r^A is a join of r,r' or there exists a join of r and r', r'' such that $r'' \leq r^A$.
Proof: It is simple to observe that, as every introduced `words` has its match-set given by the co-span of unaligned elements, and all aligned elements are in r^A, we have that the match-set of r^A, contains both the match set of r and r' hence, r^A is either a join of r and r'' or subsumes some join of r and r''.□

3.2.2.6: Theorem If there are two rules r and r' and r^J is a join of r and r', then there is an alignment of r and r', A, such that $r^A = r^J$.
Proof: It should be clear that every element in r^J that is either a literal or an entity must have corresponding elements in r, r'. This can be observed by observing that if we have a set M_σ given by the set of sentences consisting of the literals and entities of r^J in order with any number of σ inserted at any point, where σ is a symbol not mentioned by any literal in r^J or r or r', then the matches in $M_\sigma \cap match\text{-}set(r^J)$ can be consistently mapped to the matchers of r^J and r, r'. This consistent mapping can then simply give an indexed alignment.□

3.2.2.7: Corollary If $\mathcal{A}^{(r,r')}$, are all possible alignments of r and r' then it follows that

$$\{r^A | A \in \mathcal{A}^{(r,r')}\} \supseteq join\text{-}set(r, r')$$

It is easy to find $join\text{-}set(r, r')$ by taking $\{r^A | A \in \mathcal{A}^{(r,r')}\}$ and removing all rules that subsume other rules. This gives the main result for generating generalised rules, in that it is only necessary to calculate all valid alignments between two rules, in order to calculate the join-set. As this the rules which minimally match both the same sentences that both rules do.

3.2.2.8: Algorithm
Input: Two rules r,r'.
Output: $\mathcal{A}(r, r')$, the set of all alignments of r and r'.

1. First find all entities in r and match them sequentially to the entities r', to get *entity-base*, if this cannot be done or does not cover all the entities of r', **return:** $\emptyset$.

2. For each literal in r, find all equal literals in r', call this $literal - base$.

3. **return:** find-aligns(*entity-base*,*literal-base*), where

$$find\text{-}aligns(\{a_1, \ldots, a_n\}, B) = \begin{cases} \{a_1, \ldots, a_n\} & \text{if} B = \emptyset \\ find - aligns(\{a_1, \ldots, a_n, & \\ valid(a_1, b), \ldots valid(a_n, b)\}, & \\ B - \{b\}) & \text{for some } b \in B \end{cases}$$

and

$$valid(\{a_1, \ldots a_m\}, b) = \begin{cases} \{a_1, \ldots a_m, b\} & \text{if } \nexists a_i \text{ such that } a_i = (i, j), \\ & b = (i', j') \text{ and } i > i', j < j' \\ & \text{or } i < i', j > j' \\ null & \text{otherwise} \end{cases}$$

The advantage to this is that if there is a corpus, which as before is viewed as a sequence of symbols s, then it is possible to talk about the matches to this corpus of a rule $match\text{-}set_s(r)$. Furthermore if some of these are "correct" extractions, a set of correct possible matches in the corpus, $correct\text{-}matches(s)$, can be defined (this is a subset of all possible sub-sequences in s). I can now state a key result

3.2.2.9: Definition Define the *base rules* of s as a set of rules defined by for each $s' \in correct\text{-}matches(s)$ a rule $r_{s'}$, which matches s' and is minimal, that is there exists no r', which matches s' and has $r' < r_{s'}$ (note this has to exist as the rule is the shortest one made of literals and entities). Also define the *completed join-set* of a set of rules $R = \{r_1, \ldots r_n\}$ recursively by

- $r_i \in completed\text{-}join\text{-}set(R)$.
- $join\text{-}set(r, r') \subseteq completed\text{-}join\text{-}set(R)$, if $r, r' \in completed\text{-}join\text{-}set(R)$

3.2.2.10: Theorem Assume there is a set of base rules, $R_s = \{r_{s_1}, \ldots r_{s_n}\}$ as in the previous definition. Suppose there is a rule $r \in completed\text{-}join\text{-}set(R_s)$ and another rule $r' \in completed\text{-}join\text{-}set(R_s)$, with $r < r'$ and there is no $r^2 \in completed\text{-}join\text{-}set(R_s)$ such that $r < r^2 < r'$. Then there is no rule r'' such that

$$match\text{-}set(r) \cap correct\text{-}matches(s) \subset match\text{-}set(r'') \cap correct\text{-}matches(s) \subset match\text{-}set(r') \cap correct\text{-}matches(s)$$

Proof: Assume this is not true, then there exists some s', which matches r' and r'' but not r, as it is a correct match there is also a base rule $r_{s'}$. This means we must have that r'' is in the join-set of r and $r_{s'}$ hence $r'' \in completed\text{-}join\text{-}set(R_s)$.□

This theorem states the idea that rules, which are not joins of some of the base rules, correspond to all valid ways to match that capture new examples from the training data. This is useful as it cuts the amount of work necessary to search the space of possible rules which may turn out to be very large. As such I can now present my algorithm as follows

3.2.2.11: Algorithm

Input: A corpus s, with training examples *correct-matches*(s).
Output: A set of rules R.

1. Initially let the rule queue, Q, be $Q = \textit{base-rules}(s)$, $R = \emptyset$.
2. While $Q \neq \emptyset$
   (a) Take some $r \in Q$, $Q \leftarrow Q - \{r\}$.
   (b) For each $r' \in Q$ let $Q \leftarrow Q \cup \textit{join-set}(r, r')$.
   (c) $R = R \cup r$

This algorithm is more effective than algorithm 3.1.8 as it is limited to only those rules, which differ significantly, as far as can be known from the corpus and seed term pairs. It is clear that it is possible to improve this algorithm by including a heuristic cost function as described in section 3.1.1, and in step 2a of the algorithm choosing r as the rule that has the best heuristic value. However often the join-set method can cut the search space to the point that all possible rules can be extracted.

3.2.3 Join-sets for more complex languages

An advantage of the join-set is that is easily adaptable to much more complex rule languages, for example SRL has several other matching elements

- **Word lists:** This element matches a predefined set of terms, for example a list of country names.
- **Orthographic:** This element matches a term based on its orthographic properties, for example, all capitalised, initial capitals, numeric terms, writing system (Hiragana, Katakana, etc.)
- **Begins,Ends,Contains:** Matches a single word, that begins in, ends in or contains a given sub-string.
- **Optional:** This matches a given term if it appears, or a zero-width match if the term is not present.

These can be automatically created by the use of join-sets in a similar manner to the previous method, however if there are two terms that don't match exactly, they don't necessarily create a wild-card (`words`) element. Using the matchers defined above it is possible to define the concept of *matchable* terms as

- If two terms are in the same word-list they are considered *matchable*
- If two terms have the same orthographic properties, based on a pre-defined list of relevant orthographic properties they are *matchable*.

- If two terms contain the same sub-string, they are *matchable*. In practise it is sensible to assume that this sub-string has a minimum length

Then it is possible to modify algorithm 3.2.2.8 so that it creates alignments based not on what literals are equal, but on which literals are matchable. Then the alignment-to-join conversion function of theorem 3.2.2.5, is also modified so that it checks if the literals are matchable and if they are, it inserts an appropriate matching element that generalises both of these. In fact, it is also theoretically possible that non-literal elements may also be matchable, for example the list of countries could be generalised into the orthographic matcher for initial capitalisation, as in English all country names are proper nouns and start with a capital letter. The key here is that as long as it is possible to define the minimal join of any one element, then it is possible to generalise this into the minimal join of any rule, through the use of an alignment.

Summary

Handling the problem of identifying multiple word terms, requires that the text be parsed and the terms identified through the use of a chunking methodology. This leads to more complex "rules", which I take from the SRL language. With these rules I approach a more complex methodology through the use of a *join-set* defined as the minimal set of rules which subsume a pair of rules. I show two key results about join-sets, firstly that they are readily calculated by finding alignments of the two rules. Secondly after defining the base rules as those which have no wild-cards, I show that the semi-lattice given by the join-sets of all base rules, represent all rules which could have some difference in precision over the corpus. This means that any pattern not in a join-set of some base patterns is not useful for my method.

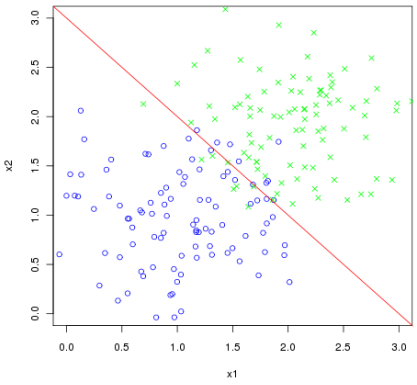

Figure 3.4: Example of a classification problem. The points represent training variable in two classes marked with green circles and blue crosses. The red line marks the decision boundary between the two classes

3.3 Classification

Once a rule set has been generated, some way to deduce whether a particular pair of terms are synonymous is required. My results and those of other authors

such as Snow et al. [2004] show that in general most rules are not very accurate at finding term pairs. To get around this I use a process known as classification. I shall first briefly recap the definition and several important algorithms for classification.

3.3.1 A brief introduction to classification

Assume there is some vector space $\mathcal{X}$, and two variable Y_1 and Y_2, which give probabilities of a vector $\mathbf{x}$, as $P(\mathbf{x}|Y_1)$ and $P(\mathbf{x}|Y_2)$ and there are also prior probabilities of the events $P(Y_1)$ and $P(Y_2)$. The goal is from a chosen data vector $\mathbf{x}$ to determine if it is more likely that this vector comes from Y_1 or Y_2 that is whether $P(Y_1|\mathbf{x}) > P(Y_2|\mathbf{x})$. By Bayes rule this can be reformulated as

$$P(\mathbf{x}|Y_1)P(Y_1) > P(\mathbf{x}|Y_2)P(Y_2)$$

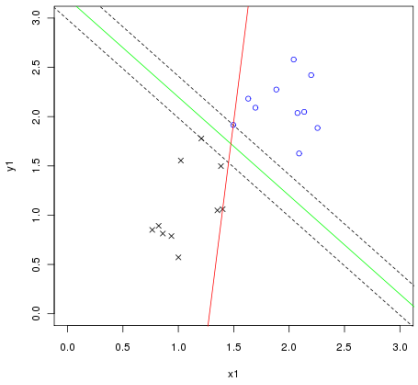

Figure 3.5: An example of the application of margin based classification, the green lines marks the decision boundary and its margin. The red line is another decision boundary with a small margin

A simple example is that Y_1 is the event that a person is male and Y_2 the event that that person is female. Then if $\mathcal{X}$ has vectors corresponding to facts such as height, weight, etc. it is possible to use this to guess if the person is male or female. This problem is easy if the probability functions are known, but instead I shall assume they are not and instead there are a number of examples $\{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$ which are called *training data*, for example $\mathbf{x}_i$ may be a sample of some random peoples height, weight etc. and y_i is whether they are male or female.

As this probability function is very difficult to estimate, it is common to make some form of independence assumption, for example in the *Naive Bayes* classifier, the assumption is that

$$P(\mathbf{x}|Y_1) = \prod P(x_i|Y_1)$$

This is then requires finding a probability distribution for each $P(x_i|Y_1)$. This can be done by assuming that $x_i|Y_1 \sim N(\mu_1, \sigma_1^2)$ and $x_i|Y_2 \sim N(\mu_2, \sigma_2^2)$. Then the prior probabilities of $P(Y_1)$ can easily be estimated by the number of examples in the training data that have the class value y_1, and standard methods can be used to estimate μ_1, μ_2, σ_1 and σ_2. This method is effective for many forms of data when the assumption that the data is distributed normally is approximately correct, however it cannot deal with more complex distributions.

Another approach is instead of attempting to calculate the probability distributions relying on finding the line where

$$P(\mathbf{x}|Y_1)P(Y_1) = P(\mathbf{x}|Y_2)P(Y_2)$$

There are a number of methods that approach the problem in this way, one that is of particular interest is called *margin classification*, these methods attempt to find a straight line that splits the data set so that each data point is as close to the line and on the right side of the line as frequently as possible.

This can be thought of in terms of a *margin*, that is the amount of separation between the two classes, for an example see figure 3.5, which shows two sets of data, and two potential decision boundaries, both of which divide the points correctly. However the red line has no margin, that is there is no gap between the classes, where as the green line has a large margin, denoted by the two dashed lines. One of the most well known forms of margin classification is known as *support vector machines* (see Cortes and Vapnik [1995]), support vector machines allow the margin to be defined by a set of support vectors, within a margin of the decision boundaries. Furthermore, the computational form of a SVM allows for it easily be adapted to find non-linear boundaries, by using the fact that the distances between vectors is calculated by their *dot product*. The methods of an SVM allow this dot product to be replaced with a function called a kernel, which is a *Gaussian radial basis function*, and performs the same function in calculating distances. But as the distances are now non-linear, a non-linear boundary is obtained, for example a commonly used kernel is a simple polynomial

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^d$$

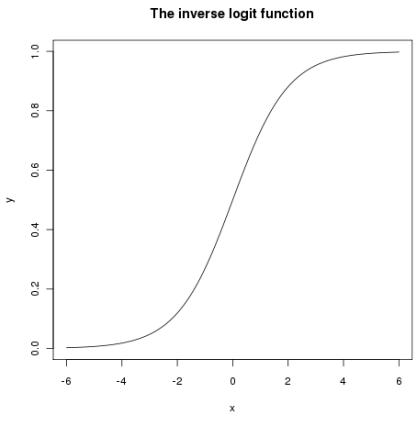

Figure 3.6: The inverse logit function

While this is effective, often not just a decision boundary but instead an actual estimate of the probability is required. For this reason a method called *Logistic Regression* is used, which is based on using the following assumption

$$P(Y_1|\mathbf{x}) = \frac{1}{1 + e^{-z}}$$

Where

$$z = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n$$

This is useful as it represents the kind of data found from a multivariate binomial distribution because it is the distribution's link function as a generalised linear model (for more discussion of this see Hastie et al. [2001]). This

method is very useful as it gives a much more accurate prediction of the probability that a data point belongs to a certain class. This method can be incorporated with support vector machines, to give a system that can both accurately predict probabilities and produces strong predictive results.

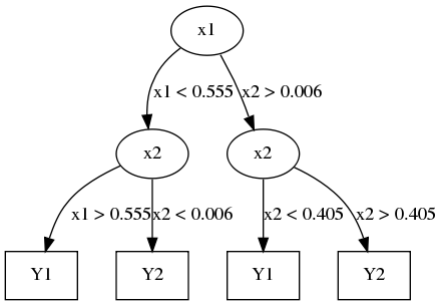

Figure 3.7: An example of a decision tree

Another classifier that is commonly used are those based on decision trees. This attempts to form a tree that can be used to decide which class the data belongs, for example see the tree in figure 3.7, this tree first checks to see if variable x_1 is greater than or less than 0.555, then it branches to the next stage that checks x_2. There are various methods for constructing such trees, for example see the C4.5 method of Quinlan [1993]. For a more complete introduction of decision trees and other topics in classification see Hastie et al. [2001].

3.3.2 Finding term pairs by classification

Now that the set of rules has been found, it is possible to find all places where they match in the corpus and extract a set of term pairs from the corpus. For each of these term pairs it is possible to extract the number of times it matches to each rule, and use this as the training data.

3.4.2.1: Definition Let the set of extracted rules be $P = \{p_1, \ldots p_n\}$ and the term pairs $T = \{(t_1, t'_1), \ldots (t_m, t'_m)\}$, for each term pair create a vector $x_i = (x_{i1} \ldots x_{in})^T$, such that x_{ij} is the number of matches of rule p_j to term pair (t_i, t'_i).

Now that vectors have been obtained, it is necessary to train the classifier. For this it is required that there is some classifications as to which of these rules are positive or negative. This could be done by using an oracle such as WordNet, defining $y_i = 1$ if this term pair is known to exhibit a relationship, $y_i = 0$ if both terms are in the oracle but no relationship is marked, and all vectors from the training set, whose term pairs are not in the oracle are removed. Alternatively given the seed pairs used to extract the rules, if it is assumed that no two terms from different seed pairs share a relationship it is possible to use this as an oracle. Once these have been extracted, they can be used to train a classifier such as the ones I talked about above. Then use this on any vectors where the class value is not known on to produce a probability that that pair exhibits the relationship. Further more it is also necessary to classify the *zero*

vector, a vector that has all zeros, as the probability of this is given as a base line for term pairs for which no rules match in the corpus. So:

- For term pairs, (t_1, t_2) in the oracle $P(r(t_1, t_2)) = 1$.
- For term pairs, (t_1, t_2) not in the oracle but with both t_1 and t_2 in some other term pair in the oracle $P(r(t_1, t_2)) = 0$.
- For term pairs, (t_1, t_2) with one term not in the oracle and a vector $\mathbf{x}$, $P(r(t_1, t_2)) = \hat{P}(\mathbf{x})$, where $\hat{P}$ is the statistical classification system
- For all other term pairs, (t_1, t_2),$P(r(t_1, t_2)) = \hat{P}(\mathbf{0})$.

Summary

I briefly introduce the concept of *classification*, which is a statistical method for classifying vectors into a number of classes based on some previous training data. I then relate this to my problem by showing how to generate vectors from generated rules and corpus and explain how this can be combined with a classification system to produce a probability value for each term pair, which indicates how likely it is that a given term pair is related by a specific relationship.

Chapter 4

Logical consistency of Ontologies

4.1 Defining the Problem

4.1.1 Networks

The problem concerns a set of elements, which are the terms extracted by the process in the previous section, define d as a finite set, E. There are also a finite number of relations on this set, which is defined as a 2-ary predicate on the set E, I write $r(e_1, e_2)$ to denote the relationship r between to elements e_1, e_2. Then define a *network* as a set of such statements, and the set of possible relationships as the set R.

4.1.1: Definition $N(E, R)$ is the set of all possible networks on the set of elements E and relationships R. Define $\mathcal{N}$ as $\mathcal{N} = \{r_i(e_j, e_k) | r_i \in R, e_j \in E, e_k \in E\}$ and as such it is clear $N(E, R) = \mathcal{P}(\mathcal{N})$.

Furthermore, define the set of *valid networks*, $V \subseteq N(E, R)$ to be some subset of the possible networks. Also define a cost function as follows

4.1.2: Definition $c : N(E, R) \to \mathbb{R}$ is a *cost function* if it satisfies the following axioms

1. $c(N) \geq 0$ for all $N \in N(E, R)$
2. $c(P) = 0$ for some $P \in N(E, R)$ and $c(N) = 0$ iff $N = P$ for all $N(E, R)$
3. If $N \ominus P \subseteq M \ominus P$ then $c(N) \leq c(M)$ for all $N, M \in N(E, R)$ [1]

Call this value P the minimal of the cost function

[1] I define $S \ominus T \equiv (S \cup T) - (S \cap T)$

The last axiom is intended to ensure that the metric is calculated only from the sections of the networks which actually differ. I now state the problem as such

4.1.3 Definition Define a *consistency satisfaction problem* as a tuple (E, R, V, c), which has that $V \subseteq N(E, R)$ and c is a cost function as in definition 4.1.2. I say that a *solution* to this problem is a network N such that $N \in V$ and there does not exist $N' \in V$ such that $c(N) > c(N')$

Note that under this definition there may not be a unique N, for the purpose of the system if N is not unique these solutions are "indistinguishable" and the system can do no better than choosing a single one at random, so it is only required that an optimal solution is found.

4.1.2 Cost functions

Assume that there is a probability distribution over $N(E, R)$ so that for each $N \in N(E, R)$ a value $p(N)$ can be calculated. Assume there is some value $P \in N(E, R)$ such that there does not exist $P' \in N(E, R)$ such that $p(P) \leq p(P')$, then I define the cost function as

$$c(N) = log(p(P)) - log(p(N))$$

It is clear that this function satisfies axiom 1 and 2 of the cost function definition and that this function increases as $p(N)$ decreases. However it may not always satisfy the third axiom of the definition 2.

4.1.2.1: Definition Define $p : N(E, R) \rightarrow \mathbb{R}$ as a function such that $0 \leq p(r(e_1, e_2)) \leq 1$ for all $r(e_1, e_2) \in N(E, R)$ I shall say that a network is independently generated, denoted $N \sim Ind(p)$ and has the following probability distribution function

$$p(N = N_1) = \prod_{r(e_1,e_2) \in N_1} p(r(e_1, e_2)) \prod_{r(e_1,e_2) \notin \mathcal{N} \ominus N_1} (1 - p(r(e_1, e_2)))$$

As a corollary it is clear that P is defined as

$$r(e_1, e_2) \in P \leftrightarrow p(r(e_1, e_2)) > 0.5$$

4.1.2.2: Lemma If $N \sim Ind(p)$ then $c(N) = log(p(P)) - log(p(N))$ is a cost function.
Proof: It is easy to show that $c(N)$ is positive and $c(P) = 0$.

$$\begin{array}{c} c(N) = log(\prod_{r(e_1,e_2) \in P} p(r(e_1, e_2)) \prod_{r(e_1,e_2) \notin \hat{N} \ominus P} (1 - p(r(e_1, e_2)))) \\ -log(\prod_{r(e_1,e_2) \in N_1} p(r(e_1, e_2)) \prod_{r(e_1,e_2) \notin \hat{N} \ominus N_1} (1 - p(r(e_1, e_2)))) \end{array}$$

Hence it follows

$$c(N) = \sum_{r(e_1,e_2) \in P \ominus N} (log(p(r(e_1, e_2))) - log(1 - p(r(e_1, e_2))))$$

So it is clear that the third axiom holds.□

Summary

I start by assuming that there is a set of elements and some set of relationships between them and that the pattern-based extraction systems of chapter 3 produce a single probability that any of these elements are related by a given relationship. I define a *network* as a set of these relations (in effect it is a multi-relation graph), and I state that a subset of all possible networks are considered valid. I define a cost function on a network and show that given a reasonable independence assumption, this leads to a linear function based on the probability values from the pattern-based extraction system.

4.2 Equivalence sets

For the first case I shall define this problem only in terms of equivalence sets, this is useful when dealing with a single relationship that is an equality relation such as *noun synonymy*.

4.2.1: Definition An *exact cover* is defined a set of subsets of a set E such that, if $C = \{C_1, \ldots C_n\}$ then

1. For all $e \in E$, $\exists i$ such that $1 \leq i \leq n$ and $e \in C_i$
2. For all i, j such that for $1 \leq i < j \leq n$, $C_i \cap C_j = \emptyset$

Define an *equivalence network*, N, derived from an exact cover, C, as a network over a single relationship r and such that $r(e_1, e_2) \in N$ if and only if $\exists i$ such that $1 \leq i \leq n$ and $e_1 \in C_i$ and $e_2 \in C_i$.

More informally defined, an exact cover is a division of the elements such that each element is in exactly one set. In the equivalence problem I define the set V as the set of all equivalence networks, that is a network is only valid if it represents an exact cover. As such it is possible to define the equality system as such

4.2.2: Definition If $N \sim Ind(p)$ and V is the set of the equivalence networks, the problem can be redefined as: find an exact cover of $C = \{C_1, \ldots C_n\}$ such that C is maximal with respect to $\sum_{i=1\ldots n} c(C_i)$ where

$$c(C_i) = \sum_{e_1 \in C_i} \left(\sum_{e_2 \in C_i} log(p(r(e_1, e_2))) + \sum_{e_2 \in E - C_i} log(1 - p(r(e_1, e_2))) \right)$$

This means that if N is network given by C, it follows that

$$c(N) = log(p(P)) - \sum_{i_1,\ldots n} c(C_i)$$

Hence the goal of minimising $c(N)$ is the same as maximising $\sum_{i_1,\ldots n} c(C_i)$

For simplicity I shall assume that $E = \{e_1, \ldots e_n\}$ and then define the *inter-node cost*, c_{ij} as

$$c_{ij} = log(p(r(e_i, e_j))) - log(1 - p(r(e_i, e_j)))$$

4.2.3: Lemma If $C_0 = \{\{e_1\}, \ldots \{e_n\}\}$ then

$$c(\{C_1, \ldots C_m\}) = c(C_0) + \sum_{i=1,\ldots m} \sum_{\{j,k\} \subseteq C_i} c_{jk}$$

Proof: It is simple to observe that

$$c(C_0) = \sum_{e_1, e_2 \in E} log(1 - p(r(e_1, e_2)))$$

Hence by adding c_{jk} it is possible to get the same expression as in definition 4.2.2.□

From this point treat $c(C_0)$ as a constant and ignore for the purpose of my algorithms.

4.2.4: Definition Define a *linear equivalence problem* as a tuple $(E, r, \{c_{ij}\})$ such that the goal is to find an exact cover of E, $\{C_1, \ldots C_m\}$ which maximises the value of

$$\sum_{i=1,\ldots m} \sum_{\{j,k\} \subseteq C_i} c_{jk}$$

4.2.1 Branch and Bound algorithms

The simplest approach is to find the set cover by adding elements one by one to the set

4.2.1.1: Algorithm
Input: A linear equivalence problem $(E, r, \{c_{ij}\})$
Output: An exact cover of E, $\{C_1, \ldots C_m\}$

1. Initially $C = \{\}$, $E' = E$
2. While C is not an exact cover
   (a) Take $e_i \in E'$, assume $C = \{C_1, \ldots C_n\}$. Let k be the maximal argument value of $\sum_{e_j \in C_k} c_{ij}$ and c be this value.

(b) Let $E' \leftarrow E' - \{e_i\}$

(c) If $c < 0$ then let $C \leftarrow C \cup \{C_{n+1}\}$ where $C_{n+1} = \{e_1\}$ otherwise let $C \leftarrow \{C_1, \ldots C_k \cup \{e_1\}, \ldots C_n\}$

This algorithm works by starting with an empty cover and taking each element one by one and adding it to an existing set or a new set by itself according to which one would cause the least increase in the cost.

4.2.1.2: Example Take $E = \{1, 2, 3\}$ and the following values for c_{ij}

$$c_{ij} = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & -1 & 2 \\ 2 & -1 & 0 & 3 \\ 3 & 2 & 3 & 0 \end{array}$$

- **Iteration 1:** Take $e = 1$. We cannot define c so we get $C = \{\{1\}\}$ and $E' = \{2, 3\}$
- **Iteration 2:** Take $e = 2$. We get $c = -1$ and hence so we get $C = \{\{1\}, \{2\}\}$ and $E' = \{3\}$
- **Iteration 3:** Take $e = 3$. We get $i = 2$ and $c = 3$ and $C = \{\{1\}, \{2, 3\}\}$ with a total overall cost of 6.

However the solution of $C = \{\{1, 2, 3\}\}$ has a higher cost of 8 so this is not the optimal. This demonstrates that algorithm 4.2.1.1 cannot be used to find the true optimal solution, but this can be performed by a branch and bound algorithm. This is done by branching whenever the algorithm would choose to add an element to a set in the cover. First I define a simple bounding condition

4.2.1.3: Definition Let

$$C^{>0} = \{c_{ij} | c_{ij} > 0\}$$

and then let

$$b(E') = \sum_{e_i \in E'} \left(\sum_{e_j \in E \wedge c_{ij} \in C^{>0}} c_{ij} + \sum_{e_j \in E \wedge c_{ji} \in C^{>0}} c_{ji} \right)$$

This means that $b(E')$ is the maximum possible increase to the overall solution by adding elements from E', which in algorithm 4.2.1.1 is the elements not yet included in the exact cover.

4.2.1.4: Lemma If C is an exact cover or $E - E'$ there does not exist and exact cover of E, C' such that $c(C') < c(C) + b(E')$ and C is a *sub-solution* of C'

(that is if $C = \{C_1, \ldots C_m\}$ and $C' = \{C'_1, \ldots C'_p\}$ then for all i with $1 \leq i \leq p$ there exists j such that $C_j \subseteq C'_i$ or $C'_i \subseteq E - E'$).

I can now present an algorithm capable of finding the exact solution to the equivalence set problem

4.2.1.5: Algorithm
Input: A linear equivalence problem $(E, r, \{c_{ij}\})$
Output: An exact cover of E, $\{C_1, \ldots C_m\}$

1. Set $c^* = -\infty$, $C^* = undef$
2. Call function $solve(\{\}, 0, E)$
3. Output C^*

function **solve**(C, c, E')

1. If $E' = \emptyset$ then if $c > c^*$ then $c^* \leftarrow c$ and $C^* \leftarrow C$. **return**
2. If $c + b(E') < c^*$ **return**
3. Take $e_i \in E'$
4. Assume $C = \{C_1, \ldots C_n\}$ for $k = 1, \ldots n$ call $solve(\{C_1, \ldots C_k \cup \{e\}, \ldots C_n\}, c + \sum_{e_j \in C_k}(c_{ij} + c_{ji})), E' - \{e_i\})$
5. Call $solve(C \cup \{\{e_i\}\}, c, E' - \{e_i\})$

This algorithm now recursively branched until it finds the optimal solution, however this still potentially covers a very large search space and I would like to improve this further. I will do this by attempting to improve on the bounding condition.

4.2.1.7: Definition Define the *set join cost* of two sets, denoted $c_{join}(C_1, C_2) : \mathcal{P}(E) \times \mathcal{P}(E) \rightarrow \mathbb{R}$ as

$$c_{join}(C_1, C_2) = \sum_{e_i \in C_1} \sum_{e_j \in C_2} (c_{ij} + c_{ji})$$

4.2.1.8: Lemma If for some $C \subseteq E$ then $C = C_1 \cup C_2$ and $C_1 \cap C_2 = \emptyset$ then

$$c(C) = c(C_1) + c(C_2) + c_{join}(C_1, C_2)$$

Proof: Follows trivially from definition of $c(C)$ and c_{ij}.

This join cost can be used to provide a much more powerful bounding criterion

4.2.1.9: Theorem For any $E' \subset E$ and $e_k \in E$, $e_k \notin E'$ and $C \subset E$, with $C \cap E' = \emptyset$ and $e_k \notin C$, if it holds that

$$c_{join}(C, \{e_k\}) + max_{C' \subseteq E'} c_{join}(C', e_k) < 0$$

Then it follows that there does not exists a set K such that $C \cup \{e_k\} \subset K \subset C \cup \{e_k\} \cup \{E'\}$ and

$$c(K) > c(K - \{e_k\}) + c(\{e_k\})$$

Proof: It is clear that for any set K

$$max_{C' \subseteq E'} c_{join}(C', e_k) \geq c_{join}(K - \{e_k\}, \{e_k\}) - c_{join}(C, \{e_k\})$$

and so

$$0 > c_{join}(K - \{e_k\}, \{e_k\})$$

Hence

$$c(K - \{e_k\}) + c_{join}(K - \{e_k\}, \{e_k\}) + c(\{e_k\}) < c(K - \{e_k\}) + c(\{e_k\})$$

By above lemma we get our result

$$c(K) > c(K - \{e_k\}) + c(\{e_k\}) \square$$

Simplified this theorem states that if the criterion

$$c_{join}(C, \{e_k\}) + max_{C' \subseteq E'} c_{join}(C', e_k) < 0$$

Holds, then any set K, which could be added to the solution and contains e_k will produce a worse overall exact cover than the cover where $K - \{e_k\}$ and $\{e_k\}$ are separate. Fortunately checking this criterion is easy as $max_{C' \subseteq E'} c_{join}(C', e_k)$ is actually easy to compute

4.2.1.10: Lemma If $C' = argmax_{C' \subseteq E'} c_{join}(C', e_k)$ then

$$C' = \{e_i | c_{ik} + c_{ki} > 0\}$$

Proof: This is actually trivial from the definition of c_{join} as it is the sum of values of the form c_{ik} and c_{ki}.$\square$

This means that there is a new bounding condition for my algorithm, when a set C has been generated and it is attempting to add an element e_k to C it is possible to first check if the condition in the above theorem holds. If it does then any solution which has $\{C \cup \{e_k\}\}$ as a sub-solution is less optimal than one which has $\{C, \{e_k\}\}$ as a sub-solution.

I can also offer one more modest improvement

4.2.1.11: Theorem If $E = E_1 \cup E_2$ and $E_1 \cap E_2 = \emptyset$ and the sets are such that

$$\forall e_i \in E_1 \forall e_j \ \in E_2 c_{ij} + c_{ji} < 0$$

Then for any set $C \subseteq E$ such that $E_1 \cap C \neq \emptyset$ and $E_2 \cap C \neq \emptyset$. If C_1, C_2 be $C_1 = C \cap E_1$ and $C_2 = C \cap E_2$ then

$$c(C) > c(C_1) + c(C_2)$$

Proof: Simply observe $c(C) = c(C_1)+c(C_2)+c_{join}(C_1, C_2)$ and that $c_{join}(C_1, C_2) < 0$.□

This theorem essentially states that if the set of elements can be divided such that all links between each set are negative, it is possible to consider the problem as a number of sub-problems.

4.2.1.12: Definition Define the *components* of a set E as the set $C^p = \{C_1^p, \dots C_m^P\}$ such that

1. $\bigcup_{i=1,\dots m} C_i^P = E$
2. For all i, j such that $1 \leq i < j \leq m$, $C_i^p \cap C_j^p = \emptyset$
3. For all i, j such that $1 \leq i < j \leq m$ $\forall e_{i'} \in C_i^p \forall e_{j'} \in C_j^P c_{i'j'} + c_{j'i'} < 0$
4. For all i such that $1 \leq i \leq m$, $\forall e_{i'} \in C_i^p \exists e_{j'} \in C_i^P c_{i'j'} + c_{j'i'} \geq 0$

4.2.1.13: Lemma For any set the components exist, are unique and can be found by an algorithm in $O(|E|^2)$ time

4.2.1.14: Algorithm
Input: A set E with corresponding costs $\{c_{ij}\}$
Output: The components of E, $\{C_1^p, \dots C_m^p\}$

1. Initially let $C^p \rightarrow \{\{e_1\}, \dots \{e_n\}\}$
2. For each i from $1, \dots n$
   (a) For each j from $i + 1, \dots n$
      i. If $c_{ij} + c_{ji} \geq 0$, let $C_{i'}^p$ and $C_{j'}^p$ be the sets that contain e_i, e_j respectively, let $C^P \leftarrow (C^P - C_{i'}^P - C_{j'}^P) \cup \{C_{i'}^P \cup C_{j'}^P\}$

This algorithm proves the above lemma and although faster variants of this algorithm exist but have the same worst case performance, for the most part this method is not exceptionally slow so I feel no need to present more in depth versions of the algorithm. It is now possible to say

4.2.1.15: Corollary If $C^P = \{C_1^P, \ldots C_m^P\}$ are the components of E and the optimal exact cover of each C_i^P is S_i for $i = 1, \ldots m$ then the optimal exact cover of E is $\bigcup_{i=1,\ldots m} S_i$

I now present an improved version of the branch and bound algorithm

4.2.1.16: Algorithm
Input: A linear equivalence problem $(E, r, \{c_{ij}\})$
Output: An exact cover of E, $\{C_1, \ldots C_m\}$

1. Find the components C^p of E by **Algorithm 4.2.1.14**
2. Set $C^* \rightarrow \{\}$
3. For each $C_i^p \in C^p$
   (a) Set $c_i^* = -\infty$, $C_i^* = undef$
   (b) Call function $solve(\{\}, 0, C_i^p)$
4. Output C^*

function **solve**(C, c, E')

1. If $E' = \emptyset$ then if $c > c^*$ then $c^* \leftarrow c$ and $C^* \leftarrow C$. **return**
2. If $c + b(E') < c^*$ **return**
3. Take $e_i \in E'$
4. Assume $C = \{C_1, \ldots C_n\}$
5. For each k from $1, \ldots n$
   (a) If $c_{join}(C_k, \{e_i\}) + max_{C' \subseteq E'} c_{join}(C', e_i) \geq 0$ then call $solve(\{C_1, \ldots C_k \cup \{e\}, \ldots C_n\}, c + \sum_{e_j \in C_k}(c_{ij} + c_{ji})), E' - \{e_i\})$
6. Call $solve(C \cup \{\{e_i\}\}, c, E' - \{e_i\})$

4.2.2 Implementation by Dancing Links

One of the main features of the algorithm is that it is in fact possible to implement it quite efficiently by implementing the matrix correctly. I do this by using a very specific form of a sparse matrix, known as a *dancing links* matrix, following Knuth [2004]

This form consists of a node with two values `i` and `j`, containing the nodes location and four pointers `up`, `down`,`left`,`right`, each of which point to the node that is immediately in that direction. If no such node exists then this pointer instead points to the first value on the opposite side of the matrix (i.e., `down` would point to the top value in the same column). this value. By which I mean if there is no node immediately above the chosen node the `up` pointer points to the node at the bottom of the same column, which could even be the chosen node.

This data structure is useful as it allows columns to easily be removed and then the column remove undone, so to remove a column it is only necessary to apply the following code to each `node`.

```
node->left->right = node->right
node->right->left = node->left
```

This has the advantage that `node` still remembers where it was in the data structure as its `left` and `right` parameters. This form also makes it possible to quickly find the reduction by a column because it is simply possible to iterate along the `left` and `up` pointers. For example see the representation of figure 4.1, which shows the matrix

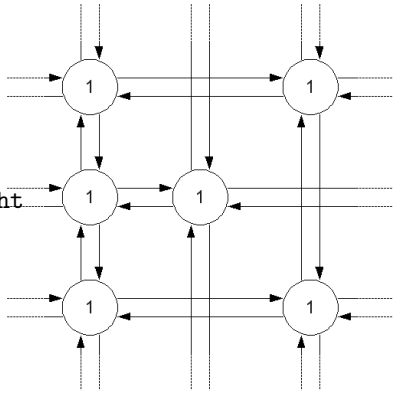

Figure 4.1: Links representation of a matrix

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

In this figure the arrows disappearing at the edge are assumed to continue on the opposing edge of the diagram. Figure 4.2 shows the effect on the matrix if a row is removed, where the second row has been removed. It is clear that the nodes in the second row are no longer accessible from any other nodes in the matrix, but that the first node in the second still has pointer to nodes in the first and third row. This effectively gives the matrix a "memory" of which columns were removed, making it possible to easily define operations to restore the matrix if needed.

4.2.3 Improving the algorithm with linear relaxation

A commonly used procedure for exact cover problems is to use linear programming to find a solution. This is done by use of relaxation, a relaxed problem is defined as a problem which is easily solved (typically in polynomial time) and under certain conditions will produce a solution to a NP-hard problem. This method is popular and has been used for a wide variety of problems of NP-Hard problems (see Hromkovič [2002]).

4.2.2.1: Definition *Linear programming* is the process of finding a vector $\mathbf{x} \in \mathbb{R}^N$ such that $\mathbf{x}$ satisfies the inequality $A\mathbf{x} \geq \mathbf{b}$ for some matrix A and constant vector $\mathbf{b} \in \mathbb{R}^N$ and there does not exist a vector $\mathbf{x}' \in \mathbb{R}^N$ such that for some vector $\mathbf{c} \in \mathbb{R}^N$ $\mathbf{c}^T\mathbf{x}' < \mathbf{c}^T\mathbf{x}$

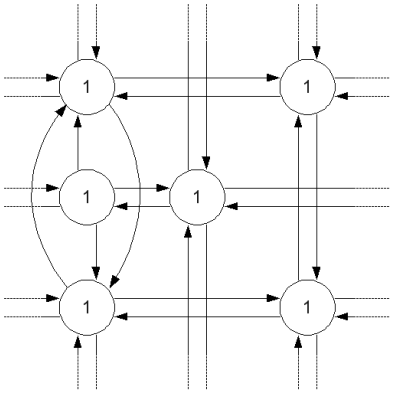

Figure 4.2: Links representation of a matrix, with one row removed

It is also easy to modify this definition without any loss of generality to include the case where there is an equality statement $A\mathbf{x} = \mathbf{b}$, instead of the inequality.

4.2.2.2: Theorem A linear programming problem can be solved in polynomial time

This theorem is proved in Khachian [1979]. I now turn to the issue of converting the problem to a linear programming problem.

4.2.2.3: Definition Let $\{C_1, \dots C_m\} = \mathcal{P}(E)$, let $A = (a_{ij})$ be a matrix of size $|\mathcal{P}(E)| \times |E|$ where $a_{ij} = 1$ if $e_j \in C_i$ and $a_{ij} = 0$ otherwise.

4.2.2.4: Lemma If $\mathbf{x} \in \{0,1\}^{|\mathcal{P}(E)|}$ is a vector such that $A\mathbf{x} = \mathbf{1}$[1] , then the set C^*, given by $C_i \in C^*$ if and only if $x_i = 1$, is an exact cover of E.
Proof: If $C_i \in C^*$ and $C_j \in C^*$ and $i \neq j$ then $C_i \cap C_j = \emptyset$ as if $e_k \in C_i$ and $e_k \in C_j$ then $a_{ik} = 1$ and $a_{jk} = 1$ hence it follows that if $\mathbf{y} = A\mathbf{x}$, $y_k \geq 2$. Also for all k, $y_k = 1$ hence it follows that there must be some i such that $a_{ik} = 1$ and $x_k = 1$, hence it follows $C_i \in C^*$ and $e_k \in C_i$.□

4.2.2.6: Definition Let $\mathbf{c} \in \mathbb{R}^{|\mathcal{P}(E)|}$ be a vector that corresponds to the values $\{c_{ij}\}$

It is now possible to restate the problem of finding the maximal exact cover to finding a vector $\mathbf{x} \in \{0,1\}^{|\mathcal{P}(E)|}$ which maximises

$$\mathbf{c}^T\mathbf{x}$$

subject to

$$A\mathbf{x} = \mathbf{1}$$

It is clear there is a relaxed version of this problem, which requires only finding a vector $\mathbf{x} \in \mathbb{R}^{|\mathcal{P}(E)|}$, which satisfies the above programming problem and then use this value to find $\mathbf{x} \in \{0,1\}^{|\mathcal{P}(E)|}$. However before attempting to do this there is a problem in that $|\mathcal{P}(E)| = 2^{|E|}$, which means that any trivial attempt to form the matrix A will be computationally intractable as the size of this

[1]Define $\mathbf{1}$ as the vector $\mathbf{1} = (1 \dots 1)^T$, and similarly the vector $\mathbf{0}$

matrix will be too large. As such I wish to find a set D such that $D \subset \mathcal{P}(E)$ and if C^* is the true optimal solution and $C^* \subset D$. The basic approach to this is to disregard all sets which cannot possibly be in the optimal solution. I have already shown two principles for doing this in Theorem 4.2.1.9 and Lemma 4.2.1.11, and they can be reused here to generate the set D

4.2.2.7: Algorithm
Input: A set E with corresponding costs $\{c_{ij}\}$.
Output: A set $D \subseteq \mathcal{P}(E)$, for which some subset of D is the maximal exact cover of E.

1. Let $J \leftarrow \emptyset$, $D \leftarrow \emptyset$
2. For each connected component V in E: **call** $generate\_matrix(J, V)$
3. **output** D

function $generate\_matrix(J, V)$

1. while $V \neq \emptyset$
   (a) Let $e_k \in V$ be such that it minimises $\sum_{i=1...|E|} c_{ik}$
   (b) Let $V \leftarrow V - \{e_k\}$
   (c) If $J = \emptyset$ or $\sum_{e_i \in J} c_{ik} > 0$ or $-\sum_{e_i \in J} c_{ik} < max_{V' \subset V} \sum_{e_i \in V'} c_{ik}$ then
      i. Add $J \cup e_k$ to D.
      ii. **call** $generate\_matrix(J \cup e_k, V)$

This algorithm incorporates the use of connected components as well and outputs the generated set D. Then use D to generate a matrix A_D which is formed in the same way as A but included only the sets that were generated in D. I then find the vector $\mathbf{x^r} \in \mathbb{R}^{|\mathcal{P}(E)|}$ that minimises $\mathbf{c}^T\mathbf{x^r}$ subject to $A_D\mathbf{x^r} = \mathbf{1}$. The goal is now to use this solution to find the optimal solution and I approach this by considering those values of $\mathbf{x}$ which are non-integer and branches into two solutions one where zero is assumed and one where one is assumed. This can be done fairly efficiently by reducing A to a smaller matrix. For the case where I assume the non-integer value to be zero, it is only necessary to set the corresponding column to all zero as this set is no longer a possible solution. In the second case it is necessary to remove all other columns which have a 1 in the same row the corresponding column has a 1.

4.2.2.8: Definition Define the reduced matrix A'_i as a matrix where $a'_{jk} = 0$ if $\exists m$ such that $a_{jm} = 1$ and $a_{im} = 1$ and $j \neq i$, and $a'_{jk} = a_{jk}$ otherwise.

4.2.2.9: Algorithm

Input: A matrix A and a vector $\mathbf{x}^r \in \mathbb{R}^{|\mathcal{P}(E)|}$ which minimises $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} = 1$.
Output: A matrix A and a vector $\mathbf{x}^r \in \{0,1\}^{|\mathcal{P}(E)|}$ which minimises $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} = 1$.
function $unrelax(A, \mathbf{x^r})$

1. Let k be a value such that $0 < x^r_k < 1$. If no value exists **return** $\mathbf{c}^T\mathbf{x^r}$.
2. Let A^0 be the matrix A, except $a_{kj} = 0$ for $j = 1 \ldots |E|$.
3. If for all i there exists j such that $a^0_{ij} = 1$ then find $\mathbf{x} \in \mathbb{R}^{|E|}$ that minimises $\mathbf{c}^T\mathbf{x}$ subject to $A^0\mathbf{x} = \mathbf{1}$. Let $\mathbf{x^0} = unrelax(A^0, \mathbf{x})$.
4. Find $\mathbf{x} \in \mathbb{R}^{|E|}$ that minimises $\mathbf{c}^T\mathbf{x}$ subject to $A'_k\mathbf{x} = \mathbf{1}$. Let $\mathbf{x^1} = unrelax(A'_k, \mathbf{x})$
5. **return** $\mathbf{x^0}$ if $\mathbf{c}^T\mathbf{x^0} < \mathbf{c}^T\mathbf{x^1}$ or **return** $\mathbf{x^1}$ otherwise.

In fact it should be noted that the derelaxation algorithm can itself be used as a solution for the problem without requiring the linear programming as in stead of searching for a new vector $\mathbf{x}$ at every stage, instead continuing the branching until the minimum solution is found, however linear relaxation can greatly reduce the number of iteration the algorithm has to perform.

4.2.4 Simplex Algorithm

There are several methods that are capable of solving linear programming problems. For this work I chose to use the *Simplex algorithm* of Dantzig [1951]. This method has exponential complexity however it has been shown to often out perform polynomial time algorithms due to its simplicity. Firstly recall the definition of a linear programming problem is minimising $\mathbf{c}^T\mathbf{x}$ subject to

$$A\mathbf{x} \geq \mathbf{b}$$

Suppose

$$A = \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}$$

Now it is possible to restate the problem as a set of linear equation by the use of *slack variables*, $s_1 \ldots s_m, t$

$$\begin{gathered} a_{11}x_1 + \ldots + a_{1n}x_n + s_1 = b_1 \\ \vdots \\ a_{m1}x_1 + \ldots + a_{mn}x_n + s_m = b_m \\ -c_1x_1 + \ldots + -c_nx_n + t = 0 \end{gathered}$$

This then makes it possible to state a single matrix, which contains all the necessary information, which is called a *tableau*

$$\left(\begin{array}{ccccccc|c} a_{11} & \dots & a_{1n} & 1 & \dots & 0 & 0 & b_1 \\ \vdots & & \vdots & 0 & \ddots & 0 & 0 & \vdots \\ a_{m1} & \dots & a_{mn} & 0 & \dots & 1 & 0 & b_m \\ \hline -c_1 & \dots & -c_n & 0 & \dots & 0 & 1 & 0 \end{array}\right)$$

It follows that it is possible to apply simple linear manipulations until a suitable matrix is achieved. The next step is to rearrange this so that the linear solution gives a simple solution, this is performed by using linear transformations to make it such that some columns contain only one zero. For example assume there is a problem with

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mathbf{b} = (3\ 2)^T, \mathbf{c} = (1\ 2)^T$$

Then this gives a tableau

$$\left(\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 1 & 0 & 2 \\ \hline -1 & -2 & 0 & 0 & 1 & 0 \end{array}\right)$$

Now by adding the first row to the third row and the second row to the third row we can obtain the following tableau

$$\left(\begin{array}{ccccc|c} 1 & 0 & 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 1 & 0 & 2 \\ \hline 0 & 0 & 1 & 1 & 1 & 5 \end{array}\right)$$

Which corresponds to the following linear equations

$$\begin{gathered} x_1 + s_1 = 3 \\ x_2 + s_2 = 2 \\ s_1 + s_2 + t = 5 \end{gathered}$$

Now it is clear that by setting $s_1 = 0$ and $s_2 = 0$ we can get the solution $\mathbf{x} = (3\ 2)^T$. Call a column *cleared* if all of its values are zero except one. The simplex algorithm attempts to then clear the variable using the row that cause the smallest increase in the cost variable t. This is performed by finding a pivot column as the column with the lowest (negative) value, and then finding a pivot row j, which is the row that has the lowest value for

$$\frac{b_i}{a_{ij}}$$

As such I can now present the simplex algorithm

4.2.3.1: Algorithm
Input: A linear programming problem, $(A, \mathbf{b}, \mathbf{c})$
Output: A vector $\mathbf{x}$, which solves the problem

1. Form a tableau
2. Choose the pivot column, j, as the column with the lowest negative value in the bottom row
3. Choose the pivot row, i, as the row that minimises $\frac{b_i}{a_{ij}}$
4. Multiply the pivot row by $\frac{1}{a_{ij}}$.
5. For each other row add the pivot multiplied by some value such that the value in the pivot column is 0.
6. If the bottom row has a negative value go to step 2
7. Output $\mathbf{x}$, the solution given by setting the variables of all uncleaned columns to zero

4.2.3.2: Theorem The simplex algorithm solves a given linear programming problem

For the proof of this see Hromkovič [2002].

Summary

This section concerns a specific problem of finding an optimal network when the model of a network is a set of mutually exclusive sets, which is the same as finding an *exact cover* of the elements. I first present a naive algorithm, which solves this through a greedy search. As this method cannot find an exact solution, I define a simple bounding condition, which allows for a branch and bound search method. I then present another algorithm which rather than adding elements one by one, instead generate a set of candidate sets and adds sets to the solution. I present a bounding condition for set generation and a results based on components, which can be used to reduce the number of candidate sets. I convert the task of finding the sets which constitute an exact cover into an *integer programming* problem, which can be efficiently solved by relaxing it to a linear programming problem. Finally, I briefly the describe the *Simplex algorithm*, which is capable of solving linear programming problems.

4.3 Simple Logic

I would like to be able to extend my system to be able to handle other kinds of structures than just the equivalence structure, most notably I would like to be able to handle hierarchical structures. When it comes to studying hierarchical structures, there are a number of potential variations on these structures. Firstly, hierarchies often have a fixed root element from which all other elements are descended or may have a fixed number of such roots. In addition, often the

hierarchy should not be a hierarchy of terms, but a hierarchy of sets of term, so I need a way to be able to combine my method with the equivalence structures found above. For these reasons I choose not to take a simple approach to hierarchy but to instead to look at the more general problem by approaching forms limited by logical rules or *axioms*. I can then state the hierarchy problems by assuming the system satisfies at least the following rules

4.3.1: Definition A hierarchical structure satisfies at least the following axioms

- $\forall i,j,k\ r(e_i,e_j) \wedge r(e_j,e_k) \rightarrow r(e_i,e_k)$. This is called *transitivity*
- $\forall i \neg r(e_i,e_i)$. This is called *irreflexivity*.
- There are no loops in the graph.

It is trivial to show that the third property of the graph is implied by the other two as if there was a set $\{e_{i_1},\dots e_{i_n}\}$ such that all links $r(e_{i_j},e_{i_{j+1}})$ and $r(e_{i_n},e_1)$ exist, then it implies that $r(e_{i_1},e_{i_2)} \wedge r(e_{i_2},e_{i_3}) \rightarrow r(e_{i_1},e_{i_3})$ can be used to show that the set $\{e_{i_1},e_{i_3},\dots e_{i_n}\}$ has the same property and hence by induction $r(e_{i_1},e_{i_1})$ holds.

I can also restate the equivalence problem in a similar way

4.3.2: Lemma An equivalence structure is defined as

- $\forall i,j,k$: $r(e_i,e_j) \wedge r(e_j,e_k) \rightarrow r(e_i,e_k)$
- $\forall i,j$: $r(e_i,e_j) \rightarrow r(e_j,e_i)$. This is called *symmetry*
- $\forall i$: $r(e_i,e_i)$. This is called *reflexivity*

If $\{e_{i_1},\dots e_{i_n}\}$ is a set such that for all j,k with $1 \leq j < k \leq n$, then $\neg r(e_{i_j},e_{i_k})$ and for all $e \in E$ there exists k, such that $r(e,e_{i_k})$ holds, then the set $\{C_1 \dots C_n\}$ given by $C_j = \{e \in E | r(e_{i_j},e)\}$ is an exact cover of E.
Proof: It is clear that for all $e \in E$, $\exists i$ such that $e \in C_i$, as we have some k such that $r(e,e_{i_k})$ hence $e \in C_k$. If we have j,k, $j \neq k$ such that there exists $e \in C_j \cap C_k$ then we have for some $r(e,e_{i_j})$ and $r(e,e_{i_k})$ hence it follows we have $r(e_{i_j},e_{i_k})$.$\square$

This shows that for structures constrained by the rules as above the problem is the same as finding an exact cover.

So to provide a more general solution it is necessary to find rules in a general logical form. I would like to limit this to the form which is most useful for general problems

4.3.1 Simple Logic

I define a logical system I call "simple logic" to specify the problem of handling structures such as hierarchical structures. This logic has the property that when it is *grounded* (all variables replaced with elements), then the form of the logic is immediately equivalent to that of *prepositional logic*.

If $N(E, R)$ is a set of networks, define a rule as follows

```
rule := expr → expr
expr := (term (,term)*)?
term :=  r (var,var) where r ∈ R
var :=  x_1,...
```

4.3.1.1: Definition If there is a rule which contains exactly the variable $x_1 \dots x_m$ then a substitution is of the form $(x_1 \leftrightarrow e_{i_1}, \dots x_m \leftrightarrow e_{i_m})$. Define the set of all possible substitutions $S_{\{x_1 \dots x_m\}}$ of a single variable. Also define a *ground instance* of a rule as a version of this rule where every variable has been replaced with the appropriate element.

4.3.1.2: Definition If there is a ground instance, t, given by

$$t = r_1(e_{i_1}, e_{i'_1}) \dots r_m(e_{i_m}, e_{i'_m}) \rightarrow r_{m+1}(e_{i_{m+1}}, e_{i'_{m+1}}) \dots r_n(e_{i_n}, e_{i'_n})$$

Define the premise set of t, $premise(t) = \{r_1(e_{i_1}, e_{i'_1}) \dots r_m(e_{i_m}, e_{i'_m})\}$ and the conclusion set of t as $conclusion(t) = \{r_{m+1}(e_{i_{m+1}}, e_{i'_{m+1}}), \dots r_n(e_{i_n}, e_{i'_n})\}$.

4.3.1.3: Example Take the transitivity rule $r(x_1, x_2), r(x_2, x_3) \rightarrow r(x_1, x_3)$ and the set of elements is $E = \{animal, mammal, cat\}$ then we can apply the substitution $(x_1 \leftrightarrow animal, x_2 \leftrightarrow mammal, x_3 \leftrightarrow cat)$ to obtain the ground instance $r(animal, mammal), r(mammal, cat) \rightarrow r(animal, cat)$. It premise is $\{r(animal, mammal), r(mammal, cat)\}$ and conclusion is $\{r(animal, cat)\}$.

4.3.1.4: Definition Define a ground instance, t, to be *tautologous* if there exists $r(e, e')$ such that $r(e, e') \in premise(t)$ and $r(e, e') \in conclusion(t)$. Say that ground instance t subsumes t' if $premise(t) \subseteq premise(t')$ and $conclusion(t) \subseteq conclusion(t')$.

4.3.1.5: Definition A ground instance

$$r_1(e_{i_1}, e_{i'_1}) \dots r_m(e_{i_m}, e_{i'_m}) \rightarrow r_{m+1}(e_{i_{m+1}}, e_{i'_{m+1}}) \dots r_n(e_{i_n}, e_{i'_n})$$

Is *inconsistent* with a network $N \in N(E, R)$ if it holds that for $j = 1 \dots m$ $r_j(e_{i_j}, e_{i'_j}) \in N$ and there does not exist j with $m < j \leq n$ with $r_j(e_{i_j}, e_{i'_j}) \in N$. Define a rule to be consistent with a network if all possible substitutions of that rule are consistent with the network.

4.3.1.6: Definition If there are a set of rules, T, define a set of networks $V_T \subset N(E,R)$, such that $N \in V_T$ if every rule in T is consistent with N.

4.3.1.7: Definition Define a *linear simple logic consistency problem* as a tuple $(E, R, T, P, \mathbf{c})$ where T is a set of rules with relations in R, P is the network with minimal value for $c(P)$ and $\mathbf{c}$ is a set of positive costs corresponding to the values $r_i(e_j, e_k)$ such that

$$c(r_i(e_j,e_k)) = |log(p(r_i(e_j,e_k))) - log(1 - p(r_i(e_j,e_k)))|$$

And the solution is $N \in V_T$, which is a minimal network in V_T with respect to

$$c(N) = \sum_{r_i(e_j,e_k) \in N \ominus P} c(r_i(e_j,e_k))$$

4.3.2 Greedy algorithm:

I will now explain a greedy approach to this problem, which can be used as a baseline. It can be shown that this will produce a solution for certain sets of rules but not all. Starting with the network $N = \emptyset$ and this method attempts to add links in a one by one order.

4.3.2.1: Definition The consequence set of link $C_{N,T}(r(e_i,e_j))$ is defined as such

- $r(e_i,e_j) \in C_{N,T}$
- If there is a ground instance of a rule in T, t, such that $premise(t) \subseteq C_{N,T} \cup N$ then $conclusion(t) \subseteq C_{N,T}$.

It is clear that this consequence set is well-defined and can be readily computed by simply initialising the set to $r(e_i,e_j)$ and repeatedly looking for rules which are inconsistent with the network $C_{N,T} \cup N$ and adding all conclusions of such rules.

4.3.2.2: Definition The *add cost* of a set $C \subset N(E,R)$ to a network N is $addcost_N(C) = \sum_{r(e_1,e_2) \in C} c(r(e_1,e_2))$.

4.3.2.3: Algorithm
Input: A linear simple logic consistency problem, $(E, R, T, P, \mathbf{c})$
Output: A network, N

1. Let $N \to \emptyset$.
2. Choose a link $r(e_1,e_2)$ that maximises $addcost_N(C_{N,T}(r(e_i,e_j)))$
3. If $addcost_N(C_{N,T}(r(e_i,e_j))) < 0$ **output** N

4. Let $N \rightarrow N \cup C_{N,T}(r(e_i, e_j))$

5. **Goto** step 2.

4.3.2.4: Lemma The network outputted by algorithm 4.3.2.3 is consistent with the rule set T, if every rule in T has a conclusion of size 1 and $\emptyset \in V_T$.
Proof: The proof of this is as simple observing that if we have a network N and $C_{N,T}(r(e_i, e_j))$ then any ground instance, which has all its premises in $N \cup C_{N,T}(r(e_i, e_j))$ must have all of its conclusions in $N \cup C_{N,T}(r(e_i, e_j))$ so there cannot be any inconsistent ground instances. By induction this means that every network generated at each iteration of the algorithm is consistent.□

It is clear that this means this algorithm is suitable for equivalence problems and nearly identical in its implementation to algorithm 4.2.1.1, however as the irreflexivity axiom translates to a rule

$$r(e_1, e_1) \rightarrow$$

It is necessary to include a small caveat in step 2 of the algorithm that a link $r(e_i, e_j)$ where $C_{N,T}(r(e_i, e_j))$ contains a link of the form $r(e_k, e_k)$ for some k cannot be chosen. It is clear that this restriction is sufficient as it stops the algorithm adding links, which would create a loop in the structure.

It is easy to demonstrate with a simple example that this algorithm does not find the optimal solution in general, however as with the equivalence algorithm it is possible to extend the procedure to a branch and bound method in order to find an algorithm, which can find the optimal solution.

4.3.3 Consistency based approach

To find a network consistent with these restrictions I approach the problem by finding a method to correct the original solution. This is done by first taking the graph that with no restrictions would score the maximal value for $c(N)$ and then attempting to "correct" it by adding or removing links, which are inconsistent with the rule set.

4.3.2.1: Definition The *inconsistent set* of a network N is the set $inconsist_T(N)$ which contains all ground instances of rules in T, which are inconsistent with N.

4.3.3.2: Lemma If $t \in inconsist_T(N)$ then and $r(e_i, e_j) \in premise(t) \cup conclusion(t)$ then $t \notin inconsist_T(N \ominus \{r(e_i, e_j)\})$

This result means that to "correct" a solution it is possible to simply add a conclusion from an inconsistent ground instance or remove a premise. This

could lead to an algorithm, which organises this into a branch and bound search, however better results can be achieved by a linear approach as before.

4.3.3.4: Definition If there is a set of ground instances G it is possible to form a matrix A_G from it, where the columns of G corresponding to links and the rows to instances from G as follows $A_G(r(e_i, e_j), g) = 1$ if $r(e_i, e_j) \in premise(g)$ or $r(e_i, e_j) \in conclusion(g)$, and $A_G(r(e_i, e_j), g) = 0$ otherwise. Call the matrix $A_{inconsist_T(N)}$ the correction matrix of N, henceforth denoted A_N

4.3.3.5: Lemma If there is a vector such that $A_N\mathbf{x} \geq 1$ then the network given by $N \ominus \mathbf{x}$[2] is consistent with all the rules from $inconsist_T(N)$.
Proof: Follows from Lemma 4.3.3.2

4.3.3.6: Example Assume $N = \{r(e_1, e_2), r(e_2, e_3), r(e_3, e_4)\}$ and we are using the hierarchical rules

- $r(x_1, x_2), r(x_2, x_3) \rightarrow r(x_1, x_3)$
- $r(x_1, x_1) \rightarrow$

Then it is clear we have the following $inconsist_T(N)$ is the following rules

- $t_1 = r(e_1, e_2), r(e_2, e_3) \rightarrow r(e_1, e_3)$
- $t_2 = r(e_2, e_3), r(e_3, e_4) \rightarrow r(e_2, e_4)$

So we can form the following matrix for A.

	$r(e_1, e_2)$	$r(e_2, e_3)$	$r(e_3, e_4)$	$r(e_1, e_3)$	$r(e_2, 3e_4)$
t_1	1	1	0	1	0
t_2	0	1	1	0	1

If we take $\mathbf{x} = (0, 0, 0, 1, 1)$, which has $A_N\mathbf{x} \geq 1$. Then we get a final network $N \ominus \mathbf{x} = \{r(e_1, e_2), r(e_2, e_3), r(e_3, e_4), r(e_1, e_3), r(e_2, e_4)\}$, which is clearly consistent with the two rules from $inconsist_T(N)$ however it is not consistent in itself as the rule $r(e_1, e_3), r(e_3, e_4) \rightarrow r(e_1, e_4)$ is not consistent with $N \ominus \mathbf{x}$. So my approach here cannot at the moment find a consistent network and I need a way to be able to add the new rules.

The simple approach of trying to solve $A_{N\ominus\mathbf{x}}$ will not work in general as it leads to loops. Say for example we choose the vector $\mathbf{x} = (0, 0, 0, 1, 1)$ as it minimised the cost vector $\mathbf{c} = (2, 3, 2, 1, 1)$, then it should be clear that the cost vector $A_{N\ominus\mathbf{x}}$ is $\mathbf{c} = (2, 3, 2, -1, -1, ?)$ (the ? corresponds to the cost for $r(e_1, e_4)$). This means that the minimal solution for this approach is to return to the original matrix N, hence this method will loop infinitely! As such I need a way to add new rows to the matrix, I will show two approaches for this, one which is simpler but makes the linear programming problem harder and one which is more complex but leads to a more solvable matrix.

[2] $N \ominus \mathbf{x} \equiv N \ominus \{r_i(e_j, e_k) | x(r_i(e_j, e_k)) = 1\}$

4.3.4 Expanding the matrix by negative values

The goal here is to create a matrix that can be used to state a linear programming problem of the form $A\mathbf{x} \geq \mathbf{b}$, which is equivalent to solving the problem of finding a network $N \ominus \mathbf{x}$, which is consistent with a set of ground instances T.

4.3.4.1: Definition Define a matrix A_T^- whose rows correspond to the ground instances of some set $T = \{t_1, \ldots t_n\}$ and columns to links from N. Consider $r(e, e')$ is the j^{th} element, then I define the matrix as

- If $r(e, e') \in \mathit{premise}(t_i)$ and $r(e, e') \in N$ then $a_{ij}^- = 1$.
- If $r(e, e') \in \mathit{premise}(t_i)$ and $r(e, e') \notin N$ then $a_{ij}^- = -1$.
- If $r(e, e') \in \mathit{conclusion}(t_i)$ and $r(e, e') \notin N$ then $a_{ij}^- = 1$.
- If $r(e, e') \in \mathit{conclusion}(t_i)$ and $r(e, e') \in N$ then $a_{ij}^- = -1$.
- $a_{ij}^- = 0$ otherwise.

I also define a vector $\mathbf{b}$ by

$$\begin{aligned} b_i = \quad & 1 - |\{r(e, e') | r(e, e') \in \mathit{premise}(t_i) \wedge r(e, e') \notin N\}| - \\ & |\{r(e, e') | r(e, e') \in \mathit{conclusion}(t_i) \wedge r(e, e') \in N\}| \end{aligned}$$

That is b_i is 1 minus the number of negative values in its corresponding row

4.3.4.2: Theorem $\mathbf{x}$ is a vector such that $A_T^-\mathbf{x} \geq \mathbf{b}$ if and only if $N \ominus \mathbf{x}$ is consistent with all ground instances in T.
Proof: I have already shown the case for columns that have no negative values, and that these must correspond to a rule t_i which is not consistent with N. Assume t_i is consistent with N then t_i is inconsistent with $N \ominus \mathbf{x}$ if and only if $\mathbf{x}$ has all the premises of t_i, which are not in N, and all the conclusions, which are, and doesn't have a premise, which is not in N, or a conclusion, which is in N. This means that for the row corresponding t_i $\mathbf{x}$ must have a 1 only where the column has a -1, this means that the product for this column is $-|\{r(e, e') | r(e, e') \in \mathit{premise}(t_i) \wedge r(e, e') \notin N\}| - |\{r(e, e') | r(e, e') \in \mathit{conclusion}(t_i) \wedge r(e, e') \in N\}|$. Hence $A_T^-\mathbf{x} \not\geq \mathbf{b}$.□

4.3.4.3: Example Extending example 4.3.3.6 further we have two new ground inconsistencies, which we need to add; these are

- $t_3 = r(e_1, e_2), r(e_2, e_4) \rightarrow r(e_1, e_4)$
- $t_4 = r(e_1, e_3), r(e_3, e_4) \rightarrow r(e_1, e_4)$

It is clear that this gives a matrix as follows

$$\left(\begin{array}{c|cccccc|c} & r(e_1,e_2) & r(e_2,e_3) & r(e_3,e_4) & r(e_1,e_3) & r(e_2,3e_4) & r(e_1,e_4) & \mathbf{b} \\ \hline t_1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ t_2 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ t_3 & 1 & 0 & 0 & 0 & -1 & 1 & 0 \\ t_4 & 0 & 0 & 1 & -1 & 0 & 1 & 0 \end{array}\right)$$

4.3.4.4: Algorithm
Input: An *LSLC* problem, $(E, R, T, P, \mathbf{c})$
Output: A network N which solves the LSLC problem.

1. Let T^0 be the ground instances inconsistent with N. Let $i \leftarrow 0$.
2. Find $\mathbf{x}$ that minimises $\mathbf{c}^T\mathbf{x}$ subject to $A^-_{T^i}\mathbf{x} \geq 1$.
3. If $N \ominus \mathbf{x}$ is consistent **output:** $N \ominus \mathbf{x}$.
4. Otherwise add all new inconsistencies to matrix as in definition 4.3.4.1.
5. Let $i \leftarrow i + 1$ and go to step 2.

This method is simple and easy-to-implement however I will now present a more complex methodology, which can greatly improve the performance of my system.

4.3.5 Expanding the matrix with resolution

4.3.5.1: Definition Say there are two ground instances t_1, t_2 such that $r(e, e') \in premise(t_1)$ and $r(e, e') \in conclusion(t_2)$. Then call the rule t_r the *resolvent* of t_1 and t_2 where

- $premise(t_r) = (premise(t_2) - \{r(e, e'\}) \cup premise(t_2)$
- $conclusion(t_r) = conclusion(t_1) \cup (conclusion(t_2) - \{r(e, e')\})$

So I can now develop an algorithm that instead of adding negative values to the matrix removes the need for them by resolving.

4.3.5.2: Definition A *correction vector* is a vector $\mathbf{x}$ such that if T are the ground instances inconsistent with a network N then $N \ominus \mathbf{x}$ is consistent with all ground instances of T

Ihave already shown in Lemma 4.3.3.5 that all correction vectors also satisfy $A_T\mathbf{x} \geq 1$.

4.3.5.3: Definition If there is a network N and a correction vector $\mathbf{x}$ then call $\mathbf{x}$ a *sufficient correction vector* if there exists no correction vector $\mathbf{x}'$ with $\mathbf{x}' \subset \mathbf{x}$ (by which I mean that if $x_i = 0$ then $x'_i = 0$ and there is some i such that $x_i = 1$ and $x'_i = 0$).

This definition essentially states a certain kind of *Occam's Razor* in that a sufficient correct vector changes no more values than are strictly necessary.

4.3.5.4: Lemma If $\mathbf{x}$ is such that $\mathbf{x}$ minimises $\mathbf{c}^T\mathbf{x}$ subject to $A_T\mathbf{x} \geq 1$ then $\mathbf{x}$ is a sufficient correction.

Proof: $\mathbf{c}$ is positive hence it follows that if $\mathbf{x}$ is not sufficient then the vector $\mathbf{x}'$ (as in definition 4.3.5.3) is also a solution to $A_T\mathbf{x} \geq 1$ and $\mathbf{c}^T\mathbf{x}' < \mathbf{c}^T\mathbf{x}$.□

4.3.5.4: Lemma If $\mathbf{x}$ is a sufficient correction vector of N then for each $r(e, e')$ which corresponds to a 1 in $\mathbf{x}$ there is a ground instance t, which is inconsistent with N and has $r(e, e')$ in its premise or conclusion.

Proof: If this were not true we would have a vector $\mathbf{x}'$ with exactly the same value as $\mathbf{x}$ in all places except $r(e, e')$ and this vector must be a correction vector as $r(e, e')$ does not occur in any rule in T.□

4.3.5.6: Theorem Assume there is a network N and a sufficient correction vector $\mathbf{x}$ and a ground instance t_1 which is consistent with N but not $N' = N \ominus \mathbf{x}$ then there exists a set of ground instance T' such that each $t_i \in T'$ is consistent with N' but not N. There is some chain of resolvents such that obtains a rule t' such that t' is inconsistent with both N', and has $premise(t') \subseteq N$ and $conclusion(t') \cap N = \emptyset$ (hence is inconsistent with N).

Proof: For t_1 consider $r(e, e')$ which is in the $premise(t_1)$ and has a 1 in the corresponding vector $\mathbf{x}$, there must be some t_2 which is not consistent with N and is with N' and contains $r(e, e')$ and this $r(e, e')$ must be in the conclusion of t_2 as $r(e, e') \in N'$ and hence $r(e, e') \notin N$ hence t_2 can only be inconsistent with N if $r(e, e') \in conclusion(t_2)$. Resolve t_1 and t_2 around $r(e, e')$ to gain a new rule t' it is clear that this process can be repeated for each $r(e, e')$ in the premise of t_1 with a 1 for $\mathbf{x}$ and then a similar process carried out for the $r(e, e')$ in the conclusions of t_1. It should be clear that the result has the required property as every rule in T does and we have eliminated all terms from t_1 which do not have the required property.□

Informally stated the previous theorem shows that whenever a inconsistency is found it is possible to find a chain of resolvents with inconsistent ground instances from previous iterations, such that the new ground instance is inconsistent with the base network, and hence can be readily added to the matrix.

4.3.5.7: Example Say $N = \{r(e_1, e_2), r(e_2, e_3), r(e_3, e_4), r(e_4, e_5)\}$ and we use the hierarchical logic so we have that T has the following ground instances inconsistent with N

1. $r(e_1, e_2), r(e_2, e_3) \rightarrow r(e_1, e_3)$
2. $r(e_2, e_3), r(e_3, e_4) \rightarrow r(e_2, e_4)$
3. $r(e_3, e_4), r(e_4, e_5) \rightarrow r(e_3, e_5)$

Now we can take the correction vector $\mathbf{x}$ equivalent to $\{r(e_1, e_3), r(e_2, e_4)r(e_3, e_5)\}$. One of the ground inconsistencies with $N \ominus \mathbf{x}$ is

$$r(e_1, e_3), r(e_3, e_5) \rightarrow r(e_1, e_5)$$

This can be solved by first resolving with t_1 to get

$$r(e_1, e_2), r(e_2, e_3), r(e_3, e_5) \rightarrow r(e_1, e_5)$$

and then with t_2 to get

$$r(e_1, e_2), r(e_2, e_3), r(e_3, e_4), r(e_4, e_5) \rightarrow r(e_1, e_5)$$

Which has the property that all its premises are in N and none of its conclusions are.

This method makes it possible to keep adding new rules to the matrix by adding their resolvents so a matrix can be developed that has no negative values, allowing for some extra optimisations to be applied.

4.3.5.8: Algorithm
Input: An *LSLC* problem, $(E, R, T, P, \mathbf{c})$
Output: A network N which solves the LSLC problem.

1. Let T^0 be the ground instances inconsistent with N. Let $i \leftarrow 0$.
2. Find $\mathbf{x}$ that minimises $\mathbf{c}^T\mathbf{x}$ subject to $A_{T^i}\mathbf{x} \geq 1$.
3. If $N \ominus \mathbf{x}$ is consistent **output:** $N \ominus \mathbf{x}$.
4. Otherwise for each ground instance t, which is inconsistent with $N \ominus \mathbf{x}$ find by the above process a non-tautologous ground instance t', which is inconsistent with N and not already in T^i or subsumed by a rule in T^i, and add it to T^{i+1}.
5. Let $i \leftarrow i + 1$ and go to step 2.

I will now show that the algorithm 4.3.5.8 produces the true minimum across all solutions, given the assumption of the cost function from definition 4.1.2.

4.3.5.9: Theorem Algorithm 4.3.5.8 solves LSLC problems
Proof: It should be clear by the stopping condition of the algorithm produces a network that is consistent with all restrictions. Suppose we have a vector $\mathbf{x}'$,

which gives a consistent network $N \ominus \mathbf{x}'$ and has $\mathbf{c}^T\mathbf{x}' < \mathbf{c}^T\mathbf{x}$. Now consider the vector $\mathbf{x}''$ with $x_i'' = x_i'$ if i corresponds to a term occurring in premise or conclusion of a rule in T^j (where j is the final iteration of the algorithm), and $x_i'' = 0$ otherwise. It should be clear that this vector has $A_{T^i}\mathbf{x}'' \geq 1$ and $\mathbf{c}^T\mathbf{x}'' \leq \mathbf{c}^T\mathbf{x}'$ and but such a vector cannot exist as $\mathbf{x}$ is the minimal solution to the problem in step 2 of the algorithm.□

In fact it is unnecessary to do the resolution in a symbolic manner, as it can easily be done as a linear procedure as follows

4.3.5.10: Algorithm
Input: A ground instance t, which is consistent with N but not $N \ominus \mathbf{x}$, where $\mathbf{x}$ is a sufficient correction vector of A (corresponding to N)
Output: A matrix A' with t added "by resolution"

1. Let t be the new inconsistent ground instance, add it to A using the method 4.3.4.1 to get A'.
2. For each -1 in the row given by r, find a row, t' that has a 1 in the same column and doesn't have a 1 anywhere else t's row has a -1.
3. Add the values of row r' to r, and then set any values greater than 1 to 1.
4. Output A'.

It is necessary to show that the rows required in step 2 exist

4.3.5.11: Lemma The row of step 2 of algorithm 4.3.5.10 exists.
Proof: It is simple to observe that a row with a 1 in the correct place exists as otherwise $\mathbf{x}$ cannot be a sufficient correction vector as there would be vector identical to $\mathbf{x}$ except without a 1 in the corresponding place. Similarly the condition that this row doesn't have multiple 1s which match up with r's -1s, also implies that there exist multiple vectors, which would make $\mathbf{x}$ not sufficient.

It should be clear that this process is simple and removes the need for any symbolic resolution to be performed.

4.3.5.12: Example Suppose there is a matrix

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Which corresponds to the rules (for here I shall denote links as just r_i instead $r(e_1, e_2)$ for simplicity)

$$r_1 \rightarrow r_2$$
$$r_3 \rightarrow r_2$$

Take the sufficient correction vector $\mathbf{x} = (1\ 0\ 1)^T$, and we get a new inconsistency

$$t = r_4 \rightarrow r_1, r_3$$

Then it can be added by negative values to the matrix to get

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ -1 & 0 & -1 & 1 \end{pmatrix}$$

Then adding row 1 to row 3 gives

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{pmatrix}$$

And adding row 2 to row 3 gives

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

This is the same effect as resolution which would have given a rule

$$r_4 \rightarrow r_2$$

I shall make a quick note about the behaviour of the algorithm when the set of ground instances are not satisfiable, that is there $V_T = \emptyset$.

4.3.5.13: Lemma If there exists a set of ground instances, T, such that $V_T = \emptyset$, then there is some sequences of resolution that lead to the empty clause $\rightarrow$.
Proof: Consider if all possible resolvents of T, were added to a matrix A if none of them were empty then it is clear that $\mathbf{x} = (1 \ldots 1)^T$ has that $A\mathbf{x} \geq \mathbf{1}$, hence there exists a sufficient correction vector $\mathbf{x}'$ which gives a consistent solution.□

4.3.5.14: Example Suppose

$$t_1 = r_1 \rightarrow r_2$$
$$t_2 = \rightarrow r_1$$
$$t_3 = r_2 \rightarrow$$

Then resolving t_1 with t_2 gives $t' = \rightarrow r_2$ and resolving that with t_3 gives the empty clause this must be added to the matrix as a row of all zeros, and it is clear that

$$0x_1 + \ldots + 0x_n \not\geq 1$$

Hence this matrix has no correction vectors. Similarly for algorithm 4.3.4.4, suppose we have $N = r_1$, then it is clear that adding all rules gives the matrix problem as

$$\begin{pmatrix} 1 & 1 \\ 0 & -1 \\ -1 & 0 \end{pmatrix} \mathbf{x} \geq \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

And it is clear that this matrix problem has no solutions.

4.3.6 Matrix optimisation

If there is a matrix A_T which is derived from a set of ground instances inconsistent with T and the cost function is known, then many of the choices can be very quickly eliminated for on straightforward reasons.

4.3.6.1: Lemma If there is a rule t and another rule t' and t subsumes t' then any correction vector $\mathbf{x}$ also is a correction vector of $A_{T-\{t'\}}$.

This is clear as t' must be satisfied if t is. This can be stated in the form of the first matrix reduction principle.

4.3.6.2: Row reduction principle If there exists i, j such that for all k if $a_{ik} = 1$ then $a_{jk} = 1$ then it is possible to remove row j.

This should rarely occur in the course of normal operation of the algorithm, but it will do if the number of columns is also reduced.

4.3.6.3: Lemma If there is a link $r_1(e, e')$ and a link $r_2(f, f')$ and for every rule t in T, if $r_1(e, e')$ occurs in t then $r_2(f, f')$ occurs in t and there is a correction vector $\mathbf{x}$, which has a 1 corresponding to $r_1(e, e')$, then the vector obtained from $\mathbf{x}$ by setting the value at $r_1(e, e')$ to 0 and the value at $r_2(f, f')$ to 1 is also a correction vector.
Proof: This is simply shown by recalling that it is only necessary to change one link in a rule to make it consistent with N, so now if we change $r_2(f, f')$ instead of $r_1(e, e')$, it makes at least the same rules from T consistent hence it follows that this is also a correction.□

4.3.6.4: Column reduction principle If there exists i, j such that for all k if $a_{ki} = 1$ then $a_{kj} = 1$ and for some cost vector $\mathbf{c}$ it holds that $c_j > c_k$, then it is possible to remove column j.

I can now present the matrix reduction algorithm

4.3.6.5: Algorithm
Input: A matrix A with $a_{ij} \in \{0, 1\}$ for all values, and a positive vector $\mathbf{c}$.

Output: A matrix A' such that if $\mathbf{x}$ minimises $\mathbf{c}^T\mathbf{x}$ subject to $A^T\mathbf{x} \geq 1$ then $A'^T\mathbf{x} \geq 1$.

1. Find all columns which can be reduced, and remove them (set all the values in that column to zero).
2. Find all rows which can be reduced, and remove them.
3. If the matrix has changed **goto** Step 1.

4.3.6.6: Example Say we have the matrix and cost vector as below

	$r(e_1, e_2)$	$r(e_2, e_3)$	$r(e_3, e_4)$	$r(e_1, e_3)$	$r(e_2, 3e_4)$	$r(e_1, e_4)$
t_1	1	1	0	1	0	0
t_2	0	1	1	0	1	0
t_3	1	1	1	0	0	1
$\mathbf{c}$	3	4	5	1	6	6

We can see that the columns corresponding to $r(e_2, e_4)$, $r(e_1, e_4)$ and $r(e_1, e_2)$ can all be removed as they cost more than $r(e_3, e_4)$, this obtains the following matrix

	$r(e_1, e_2)$	$r(e_2, e_3)$	$r(e_1, e_3)$
t_1	1	1	1
t_2	0	1	0
t_3	1	1	0
$\mathbf{c}$	3	4	1

Now we can remove the row corresponding to t_1 as it is subsumed by t_2.

	$r(e_1, e_2)$	$r(e_2, e_3)$	$r(e_1, e_3)$
t_2	0	1	0
t_3	1	1	0
$\mathbf{c}$	3	4	1

This leads us to remove $r(e_1, e_3)$ as an empty solution

	$r(e_1, e_2)$	$r(e_2, e_3)$
t_2	0	1
t_3	1	1
$\mathbf{c}$	3	4

Finally we remove t_2, then $r(e_1, e_2)$ to get the trivial matrix

	$r(e_2, e_3)$
t_2	1
$\mathbf{c}$	4

This states that the optimal solution is $\mathbf{x} = (0, 1, 0, 0, 0, 0)^T$

It is important to note that when using this algorithm in the resolution based process, these matrix reductions are only useful for solving a single matrix, once the optimal solution has been found by simplex or similar approaches, it is necessary to restore the matrix to its original form before attempting to resolve and add new rules.

4.3.7 MAX-SAT approaches

The weighted maximum satisfiability problem (Weighted MAX-SAT) is a very similar problem and I will show that it is in fact possible to represent my problem as a special form of weighted MAX-SAT. I will also show that for several reasons approaching the problem as a weighted MAX-SAT problem is not as efficient as my specialised approach.

4.3.7.1: Definition Say there is a set of ground instances T and for each ground instance there is a cost function $c : T \to \mathbb{R}$, then define a maximum satisfaction of T as a set $T' \subseteq T$ such that $V_{T'} \neq \emptyset$ and is maximal w.r.t.

$$\sum_{t \in T'} c(t)$$

That is there does not exist $T'' \subseteq T$, which has $V_{T''} \neq \emptyset$ and has

$$\sum_{t \in T''} c(t) > \sum_{t \in T'} c(t)$$

It is clear that the primary difference between weighted MAX-SAT and my problem is that in my problem I assign the costs to the links, i.e., the terms of the rules, where as in weighted MAX-SAT it is assigned to the ground instances. It is not in general possible to find a way to convert costs from my problem into costs in a weighted MAX-SAT problem, however I can get around this by adding further ground instances to describe the problem.

4.3.7.2: Definition For each link $r_i(e_j, e_k)$ define the cost of this link as

$$c(r_i(e_j, e_k)) = log(p(r_i(e_j, e_k)) - log(1 - p(r_i(e_j, e_k)))$$

Then as in definition 4.3.1.1, it is easy to see that

$$log(p(N)) = log(P(\emptyset)) + \sum_{r_i(e_j,e_k) \in N} c(r_i(e_j, e_k))$$

I also define the *maximal cost* M as

$$M = \sum_{r_i(e_j,e_k) \in \mathcal{N}} |c(r_i(e_j, e_k))|$$

4.3.7.3: Lemma For all $N \in N(E, R)$ it holds that

$$M \geq log(p(N))$$

This lemma follows directly from the definitions above. I can now restate the problem as a weighted MAX-SAT problem

4.3.7.4: Definition If T is the set of ground instances and then let T_b be the set given by ground instances of the form

$$r_i(e_j, e_k)$$

Such that there exists one ground instance for each $r_i(e_j, e_k) \in \mathcal{N}$. Then define a function $c : T \cup T_b \rightarrow \mathbb{R}$ such that for $r_i(e_j, e_k) \in T_b$, c is as in definition of 4.3.7.2 and for $t \in T$, $c(t) = M$

4.3.7.5: Theorem If T' is the maximum satisfaction of $T \cup T_b$ weighted by c then the network given by $T' \cap T_b$ is the solution to the problem stated in solution 4.3.1.7 or $V_T = \emptyset$ (T has no consistent solutions).
Proof: Suppose there is a network $N \in V_T$ it is clear that there is a solution given by $N \cup T$ as we have that N is consistent with T, this means that the maximal solution has a cost of at least $|T| \times M$, we can also see from the definition of M that any subset of $T \cup T_b$ has a cost less than $(|T| + 1) \times M$. From this we can deduce that the maximal solution must contain T, and it is clear from Lemma 4.3.7.3 that this maximal solution maximises the same quantity that the cost function of definition 4.3.1.7 minimises.□

It is clear that this method is correct but in most cases will result in a problem that is far to large to deal with, so for the same reasons as for the previous approach I will limit the method to only a small section of the rules.

4.3.7.6: Definition Let P be a network where $r_i(e_j, e_k) \in P$ if $c(r_i(e_j, e_k)) > 0$, let T_P be the ground instances inconsistent with P, let R_P be the set of terms occurring in ground instances of P.

4.3.7.7: Algorithm
Input: A set of ground inconsistencies T on a network given by relations R and elements E, and cost function c on this network
Output: A network N, which is maximal w.r.t. theorem 4.3.7.5

1. Let $T' \leftarrow T_P \cup R_p$ as in definition 4.3.7.6

2. Let N be the network gained by finding the weighted maximum satisfaction of T'.
3. If N is consistent with T **output** N.
4. Let $T' \leftarrow T' \cup T_N \cup R_N$.
5. **Go to** Step 2.

4.3.7.8: Example Assume we have $E = \{e_1, e_2, e_3\}$, $R = \{r\}$ and we have that $T = \{r(e_1, e_2), r(e_2, e_3) \rightarrow r(e_1, e_3)$ and c defined by

c	e_1	e_2	e_3
e_1	0	-1	-2
e_2	-1	0	-3
e_3	2	3	0

Now we get our problem as finding the maximum weighted satisfaction of

$r(e_1, e_2)$	-1
$r(e_2, e_3)$	3
$r(e_1, e_3)$	2
$r(e_1, e_2), r(e_2, e_3) \rightarrow r(e_1, e_3)$	12

While this shows that using an "off-the-shelf" MAX-SAT algorithm is possible in theory, there are a number of issues that make most MAX-SAT algorithms unsuitable for this problem

- **Very large clause count:** Consider the transitivity rule given as

  $$r(x_1, x_2), r(x_2, x_3) \rightarrow r(x_1, x_3)$$

  For a reasonably sized problem of $|E| = 1000$, there are a billion ground instances (although technically 1000 are trivial). This means that any algorithm such as *UniSAT*, [Gu, 1992] which relies on global optimisation across all clauses are not suitable.[3]

- **Assignment of costs:** As most clauses in the system will have the same cost, i.e., M, many algorithms will have particular problem in deciding which clause to add at which time. Furthermore, as the costs are only on unit clause this is very difficult for any algorithm based on flow algorithms of Yannakakis [1992][4]

[3] UniSAT works by converting logical clauses into an algebraic form and solving as an algebraic problem

[4] This relies on viewing unit clauses as "sources" and "sinks", and plotting the "flow" through the system of clauses. For my problem this is not applicable as all limits on the "flow" would occur at the "sources" or "sinks"

- **Solution must be consistent:** It is required that the solution gives a network is consistent with all ground instances. Approximate weighted MAX-SAT algorithms cannot guarantee this unlike approximate algorithms designed for my problem.

- **High density of correct solution:** As my problem has very few base clauses, but give a very large number of potential consistent networks. For example, for the equivalence logic this is given by the *Bell Number* [Rota, 1964]

$$|V_T| = B_n = \sum_{k=0}^{n} \left(\begin{array}{c} n \\ k \end{array} \right) B_k$$

  This means that it is highly likely that the optimal consistent network can be found from the network with highest probability P, by adding or removing only a small number of links. This suggests that the stronger algorithms for this problem will start at P, to reduce the amount of work required.

I will also adapt an algorithm that is frequently used for SAT and MAX-SAT problems call GSAT [Selman et al., 1992], and was adapted for Weighted MAX-SAT in Yuejun Jiang [1995]. This algorithm works by "flipping" links, that is adding or removing links, like my methodology, and can be viewed as a kind of greedy search. I use two parameters to decide which link to flip firstly the *consistency gain* which is defined as

$$\textit{consist-gain}_N(r(e_1, e_2)) = |\textit{inconsist}_T(N)| - |\textit{inconsist}_T(N \ominus \{r(e_1, e_2)\}|$$

That is the consistency gain is equal to the change in the number of inconsistent rules gained by flipping link $r(e_1, e_2)$. I can now define the adapted GSAT algorithm

4.3.7.9: Algorithm
Input: An LSLC problem (E, R, T, P, c)
Output: A network N which is consistent with T.

1. Initially $N \leftarrow P$.

2. While $\textit{inconsist}_T(N) \neq \emptyset$.

   (a) Choose the link $r(e_1, e_2)$ which maximises

$$c(r(e_1, e_2))\textit{consist} - \textit{gain}_N(r(e_1, e_2))$$

   If multiple such links exist choose one *at random*.

   (b) Let $N \leftarrow N \ominus \{r(e_1, e_2)\}$

This algorithm is not guaranteed to terminate, for example it cannot possibly terminate if T has no consistent networks, however I have found from my experiments that it rarely fails to terminate in a reasonable time. In the original paper [Selman et al., 1992], the authors suggest that the start point of GSAT should be a random network, for this problem I choose to use P, as the intuition is that the optimal network should be within a few flips of P, however I accept that adding some random permutations to P and running multiple times may be an acceptable compromise. I will also present one simple method that can improve the efficiency of the algorithm and that is to store two variables the *gain* and *loss* of the link, these represent the set of new links that would be gained or lost by flipping one link, i.e., formally

$$gain(r(e_1, e_2)) = inconsist_T(N) - inconsist_T(N \ominus \{r(e_1, e_2)\})$$
$$loss(r(e_1, e_2)) = inconsist_T(N \ominus \{r(e_1, e_2)\}) - inconsist_T(N)$$

It should clear that

$$consist\text{-}gain(r(e_1, e_2)) = |gain(r(e_1, e_2))| - |loss(r(e_1, e_2))|$$

If we choose to flip a link $r(e_1, e_2)$, then it is possible to update gain and loss by

$$gain(r(e_1, e_2)) \leftarrow loss(r(e_1, e_2))$$
$$loss(r(e_1, e_2)) \leftarrow gain(r(e_1, e_2))$$

And more importantly for each link $r'(e'_1, e'_2) \neq r(e_1, e_2)$ their loss and gain are updated by

$$gain(r'(e'_1, e'_2)) \leftarrow (gain(r'(e'_1, e'_2)) - gain(r(e_1, e_2)))$$
$$loss(r'(e'_1, e'_2)) \leftarrow loss(r'(e'_1, e'_2) - loss(r(e_1, e_2))$$

The first part of both formulae should be clear if $t \in T$ is satisfied by flipping $r(e_1, e_2)$, then it cannot now be satisfied by flipping $r'(e'_1, e'_2)$, as it is already satisfied. This methodology is effective but can cause problems consider a rule

$$\rightarrow r(e_1, e_2), r'(e'_1, e'_2)$$

When both $r(e_1, e_2) \in N$ and $r'(e'_1, e'_2) \in N$, this is clearly satisfied and is not in the loss of either link as we would need to flip both links to make it unsatisfied. This means when updating the gains and losses, this link is never included as it is not in the loss or gain of any link, however once we have flipped $r(e_1, e_2)$, $r'(e'_1, e'_2)$ should have this rule in its loss. For this reason the best way to implement this algorithm is to use the above update formula, and after a fixed number of iterations, rebuild the gain and loss set. This gives a good balance between having to search frequently for all inconsistencies (in general a slow process), and is an effective solution.

4.3.8 Improvements to simple logic

I will now propose some simple extensions to improve the usability of the language, which should make it possible to represent most useful forms of first order logic statements.

4.3.8.1: Definition I will now say that the set of relations R is in fact split into two sets $R = R^1 \cup R^2$ and that a network consists of a set of links of the form $r(e_i, e_j)$ where $r \in R^1$ and links of the form $r(e_i)$ where $r \in R^2$.

The purpose of allowing *unary* relation statements as well as *binary* statements is to allow for statements about sets to be made, for example the first order logic statement $e \in S$ can be restated as $r_S(e)$. I shall also reformulate the language for describing rules as such.

4.3.8.2: Definition The extended simple logic rule is of the form

`rule := expr` $\rightarrow$ `expr`
`expr := (term (,term)*)?`
`term :=` r `(var,var)` *where* $r \in R^1$ | r `(var)` *where* $r \in R^2$
`var :=` $x_1, \ldots$ | e_i *where* $e_i \in E$

This extends the logic to be able to make statements about the unary operators and makes it possible to state facts about particular elements from E. This ability is useful as it allows for handling things such as in a hierarchical logic, that all terms are derived from some root term (for example "object"), a rule of the following from could be used

r(``object'',x_1)

I shall also allow for certain links to be *immutable*, meaning that for the purposes of the algorithms presented, it is not possible to change an immutable link. This is useful as this allows us to expand on existing ontologies by including the known links as immutable links. In effect this means that for those methods based on choosing links such as algorithms 4.3.2.3 and 4.3.7.9 I simply modify the algorithms to not choose an immutable link and for those based on matrices such as algorithms 4.3.4.4 and 4.3.5.9 I do not allow the inclusion of a column for an immutable link in the matrix.

I have also always made the explicit assumption that all elements in E are different, that is that $e_i \neq e_j$ if $i \neq j$, however for representing some of the more complex forms seen in first order logic it would be advantageous to be able to relax this.

4.3.8.3: Definition Redefine E as a set $E = E' \cup K$ (and $E' \cap K = \emptyset$), and I shall say that R has a relationship $e(.,.)$ such that the following axioms hold

1. $\forall e_i \in E : e(e_i, e_i)$
2. $\forall e_i, e_j \in E : e(e_i, e_j) \rightarrow e(e_j, e_i)$
3. $\forall e_i, e_j, e_k \in E : e(e_i, e_j) \wedge e(e_j, e_k) \rightarrow e(e_i, e_k)$
4. $\forall e_i, e_j, e_k \in E, r \in R : r(e_i, e_j) \wedge e(e_j, e_k) \rightarrow r(e_i, e_k)$

5. $\forall e_i, e_j, e_k \in E, r \in R : r(e_i, e_j) \wedge e(e_j, e_k) \rightarrow r(e_k, e_j)$
6. $\forall k_i \in K, \exists e_i \in E' : e(k_i, e_i)$
7. $\forall k_i \in K, e_i, e_j \in E' : e(k_i, e_i) \wedge e(k_i, e_j) \rightarrow i = j$
8. $\forall e_i, e_j \in E : e(e_i, e_j) \rightarrow i = j$

$e(\cdot, \cdot)$ is a relationship in the network that is used to represent the idea of equality. This makes it possible to introduce a number of *constants* K to the system, however there will be some problems reformulating the logic to handle these new forms. The first five axioms are standard forms and can just be included as simple logic rules, however the sixth, seventh and eighth are harder.

4.3.8.4: Definition The *constant axioms* are such that for each element $k \in K$, if $E' = \{e_1, \ldots e_n\}$, there is a ground instance

$$\rightarrow e(k, e_1), \ldots e(k, e_n)$$

And for each pair $e_i, e_j \in E'$, $i \neq j$ there are two ground instance

$$e(k, e_i), e(k, e_j) \rightarrow$$
$$e(e_i, e_j) \rightarrow$$

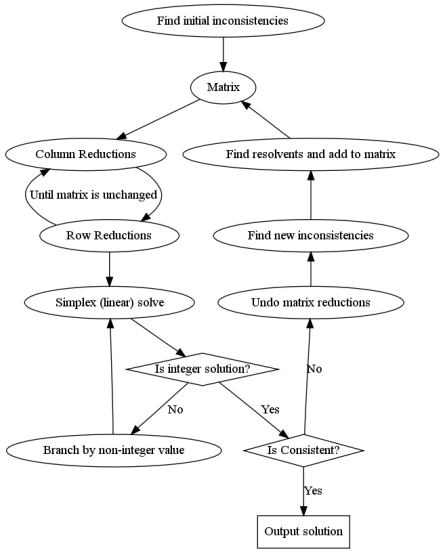

Figure 4.3: The work-flow of algorithm 4.3.5.9

Summary

For dealing with different kinds of network structures, I propose the use of axioms to describe them, as it is more flexible. In this section I present a form of logic, which I call *simple logic*, which has the property that its ground instances can be viewed as propositional logic clauses. As before I start by presenting a simple greedy approach as a starting point, and show that it can create a consistent solution for a subset of simple logic problems. I approach the problem of finding the optimal solution, as that of starting with the network that is most likely from the output of the pattern-based extraction system, and enumerating the inconsistencies of this solution. My method then attempts to correct the solution by adding or removing links. I show that this can be presented as an integer programming problem, however it is not easy to include inconsistencies that are introduced by changes of the

matrix, which means the algorithm cannot naturally proceed after the first iteration. This can be solved by the introduction of negative values or through the use of resolution and I present both methodologies. I find the resolution based method to be preferable as it allows for certain matrix reduction principles to be applied, which can often reduce the matrix to a trivially solvable form. The work flow of my method is presented in figure 4.3.

I then present an alternative approach, which converts the problem into a weighted MAX-SAT problem, which is a widely studied problem. This conversion makes the problem difficult for most standard solution methods of weighted MAX-SAT but I do present an adaption of the GSAT method, which is practical for most situations. Finally I present a few extensions to the simple logic algorithm, which allow for a wider range of axioms to be used without fundamentally changing how the solution methods I present operate.

4.4 First-order logic

4.4.1 Conjunctive Skolem Normal Form

The problem of representing first-order logic is hard, however most forms can be represented through the use of *skolemization*. This is a process where existence statements are converted into a set of functions and constants.

4.4.1: Definition *Conjunctive Skolem Normal Form* is a logical form as follows

```
statement := clause (∧ clause)*
clause := term (∨ term)*
term := predicate ‘‘(’’ parameter (‘‘,’’ parameter)* ‘‘)’’
parameter := var | constant | function ‘‘(’’ parameter (‘‘,’’
parameter)* ‘‘)’’
var := x_1,...
constant := k_1,...
function := f_1,...
```

4.4.2: Definition Let F^n be the set of function with n parameters and F^0 be the set of constants in a CSNF. The CSNF is consistent with a network $N \in N(E, R)$ if there exists a *functional* mapping from the set F^0 to E and a functional mapping from $F^0 \times E$ to E etc. And these mappings can be used to translate the clauses of the CSNF to a set of simple logic ground instances which are consistent with N.

4.4.3: Theorem Any first order logic can be converted into Conjunctive Skolem Normal Form.

For proof of this theorem and discussion of algorithms suitable for converting first order logic forms to CSNF please see Nomengart et al. [1998]

The first task for solving more general first order logic problems is to define the constant set K from definition 4.3.8.3, this is performed as follows

4.4.4: Definition If there is some CSNF, L, and we have the set of function statements F and the set of constant statements C and there is a set of elements E then define the constant set K by

- $C - E \subset K$
- For each $f(\ldots) \in F$, if $x_1, \ldots x_m$ are the variables occurring in $f(\ldots)$ then there is a constant $k_f|_{x_1=e_i,\ldots}$ for each substitution of $x_1, \ldots x_m$ with variable from E.
- For each $f(\ldots) \in F$, if g is a function symbol in $f(\ldots)$ similarly derive a constant symbol by replacement of any variables.

4.4.5: Example Suppose $E = \{e_1, e_2\}$ and we have CSNF equivalent to $\forall x \exists y : r(x, y)$

$$r(x_1, f(x_2))$$

Then we have that $K = \{k_{f(e_1)}, k_{f(e_2)}\}$

If there are complex functional symbols it is sometimes necessary to introduce some extra equivalences to ensure these functions work correctly. For example assume we have $F = \{f(x_1), g(x_1), f(g(x_1))\}$, we would need to introduce an axiom to ensure that we don't have a situation such as $g(e_1) = e_2$ and $f(e_2) = e_3$ and $f(g(e_1)) \neq e_3$.

4.4.6: Definition The *functional axioms* of a set of functions are defined as such that if there exists a parameter p in the CSNF and there is a function $f(\ldots)$ such that somewhere in this function p occurs (denote this $g(\ldots p \ldots)$, although p may occur nested anywhere in this statement) then it is necessary to introduce a rule of the form

$$e(x_1, f(p)), e(x_2, g(\ldots f(p) \ldots) \rightarrow e(x_2, g(\ldots x_1 \ldots))$$

4.4.7: Theorem If there is a CSNF problem, T_{CSNF} then by taking K as the constant set as defined in definition 4.4.4, then applying the constant axioms of 4.3.8.4, the functional axioms of 4.4.6, and all substitutions to the remaining rules, it is possible to gain a set of ground instances T_{simple} such that for any network $N \in N(E, R)$, N is consistent with T_{simple} according to definition 4.3.1.5 if and only if N is consistent with T_{CSNF} according to definition 4.4.2
Proof: We show that the mappings m_n from $F^n \times E \times \ldots \times E$ are such that for each $f(\ldots) \in F^N$ we have for each $e_1, \ldots e_n \in E$ we have that if $m_n(f(e_1, \ldots e_n)) = e'$ then we can create a network such that the statement

$e(f(e_1, \ldots e_n), e')$ is in the network (hence satisfying constant axiom 6 of definition 4.3.8.4) and we say there is no e'', such that $e(f(e_1, \ldots e_n), e'')$. Then we create a network such that this holds for all mappings. If we have a CSNF instance then by repeated applications of function axioms we can first remove all nested function statements. After that by applications of constant axioms 4 and 5 of definition 4.3.8.4 we can create a ground instance in simple logic which is true if the CSNF statement is true and the axioms are true in the network. It is clear that the function axioms must be true by the CSNF mapping and that the ground axioms can easily be satisfied, for constant axioms 1-3 of definition 4.3.8.4 simply observe that for each value $e' \in E$, we have a set defined by $\{k_{f(p)}|m(f(p)) = e'\}$ and we have an equivalence relationship between elements of the set $\{k_{f(p)}|m(f(p)) = e'\} \cup \{e'\}$. Axioms 4 and 5 of definition 4.3.8.4 can be trivially made consistent by adding their conclusions to the network.
If we assume that we have a network that is consistent with the T_{simple} and the corresponding axioms we can simply construct the maps by finding the value e' for each $f(e_1, \ldots e_n)$ such that $e(f(e_1, \ldots e_n), e')$ is in the network. This value must exist by axiom 6 and be unique by axiom 7 of definition 4.3.8.4 so we can create a mapping which leads to the same consistent network.□

This means that it is possible to simply solve problems in first order logic by converting them to CSNF and then creating the appropriate set of axioms and solving this as a simple logic problem. It is important to note here that by doing this it modifies the semantics of first order logic, essentially introducing a *closed world assumption* in that every skolem function's ground instance must be mappable to a single element from E. This closed world assumption is explicitly stated in axioms 6, 7 and 8 of definition 4.3.8.3. This means that this definition avoids the problems of *undecidability* normally found in first order logic, however this means that some first order logic problems may be considered "unsatisfiable" under these semantics, when they would be "satisfiable" under normal first-order semantics. This is easily verifiable by the fact that my semantics mean that as it is possible to establish the following two identities

$$\forall x_i \Leftrightarrow \bigwedge_{e_i \in E}$$
$$\exists x_i \Leftrightarrow \bigvee_{e_i \in E}$$

It is clear that under this transformation it is possible to convert any first-order logic problem not involving functions into a propositional logic form and then determine if it has any solutions. It should also be noted that applying these two identities gives another way to solve first order logic problems, however the effect of converting even simple first order logic statements into huge number of simple logic ground instances. For example, the first order logic statement

$$\exists x \forall y : r(x, y)$$

Converts to

$$\bigvee_{e_i \in E} \bigwedge_{e_j \in E} r(e_i, e_j)$$

Which when translated into ground statements gives $|E|^{|E|}$ ground instances of the form

$$\rightarrow r(e_1, e'_1), \ldots r(e_{|E|}, e'_{|E|}) \text{ where } e'_i \in E$$

For each possible choices for each e'_i.

There are still several problems with simply applying my previous algorithms to the problem, and these are caused by introducing a new relationship e into the system. In contrast to the relationships before when there was a known probability value for each relation, $p(r(e_i, e_j))$ for e there is no such value. It is clear however that the cost of any particular assignment to e should not effect the overall solution cost so I assume that

$$P(e(e_i, e_j)) = 0.5$$

Hence

$$c(e(e_i, e_j)) = log(P(e(e_i, e_j))) - log(1 - P(e(e_i, e_j))) = 0$$

This is good but it makes it impossible to choose which e relations to include in the initial network P. To solve this I assign a small probability value to each e relation, which is small enough that it will not affect the overall solution. This value is also useful in the solution method as if the values are chosen prefer those links which are more likely to be true, this will guide the solver to choosing better choices first.

4.4.8: Definition Define ϵ as value such that if ϵ' is a value that is the smallest nonzero value of

$$|r(e_i, e_j) - r(e'_i, e'_j)|$$

Where $r \in R$ and $e_i, e_j, e'_i, e'_j \in E$ and it is not the case that $e_i = e'_i$ and $e_j = e'_j$. Then ϵ is take to be a value such that

$$\epsilon \leq \frac{\epsilon'}{|K|}$$

In practise it is rarely necessary to calculate this value, and instead just guess a suitable value. For example on a machine using double precision floating points a value of $2^{-1000} \simeq 10^{-300}$, is usually suitable.

4.4.9: Definition If there is a set of CSNF propositions T and a network N, and by definition 4.4.4 it is known that this will cause there to be a set of constants, K. Then define the *gain* of a mapping from $k \in K$ to $e \in E$ as

$$\begin{aligned} gain(e, k) = \quad & |\{t \in T | r(k, e') \in premise(t), r(e, e') \notin N)\} \\ & \cup \{t \in T | r(e', k) \in premise(t), r(e', e) \notin N\} \\ & \cup \{t \in T | r(k, e') \in conclusion(t), r(e, e') \in N\} \\ & \cup \{t \in T | r(e', k) \in conclusion(t), r(e', e) \in N\}| \end{aligned}$$

Similarly define the *loss* as

$$\begin{aligned} loss(e,k) = \quad & |\{t \in T | r(k,e') \in premise(t), r(e,e') \in N)\} \\ & \cup \{t \in T | r(e',k) \in premise(t), r(e',e) \in N\} \\ & \cup \{t \in T | r(k,e') \in conclusion(t), r(e,e') \notin N\} \\ & \cup \{t \in T | r(e',k) \in conclusion(t), r(e',e) \notin N\}| \end{aligned}$$

And then define the *utility* of the mapping as

$$utility(e,k) = \frac{max_{e' \in E}(gain(e',k) - loss(e',k)) - (gain(e,k) - loss(e,k))}{max_{e' \in E}(gain(e',k) - loss(e',k)) - min_{e' \in E}(gain(e',k) - loss(e',k))}$$

Stated informally the *gain* is "the number of clauses that could be satisfied by replacing k with e", the *loss* is "the number of clauses that may become inconsistent if k is replaced with e", and the *utility* normalises this value in the range $0 \leq utility(e,k) \leq 1$. This means it is possible to set the cost for the relationship e as follows

$$c(e(e,k)) = \epsilon \times utility(e,k)$$

This value is now guaranteed to be less than the smallest possible difference between two otherwise equal solutions, although it may end up differentiating solutions with equal value. It is also possible to define the starting network P as the network given by definition 4.2.1, but in addition it holds that for each $k \in K$, $e(e',k) \in P$ and $e(k,e') \in P$ if $e' = argmin_{e \in E} utility(e,k)$.

Ground instances such as those generated by constant axiom 6 (of definition 4.4.4) can be very long, however fortunately it is possible to easily deal with this problem by using the column reduction principle (4.3.6.5). I reapply this result so that I can add columns to the matrix in a "lazy" way.

4.4.10: Lazy column introduction principle For ground instances derived from axiom 5, at each stage add values to columns, which

- Correspond to ground instances inconsistent with the network, i.e, columns which are already in the matrix
- Have the lowest value for $c(r(e_i,e_j))$ among values for $r(e_i,e_j)$ which are not already in the matrix.

I now need to mention two more things: for relations $r \in R$ which aren't e, I simply set the cost of all values $r(e',k)$, where $e \in E'$ and $k \in K$ as

$$c(r(e',k)) = 0$$

The reason for doing this is to make it easier for the system to follow deductions made by constant axioms 4 and 5 (of definition 4.3.8.4). Secondly I shall say

that the network from definition 4.4.9 also includes all statements of the form $e(e', e')$ where $e' \in E' \cup K$. We shall now follow through a simple example

4.4.11: Example Suppose we have a network N on the set of elements $E' = \{e_1, e_2, e_3\}$ and relation $R = \{r\}$ given by

$$N = \{r(e_1, e_2), r(e_1, e_3)\}$$

And we wish to find a network, which is consistent with the following first order logic statements

- $\forall x : \neg r(x, x)$
- $\forall x, y, z : r(x, y) \wedge r(y, z) \rightarrow r(x, z)$
- $\exists x \forall y r(x, y) \vee x = y$

It follows then that in CSNF this is given as

1. $r(x_1, x_1) \rightarrow$
2. $r(x_1, x_2), r(x_2, x_3) \rightarrow r(x_1, x_3)$
3. $\rightarrow r(k, x_1), e(k, x_1)$

And we have that $K = \{k\}$. In addition we introduce the following constant axioms

1. $\rightarrow e(x_1, x_1)$
2. $e(x_1, x_2) \rightarrow e(x_2, x_1)$
3. $e(x_1, x_2), e(x_2, x_3) \rightarrow e(x_1, x_3)$
4. $r(x_1, x_2), e(x_2, x_3) \rightarrow e(x_1, x_3)$
5. $r(x_1, x_2), e(x_1, x_3) \rightarrow e(x_3, x_2)$
6. $\rightarrow e(k, e_1), e(k, e_2), e(k, e_3)$
7. $e(k, e_1), e(k, e_2) \rightarrow$
8. $e(k, e_1), e(k, e_3) \rightarrow$
9. $e(k, e_2), e(k, e_3) \rightarrow$
10. $e(e_1, e_2) \rightarrow$
11. $e(e_1, e_3) \rightarrow$
12. $e(e_2, e_3) \rightarrow$

(Note I have omitted unnecessary variants of axioms 7-12)

I now calculate the gain and loss of each assignment to k. For each value we have at least gain=1 from axiom 6 and loss=2 from axioms 7 (of definition 4.3.8.4) in addition we have the ground instances of rule 3 which are

- $\rightarrow r(k, e_1), e(k, e_1)$
- $\rightarrow r(k, e_2), e(k, e_2)$
- $\rightarrow r(k, e_3), e(k, e_3)$

Hence we have that

	gain	loss	utility
k, e_1	3	3	0
k, e_2	1	3	1
k, e_3	1	3	1

Hence it follows that our initial network N' is given as

$$N' = \{r(e_1, e_2), r(e_1, e_3), e(k, e_1), e(k, e_2), e(e_1, e_1), e(e_2, e_2), e(e_3, e_3), e(k, k)\}$$

And this leads to the following ground inconsistencies with N'

- $\rightarrow r(k, e_2), e(k, e_2)$
- $\rightarrow r(k, e_3), e(k, e_3)$
- $r(e_1, e_2), e(k, e_1) \rightarrow r(k, e_2)$
- $r(e_1, e_3), e(k, e_1) \rightarrow r(k, e_3)$
- $r(e_1, e_2), e(e_1, k) \rightarrow r(k, e_2)$
- $r(e_1, e_3), e(e_1, k) \rightarrow r(k, e_3)$

And that this can be solved for zero cost by adding $\{r(k, e_2), r(k, e_3)\}$ to N'.

4.4.12: Algorithm to convert first-order logic problem to simple logic
Input: A network $N \in N(E', R)$ and a set of first order logic statements T_{first}, and cost function on $\mathcal{N}_{(E',R)}$, $c(\ldots)$
Output: A network $N' \in N(E' \cup K, R \cup \{e\})$, with a corresponding cost function $c(\ldots)$ and a set of ground instances T_{simple}

1. Skolemize:
   (a) Replace all instances of $\exists$ with appropriate skolem functions.
   (b) Rearrange the formulae into CSNF, and add all ground instances to T_{simple}.
2. Find the set K which corresponds to all ground instances of functions.

3. Introduce all ground instances of constant axioms to T_{simple} as in definition 4.3.8.3. Note the axioms 6 and 7 must be introduced as ground instances as per definition 4.3.8.4.

4. If necessary, introduce functional axioms as in definition 4.4.6 to T_{simple}.

5. For each $k \in K$ calculate the utility of k with each element $e' \in E'$ as per definition 4.4.9.

6. Form N' by adding all links $e(k, e')$ and $e(e', k)$, which have maximum utility.

7. Add all links of the from $e(e', e')$, where $e' \in E' \cup K$ to N'.

8. Calculate and add the new values to the cost function for links $e(e', k)$ where $e' \in E'$ and $k \in K$ by $c(e(e', k)) = \epsilon \times utility(e', k)$ and set the cost of all other links not defined by the original cost function to 0.

4.4.2 Conversion from OWL

I shall now present the methodology that can convert an ontology written in OWL into a sequence of statements that can be used with my system to develop or expand the ontology. I shall start with *classes*, which are defined in OWL[5] as

```
<owl:Class rdf:ID="classID"/>
```

For each such statement I create a unary relationship $classID(\cdot)$. It is then possible to make certain statements about classes, firstly the `subClassOf` statement is written as

```
<owl:Class rdf:ID="subClass">
<rdfs:subClassOf rdf:resource="superClass"/ >
</owl:Class>
```

These statements convert to a rule

$$subClass(x) \rightarrow superClass(x)$$

Similarly the `disjointWith` statement such as

```
<owl:Class rdf:ID="class">
<owl:disjointWith rdf:resource="otherClass"/>
</owl:Class>
```

Translates to

$$class(x), otherClass(x) \rightarrow$$

I also translate the `equivalentClass` statement similarly

[5]Using the XML notation for OWL

```
<owl:Class rdf:ID="class">
<owl:equivalentClass rdf:resource="otherClass"/>
</owl:Class>
```

Becomes the following two rules

$$class(x) \rightarrow otherClass(x)$$
$$otherClass(x) \rightarrow otherClass(x)$$

I then define each property as a relationship taking two parameters, and I can then state logical rules to define their domain and range for example

```
<owl:ObjectProperty rdf:ID="prop">
<rdfs:domain rdf:resource="domainClass">
<rdfs:range rdf:resource="rangeClass">
</owl:ObjectProperty>
```

Gives the following rules

$$prop(x, y) \rightarrow domainClass(x)$$
$$prop(x, y) \rightarrow rangeClass(y)$$

I also define the `subProperty` and `equivalentProperty` as I did for the same properties of classes and the inverse property as in

```
<owl:ObjectProperty rdf:ID="prop">
<rdfs:domain rdf:resource="domainClass">
<rdfs:range rdf:resource="rangeClass">
<owl:inverseOf rdf:resource="inverseProp">
</owl:ObjectProperty>
```

Is defined as

$$prop(x, y) \rightarrow inverseProp(y, x)$$
$$inverseProp(x, y) \rightarrow prop(y, x)$$

There are also a number of special properties for relationships for example

```
<owl:ObjectProperty rdf:ID="prop">
<rdfs:domain rdf:resource="domainClass">
<rdfs:range rdf:resource="rangeClass">
<rdf:type rdf:resource="&owl;TransitiveProperty">
</owl:ObjectProperty>
```

I translate these as follows

- `TransitiveProperty`: $prop(x_1, x_2), prop(x_2, x_3) \rightarrow prop(x_1, x_3)$
- `SymmetricProperty`: $prop(x_1, x_2) \rightarrow prop(x_2, x_1)$
- `FunctionalProperty`: $\rightarrow prop(x_1, f(x_1))$
- `InverseFunctionalProperty`: $\rightarrow prop(f(x_1), x_1)$

- `AsymmeticProperty`[6]: $prop(x_1, x_2), prop(x_2, x_1) \rightarrow e(x_1, x_2)$
- `ReflexiveProperty`[6]: $\rightarrow prop(x_1, x_1)$

If there is a class that is defined by `Restriction` it is necessary to convert this as well, for example OWL often uses the following formulation

```
 <owl:Class rdf:ID="class">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#prop"/>
<owl:hasValue rdf:resource="#obj"/>
</owl:Restriction>
</owl:subClassOf>
</owl:Class>
```

This can easily be translated to a statement such as

$$prop(x, \text{“obj”}) \rightarrow class(x)$$

Similarly it is easy to create rules for the similar restrictions `allValuesFrom` and `someValuesFrom`. Another two restrictions of particular interest are `minCardinality` and `maxCardinality`. These are easily representable but can often lead to long clauses, in normal first order logic the `minCardinality` is n restriction can be stated as

$$\forall y \exists x_1, \ldots x_n : prop(y, x_1) \wedge \ldots \wedge prop(y, x_n) \wedge x_1 \neq x_2 \wedge \ldots \wedge x_{n-1} \neq x_n$$

This gives a large number of clauses of the form

$$\begin{gathered} \rightarrow prop(y, f_1(y)) \\ \vdots \\ \rightarrow prop(y, f_n(y)) \\ e(f_1(y), f_2(y)) \rightarrow \\ \vdots \\ e(f_{n-1}(y), f_n(y)) \rightarrow \end{gathered}$$

Similarly the `maxCardinality` restriction translates to first-order logic as

$$\forall y \nexists x_1, \ldots x_{n+1} : prop(y, x_1) \wedge \ldots \wedge prop(y, x_{n+1}) \wedge x_1 \neq x_2 \wedge \ldots \wedge x_{n+1} \neq x_n$$

Which instead translates into CSNF as a single long clause of the form

$$prop(y, f_1(y)), \ldots prop(y, f_{n+1}(y)) \rightarrow e(f_1(y), f_2(y)), \ldots e(f_n(y), f_{n+1}(y))$$

Finally I define the instances of the ontology as the basis for the set E, and I say that for instances the OWL elements `differentFrom` and `AllDifferent` give simple statements of the form

$$e(x_1, x_2) \rightarrow$$

As such it should be clear that I can efficiently covert an OWL ontology into a set of CSNF formulae that can be used with my ontology discovery system.

[6]OWL 2 Only

Summary

I extend simple logic into a particular form of first-order logic, which is called *Conjunctive Skolem Normal Form*. I then make a *closed world assumption*, that all functions and constants map uniquely to an element of E, and define a specific relation e, which represents equality in my system. Then by introducing a number of axioms I show how the CSNF rules can be converted into a set of simple logic rules. This means that the system can approach a first-order problem under these assumptions in the same way as a simple logic problem, so if for example the initial set of restrictions were inconsistent then a contradiction would occur as in example 4.3.5.14. Finally I show how it is possible to convert the statements of OWL into a set of CSNF rules.

Chapter 5

Results

5.1 Pattern system and synonymy

I will start by presenting the work published in McCrae and Collier [2008][1]. In that paper I showed one of the first principles that motivate the need for the need for good automatic ontology construction, and that is that a lot of extant resources do not cover the kinds of terms seen in real data very well. In order to show this I first obtained the 150 top ranked PubMed abstracts for the keywords "infectious disease". I then went through these documents, extracted all medical terminology and grouped these according to which sets of terms were synonymous. This is not totally straight-forward as there are many aspects with terms being close or nearly exactly synonymous but not quite. I found that the following issues were common among the terms I saw:

- **Modifiers:** This is the inclusion of an adjective modifier in a term. For example the term "acute headache" is not the same as "headache" although it is nearly synonymous. I decided that it was better in general to view these terms as not synonymous as their meaning is not the same. It is important not to confuse this with modifiers that are intrinsic to the meaning of a term such as "mental retardation".
- **Granularity:** This covers cases where terms are nearly always used synonymously but have slight differences in their meaning. For example the term "HIV-1" is the most common strain of "HIV", and as such they generally refer to the same thing, as "HIV-2" is less easily transmitted and mostly confined to a small area of West Africa. Again I decided not to count such terms as synonymous.
- **Property:** This means that two terms refer to the same thing but with a slightly different property, for example "dengue shock syndrome" is a late stage development of "dengue fever", as these are the same disease but differ in severity, I decided that these terms are not synonymous.

[1] http://www.biomedcentral.com/1471-2105/9/159

	Precision	Recall	F-Measure	Coverage
Wikipedia (redirect)	46.4%	18.8%	26.8%	54.1%
Wikipedia (search)	40.1%	24.6%	30.9%	56.1%
WordNet	100%	6.9%	13.0%	38.0%
Medline Encyc.	66.7%	4.0%	7.5%	28.1%
MeSH	61.6%	15.8%	25.2%	55.1%
UMLS	94.0%	46.5%	62.3%	79.7%

Figure 5.1: Encyclopedia Results

In total from the 150 documents I found 301 terms, which I grouped into 221 synonym sets, representing 101 synonym links. A second annotator also performed the same task and I found that there was an inter-annotator agreement (Cohen's agreement) of $\kappa = 68.6\%$. I then looked at how well these terms were represented by various resources, firstly WordNet, then two specialised domain thesauri MeSH and UMLS. I also used two general knowledge sources the layman-orientated Medline encyclopedia and Wikipedia, which were not designed as thesauri so I needed some manipulation to extract this information

- **Wikipedia (redirect):** Wikipedia includes redirect tables that allow the system to redirect users to a single page based on several queries. For example searching Wikipedia for "WNV" and "West Nile Virus" both direct to the same page.[2]

- **Wikipedia (search):** This uses the top result based on Wikipedia's search engine, I assume two terms are synonymous if they have the same top result.

- **Medline encyclopedia:** The articles in this encyclopedia have an "Alternate names" field I used as a basis for this method.

I then took the term pairs and used these resources to see how well it matched the manual extraction.

In figure 5.1 I show the results for these information sources based on their precision, recall and F-Measure and coverage that is the number of terms for which they had a reference for.

Then I used the BioCaster ontology [Collier et al., 2008] to extract a set of seed terms, which is an ontology which specialises in disease outbreak and so should be reasonably close to the extracted data. The BioCaster ontology contained 450 terms, grouped into 244 synsets, giving 477 synonym pairs, of which 35 terms and 16 pairs were also in my manually extracted set. I then used the total set of 751 terms to extract a corpus by querying PubMed for each term and taking the top 250 abstracts for each term. This gave a corpus of 83,492 documents constituting 1,506,402 sentences of which 46,216 contained a

[2]Disclaimer: Wikipedia changes constantly, the example was checked and works on 24/4/09

term pair from one of the sets. I first examined the effect of just examining the co-occurrences of these terms, I found that out of all possible pairs of terms in the manually-extracted set that only 1.7% of occurrences were of synonymous terms. This demonstrates the need for patterns to identify those terms which are actually synonymous, I also found that 63.4% of the manually extracted synonym pairs were in the corpus. I then applied the methodology of section 3.1 to extract patterns from this set using the set of synonym pairs from the BioCaster ontology as my seed pairs. Table 5.1 presents some of the highest scoring patterns for synonyms with some examples of the extractions given. I used several statistical classification methods to examine the ability of the statistical classifier to output the results based on the manually extracted set, these are presented in table 5.5, these classifiers are SVM and Logistic Regression as I have already discussed, In all cases I used the WEKA[3] implementations to perform the classification.

I reviewed the results based on their performance and it was clear that the support vector machine produced by far the strongest result and had a very high precision, which meant that it was far more likely that the results it was producing were correct. Also the support vector machine approach can be adapted to use logistic regression, which means that a better estimate of the output probability can be obtained. This is highly advantageous as the logical consistency system works requires estimates of the probabilities to work well.

I then used algorithm 4.2.1.16 (Set Branch and Bound) to find the optimal way to group these into synonymy sets. Using the results from the SVM classifier I found that the precision was slightly decreased from 74.1% to 73.2%, however the recall improved from 22.8% to 29.7%, increasing the overall F-Measure to 42.3% from 35.7%. In figure 5.4, I plotted the result of stopping the algorithm at iterations before it found the optimal solution, I calculated the theoretical cost function as defined in section 4.1.2 and compared it with the actual F-Measure. This result shows that as the solution method finds a "theoretically" better solution this corresponds to better actual results.

In figure 5.3, I analysed the errors of the extractions splitting them into the following classes

- **Modified:** The term pair differed by inclusion of a modifier.
- **Variant:** The terms were closely related, normally this meant they were organism of the species or disease caused by the same agent.
- **Method of Transmission/Point of Infection:** These terms differed by the method of transmission or point of infection.
- **Agent/Disease:** The terms were an agent and the disease caused by that agent.
- **Hypernym:** The terms were hypernyms (and not of any other class mentioned).

[3]WEKA is available at http://www.cs.waikato.ac.nz/ml/weka/

["CA-MRSA", "community-acquired MRSA", "methicillin-resistant Staphylococcus aureus", "MRSA", "methicillin-resistant Staphylococcus"] – *MoT/PoI errors*
["Litopenaeus vannamei", "shrimp"] – *Hypernym relations*
["Yersinia pestis", "Plague"] – *Disease/Agent errors*
["dengue shock syndrome", "DHF", "dengue hemorrhagic fever", "dengue", "dengue fever"] – *Variant terms*
["rubella", "mumps", "measles"] – *Other relation (there is a widely-used triple vaccine for these diseases)*

Figure 5.2: Example Output from pattern-extraction system

	Correct	Modified	Variant
Binary	40.8%	1.4%	7.5%
Synset	39.5%	0%	12.1%
MoT/PoI	Agent/Disease	Hypernym	Error
7.5%	12.9%	2.0%	29.3%
9.9%	17.6%	3.3%	17.6%

Figure 5.3: Analysis of Errors (from McCrae and Collier [2008])

- **Error:** No semantic relation between the terms.

These results show that compared to the result from just the statistical classifier ("Binary"), the logical consistency algorithm managed to reduce the number of errors in all classes, especially for the case when the terms had no semantic relation. Finally, figure 5.2 shows a sample of the system's output, showing both correct matches and some erroneous ones labelled with the categories above.

5.2 Relation Extraction

I wish to test the join-set methodology I developed in section 3.2, I do this by attempting to extract two relations synonymy and hypernymy. I also choose to focus on a particular sub-domain of disease. I start by collecting a set of terms, for this I use all terms, which are hyponyms of "disease" in WordNet. This gives a set of 1152 terms, I then use WordNet to create a list of synonym and hypernym pairs between these terms. In total there are 7150 hypernym term pairs 3814 synonym term pairs. I then collected a corpus by downloading a set of document from PubMed which include the terms, by the same method as before. I then obtained a corpus of 22895 documents constituting approximately 15 million words, and identified all terms using the method described in section 3.2.1. In figures 5.6 and 5.7 I show the occurrences of each term pair in the corpus, and find that the overwhelming majority of these term pairs do not occur in the corpus, in fact only 314 synonym pairs and 299 hypernym pairs are found in the corpus. We also see that only 63 hypernym term pairs and

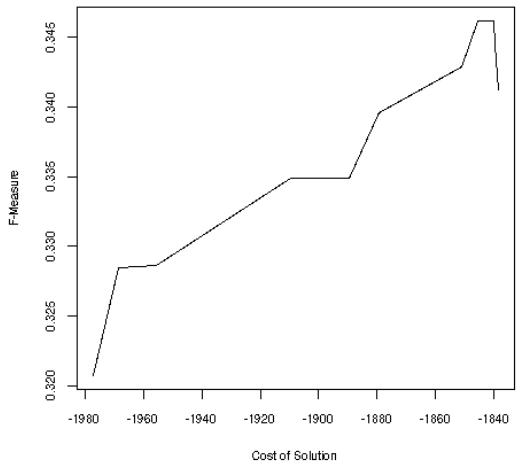

Figure 5.4: F-Measure versus theoretical cost (from McCrae and Collier [2008])

	Precision	Recall	F-Measure
Baseline	1.7%	63.4%	3.3%
Naïve Bayes	30.6%	37.5%	33.8%
Logistic Regression	40.3%	29.7%	34.2%
C4.5	74.1%	21.3%	33.1%
SVM	82.2%	22.8%	35.7%

Figure 5.5: Results by classifiers

Pattern	Precision	Recall	F-Measure	
`* of # (* * * #) * *`	63.6%	17.3%	27.2%	
	overview of *avian influenza* (a.k.a. *bird flu*) and phases			
`# (#)†`	79.4%	13.4%	22.9%	
	community-acquired MRSA (*CA-MRSA*)			
`of * # (* #) * *`	65.1%	13.9%	22.9%	
	of non-*severe acute respiratory syndrome* (Non-*SARS*)-related human			
`and # * (# *`	75.0%	13.4%	22.7%	
	and *hepatitis C virus* infection (*HCV*) and			
`*, * # * (# * *`	60.0%	13.4%	21.9%	
	septicemia, anthrax, *swine* fever (*hog* cholera), and			
`against # (* #`	64.0%	7.9%	14.1%	
	against *enterotoxigenic Escherichia coli* (or *ETEC*			
`prevalence of # * #`	26.9%	8.9%	13.4%	
	*prevalence of *FeLV* and *FIV* prevalence of *paratuberculosis* (ie, *Johne's Disease*)			
`patients with # * * * * # * *`	12.8%	7.4%	9.4%	
	patients with *HTLV-associated myelopathy* (HAM)/ tropical spastic paraparesis (*TSP*)			
`* * # * known as #†`	20.0%	0.5%	1.0%	
	*an atypical *pneumonia*, also known as *severe acute respiratory syndrome*			
`#/#†`	25.0%	1.5%	2.8%	
	**HIV/AIDS* *HTLV-associated myelopathy/tropical spastic paraparesis*			
`#, #†`	1.7%	63.4%	3.3%	
	**measles*,*mumps*, rubella vaccine due to efforts to eradicate *poliomyelitis*, *polio* cases have fallen			

Table 5.1: Individual pattern results and example matches. (†: suggested in Yu and Agichtein [2003]; *non-synonymous term pair; *terms identified*)

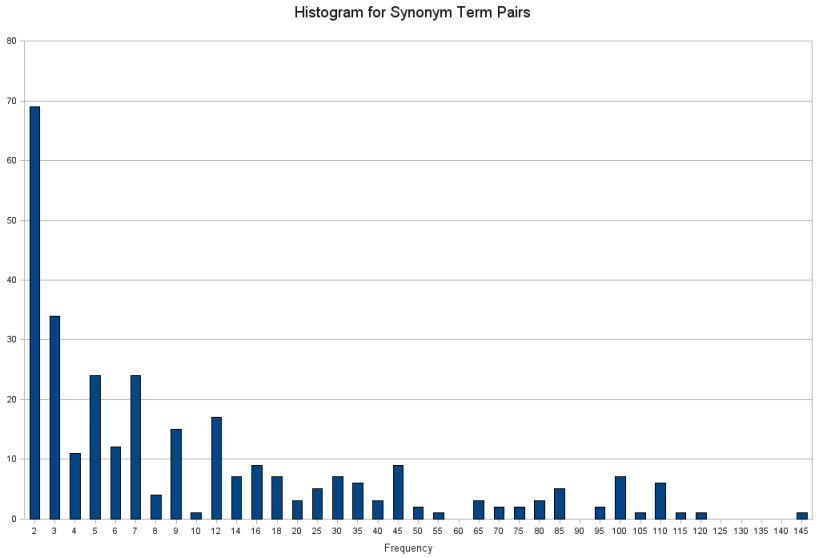

Figure 5.6: Frequency of synonym pairs from "disease" in PubMed corpus

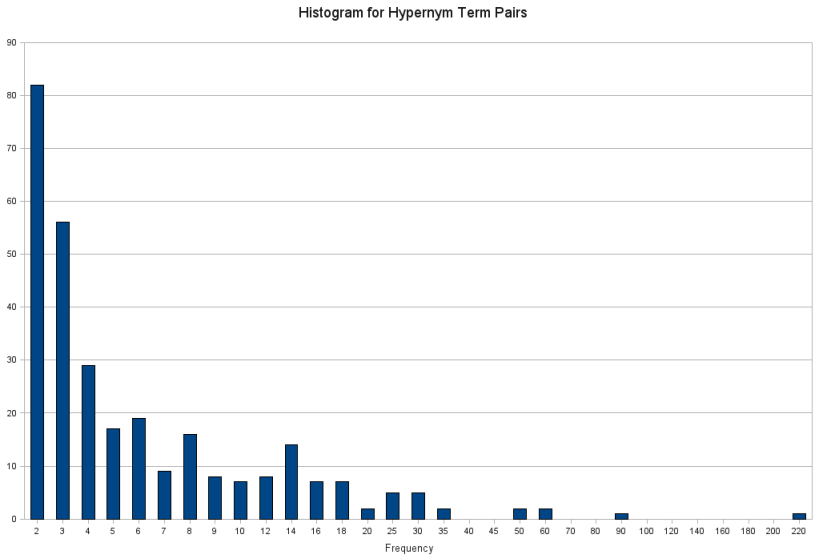

Figure 5.7: Frequency of hypernym pairs from "disease" in PubMed corpus

111 synonym term pairs occur more than 10 times. I then used the process as in definition 3.1.3 to create a set of base patterns, giving 2176 base patterns for synonymy and 1522 base patterns for hypernymy, I then calculate all join-sets of these patterns giving 1814 new pattern for synonymy and 3835 patterns for hypernymy. In comparison to the method not using join-sets there were 9747 patterns for hypernymy that would be generated and 13932 for synonymy. This represents a large reduction of the search space for both relations, in fact 87.0% for synonyms and 60.7% for hypernymy, which clearly shows the value of using join-sets. Then I chose to take only those patterns which match at least 3 contexts and have a precision greater than 10%, this drastically cuts down the number of patterns to 10 synonym patterns and 45 hypernym patterns. These patterns were then used to create vectors, and an SVM was trained using the matches between the seed terms as the training data. I then applied this to all the extracted vectors and manually evaluated the correctness of the result, finding that for hypernymy I extracted 127 term pairs of which 80 (63.0%) were correct and of those 47 were not in the original training set (58.8%), for synonymy I found only 16 term pairs, 9 of which were correct (56.3%) and 6 were new (66.7%).

These results were not particularly strong so I decided to repeat the experiment this time adding the new terms (regardless of whether they were correct or not) as new training data and then used these results to further expand the sets. Figures 5.8 and 5.9 show the results at every iteration, the results are improved for synonymy which by the final iteration has improved to finding 93 terms of which 61 (65.6%) are correct 53 (86.9%) and were not in the original set. The results for hypernymy improve for the first couple of iterations producing 184 term pairs (107 correct and 78 new), before slightly decreasing.

I then ran the same system on a larger corpus using more term pairs. I took all term pairs from WordNet, which are hyponyms of the term "organism", which gave a total of 35,718 terms, with a total of 32,247 synonymous term pairs and 527,985 hypernymous term pairs. I then collected a corpus, consisting of the documents from the Wikipedia 2009 DVD collection[4], which gave a total corpus of about 28,000,000 words. I then used these terms as seed term pairs and used the pattern based extraction system of section 3.2 to extract term pairs for both synonymy and hypernymy, these values were then compared to WordNet, so it is likely that the precisions reported are lower than actually obtained due to correct synonym/hypernym pairs being absent from WordNet. Then I used the equivalence logic (see figure 5.11) on the synonym results, to form a consistent network and similarly the hierarchical logic (see figure 5.11) on the hypernym results, and the Equivalence-Hierarchical logic (see figure 5.11) on both sets. In figure 5.10 I show the results of this analysis, and it is clear to see that for both hypernymy and synonymy applying the logic consistency algorithm improves the overall performance (from 67.1% to 68.7% and 47.6% to 49.6% respectively). The number of synonym term pairs extracted is much lower however it should be noted that both this is probably related to the

[4]Available at http://schools-wikipedia.org/

fewer number of seed term pairs for synonymy, and as such the lower number is due to the much lower number of synonym pairs that could be extracted. For the combined system I show the result obtained from separately applying equivalence logic to the synonym results and hierarchical logic to the hypernym system compared to applying the equivalence-hierarchical logic. These results show that applying the logic method separately improves the result from 66.2% to 67.80% and by applying the combined logic it improves to 67.84%, which is a slight improvement but cannot be considered significant.

5.3 Algorithmic complexity results

5.3.1 Simulation Method

To test my method I require that I have a methodology that can create a data that resembles the expected output of the pattern extraction system. Firstly I start with a network, N, which is consistent with a set of restrictions, T. However it is desirable that this network has a similar density to actual ontologies, I define this by the use of a value called *connectivity*, which is defined as the network, where n^{-c} of the elements are in the network, where c is the connectivity. I use this rather than a straight percentage as there are a number of interesting properties of this value which come from the theory of random graphs. First, it should be rather obvious that for $c > 2$, the random graph generated will not contain any links as there are n^2 links.

Theorem: If G is a graph generated with probability n^{-c}, define a connected component as a subset of the nodes of G, such that there is a sequence of links in G connecting each element of the subset. Denote the size of the largest connected component as $lcc(G)$. Then as $n \to \infty$, then

$$\begin{array}{ll} P(lcc(G) > k) \to 1 & \text{for } c > \frac{1}{k+1} \\ P(lcc(G) < k) \to 0 & \text{for } c < \frac{1}{k+1} \end{array}$$

This theorem states that if $c = \frac{1}{k+1}$, then it is highly unlikely that there will be a larger connected component than one of size k, which helps limit the complexity of the model. Another related result also states that the graph is likely to become completely connected when the value of $c = \frac{\ln n}{n}$. The proof of both of these results can be found in Spencer [2000].

I can simply generate a network at random by saying that $r(e_1, e_2) \in P$ with probability n^{-c}, and then require that a network is found that is consistent with the network. I use the GSAT algorithm to find a network but I slightly modify it so that step 2a now reads

> *2a.* Choose the link $r(e_1, e_2)$ which maximises
>
> $$c(r(e_1, e_2))consist - gain_N(r(e_1, e_2))$$
>
> If the maximal value for $consist - gain_N(r(e_1, e_2))$ is zero or negative then **fail**. If multiple such links exist choose one *at random*.

This removes the worst part of the computational aspect from GSAT, that is "flipping" links at random until it can find a way to make process. I then generate a consistent network in the following way

Algorithm

Input: A connectivity value c, node count n and set of ground instances T.

Output: A network, N consistent with T.

1. Generate a random network N^0 with n^{1-c} links.
2. Use modified GSAT algorithm (algorithm 4.3.7.9) to find a network N. If the GSAT algorithm failed go to step 1 otherwise **output** N.

I now need to generate some probability values from my method, which I do by generating an *evidence value*, which I assume is distributed normally with mean μ_0 or μ_1 and variance σ_0 or σ_1, where μ_1 is the value for links $r(e_1, e_2) \in N$ and μ_0 for values $r(e_1, e_2) \notin N$, a σ_0 and σ_1 the standard deviation. I then use this value to train a logistic regression system and use this to create probability values much like the actual process. I can change the values of μ_0 and μ_1 to adjust the values of the precision/recall of the data and I shall assume that $\mu_1 > \mu_0$. Firstly I shall assume that the logistic regression procedure finds the optimal decision boundary $x = \phi$ given by

$$p(r(e_1, e_2) \in N)P(\phi | r(e_1, e_2) \in N) = p(r(e_1, e_2) \notin N)P(\phi | r(e_1, e_2) \notin N)$$

This can be then be restated as

$$\frac{n^{-c}}{\sigma_1\sqrt{2\pi}} exp(-\frac{(\phi - \mu_1)^2}{2\sigma_1^2}) = \frac{1 - n^{-c}}{\sigma_0\sqrt{2\pi}} exp(-\frac{(\phi - \mu_0)^2}{2\sigma_0^2})$$

This can be simply rearranged into a quadratic equation in ϕ and the boundary value found. I shall use Φ to denote the cumulative density function of the normal distribution with mean 0 and SD 1 i.e.,

$$\Phi(x) \equiv \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} exp(-\frac{x^2}{2})$$

It can now be said that the expected number of true positives is given by

$$\widehat{TP} = n^{c+1}(1 - \Phi(\frac{\phi - \mu_1}{\sigma_1}))$$

And the expected number of false negatives given by

$$\widehat{FN} = n^{c+1}\Phi(\frac{\phi - \mu_1}{\sigma_1})$$

And the expected number of false negatives given by

$$\widehat{FP} = (n^2 - n^{c+1})(1 - \Phi(\frac{\phi - \mu_p}{\sigma_0}))$$

I can now use these values to calculate suitable values of μ_0,μ_1,σ_0 and σ_1 to achieve suitable values for the error rates. The formulae I presented are not easily solved algebraically so instead I simply use *gradient descent* to efficiently calculate suitable values. Once these values are calculated, it is easy to create a simulated network with probability values for each particular link, by training a logistic regression classifier with a random set of data, generating evidence values for each link and using the classifier to gain output probabilities.

5.3.2 Simulation results

For equivalence methods I have a several algorithms that I have already mentioned in this paper to compare to. I shall briefly recap them:

- **Greedy Sets (Algorithm 4.2.1.1)** This is a basic method of greedily adding sets until it finds an exact cover of the problem.
- **Set Branch and Bound (Algorithm 4.2.1.16)** This is an improvement on the Greedy Sets algorithm, which uses several optimisations, and then covers the search space using a Branch and Bound methodology. It finds the optimal solution.
- **Greedy Construction (Algorithm 4.3.2.3)** This is a simple approach based on using the equivalence and adding the link consistently until a network is obtained.
- **NoRes Relaxation (Algorithm 4.3.4.4)** This algorithm introduces negative value in later iterations to build a matrix, which ensures consistency with all rules. It produces an optimal solution.
- **Resolution Relaxation (Algorithm 4.3.5.8 and 4.3.6.6)** This algorithm grows a matrix by using resolution, it then applies an extra stage of matrix reductions, which are not available to the NoRes Relaxation Algorithm. It produces an optimal solution.
- **GSAT (Algorithm 4.3.7.8)** This algorithm works by flipping a link one at a time in a greedy manner until it reaches a consistent solution.

I present these methods over results using three different set of axioms shown in figure 5.11

Figure 5.12 shows the F-Measure based on simulate data for the equivalence logic, these results are all averaged on 10 runs. The baseline value is labelled "Baseline" and is the result with no attempt to improve the result, I generate this around 85% mark and as can be seen the simulate data generator is reasonably good at finding a network that is approximately 85% correct. My results are all averaged over 10 runs for each system, and the line labelled "Optimal algorithms" covers the algorithms "Set Branch and Bound", "NoRes Relaxation" and "Resolution Relaxation", all of which produce the same optimal solution as defined in definition 4.3.1.7. I conclude from this result, that "Greedy Sets"

algorithm produces the poorest result, often not improving significantly on the base-line value, this is due to its simple approach and tendency to ignore large values. A better result is obtained from the "Greedy Construction", which produces between 90% and 95% accuracy which is good but still significantly lower than the true optimal value. Finally, the results show that the "GSAT" algorithm is consistently poorer than any of the exact algorithms, however it is often within a few percentage points. As expected in all cases the highest value is gained by exact algorithms.

Figure 5.13 shows the computation times for each of these algorithms, and there is one immediately surprising result and that is the very high computation times for "Greedy Construction", which is a polynomial time algorithm! In fact, this is not hard to explain as one of the aspects, which this algorithm has to perform is a consistency check at each iteration, and as the equivalence logic is being used, there is a clause

$$r(x_1, x_2), r(x_2, x_3) \rightarrow r(x_1, x_3)$$

This clause has n^3 ground instances, and each of these instances must be checked to see if they have become inconsistent at every iteration of the algorithm. Furthermore this algorithm starts out with $N = \emptyset$ and it needs to individually add each link to its final network, this means that this algorithm actually needs to do a lot more work than some of the other algorithm. The other algorithms all have very short run times, all far under $1/10^{th}$ of a second, it can be seen that these results seem to correspond to the following order: "GSAT" is the slowest, followed by "NoRes Relaxation", then "Set Branch and Bound", then "Resolution Relaxation" and finally "Greedy Sets". It should perhaps be no surprise that the algorithm, which produced the worst accuracy is also the quickest, and it does not have the problems of "Greedy Construction" as it does not need to evaluate logical statements as they are incorporated into the operation of the algorithm. Furthermore the extra matrix reductions that "Resolution Relaxation" has over "NoRes Relaxation" offer notable improvements in performance.

I also repeated the results for the other two logics. Figure 5.14 shows the F-Measure results for the hierarchical logic, for which I cannot apply methods of Greedy Sets and Set Branch and Bound, as these are specific to the equivalence logic. There are two differences in the results, firstly it can be seen that the Greedy Construction algorithm actually managed to perform worse than the base line, this is surprising but can probably be explained by the system picking up high scoring links too early which are not part of the actual optimal solution. The performance of the GSAT algorithm is also much closer to the optimal solution, in fact for $n < 60$ the GSAT algorithm is exactly on the optimum, and for higher n, the results are within 0.1% of the exact solution, which, as the results are averaged on 10 simulations, suggests that in nearly all runs the GSAT algorithm reached the optimal solution. Looking at the results for computation time over the same run in figure 5.15, the results echo those of the equivalence logic, in that Greedy Construction is much slower and then GSAT, then NoRes-Relaxation and finally Resolution-Relaxation. Finally I apply the

results to the combined Equivalence-Hierarchical logic, which is a model of the structure used by WordNet. In figure 5.16 the accuracy results are as expected a blend of the results we saw previously, again GSAT is very close to the optimal solution with similar differences of approximately 0.1%, the Greedy Constructor is closer to the baseline than in the hierarchical logic, but still mostly performs worse than the baseline. More interestingly the computation time in figure 5.17 show a very large spike for $n = 90$ and $n = 95$, this seems to affect all the algorithms, but mostly the Greedy Construction algorithm which increase to a rather high 10.75 seconds, this is assumedly due to the need for searching for consistency checks. As I have already mentioned the Greedy Construction algorithm checks for inconsistencies at every iteration, in contrast the GSAT algorithm checks periodically preferring to keep a possibly inaccurate set of inconsistencies in its gain/loss sets, and the relaxation methods may only check twice, once to build the initial matrix and once to check the first solution is consistent. To test this hypothesis I decided to profile a run where the Greedy Construction algorithm took a significant amount of time. In this run I found that the Greedy Construction algorithm took 16.37 seconds to complete[5], during its run it checked for inconsistent rules 101 times, taking 14.92 seconds in total (91.2%).

In figure 5.18, I compare the performance of the algorithms for larger values, it is clear that the *Greedy Construction* algorithm is not worth including in the analysis. These results show as before that the *NoRes-Relaxation* algorithm is significantly slower than the version that uses resolution. There is one interesting result though as for values near a 1000 the GSAT algorithm starts to become noticeably faster than the exact method of Resolution-Relaxation. This is not totally surprising, as the increased complexity of the problem as it grows, probably means the approximate algorithm becomes slightly more robust. Figure 5.19 shows that the quality of the GSAT algorithms approximation remains roughly constant in spite of the increase in complexity of the problem.

[5]There are significant overheads to using a profiler

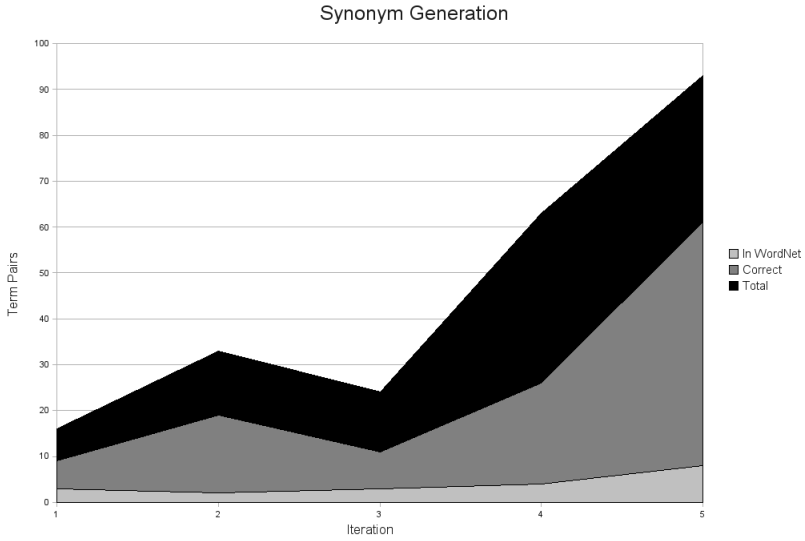

Figure 5.8: Extracting synonymous disease terms from PubMed corpus by iteration

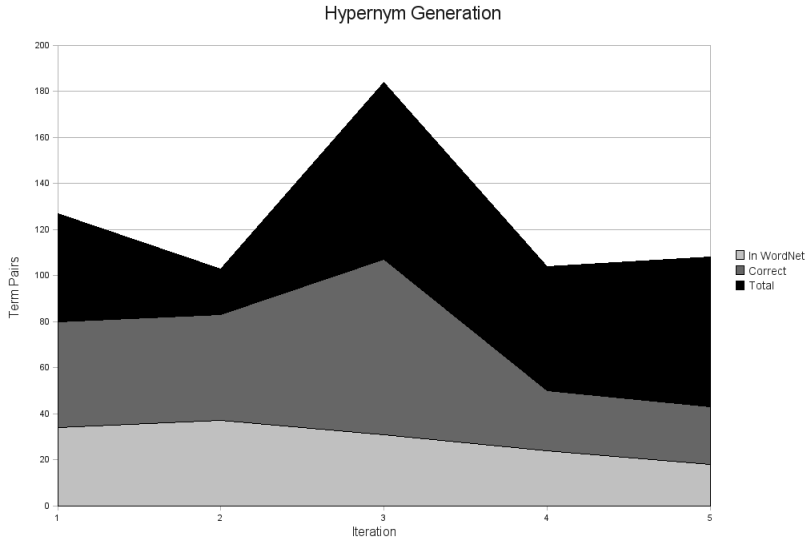

Figure 5.9: Extracting hypernymous disease terms from PubMed corpus by iteration

Links	Correct	Incorrect	Precision
Hypernym (before)	3941	1929	67.1%
Hypernym (after)	3814	1738	68.7%
Synonyms (before)	139	153	47.6%
Synonyms (after)	136	138	49.6%
Both (before)	4080	2082	66.2%
Both (separate)	3950	1876	67.8%
Both (together)	3919	1858	67.8%

Figure 5.10: Accuracy of pattern based extraction before and after applying logical consistency algorithms

Equivalence Logic:

- $r(x_1, x_2), r(x_2, x_3) \rightarrow r(x_1, x_3)$
- $r(x_1, x_2) \rightarrow r(x_2, x_1)$
- $\rightarrow r(x_1, x_1)$

Hierarchical Logic:

- $r(x_1, x_2), r(x_2, x_3) \rightarrow r(x_1, x_3)$
- $r(x_1, x_1) \rightarrow$

Equivalence-Hierarchical Logic:

- $r_1(x_1, x_2), r_1(x_2, x_3) \rightarrow r_1(x_1, x_3)$
- $r_1(x_1, x_2) \rightarrow r_1(x_2, x_1)$
- $\rightarrow r_1(x_1, x_1)$.
- $r_2(x_1, x_2), r_2(x_2, x_3) \rightarrow r_2(x_1, x_3)$
- $r_2(x_1, x_1) \rightarrow$
- $r_2(x_1, x_2), r_1(x_2, x_3) \rightarrow r_2(x_1, x_3)$
- $r_2(x_1, x_2), r_1(x_1, x_3) \rightarrow r_2(x_3, x_2)$

Figure 5.11: The Hierarchical, Equivalence and Equivalence-Hierarchical Logics

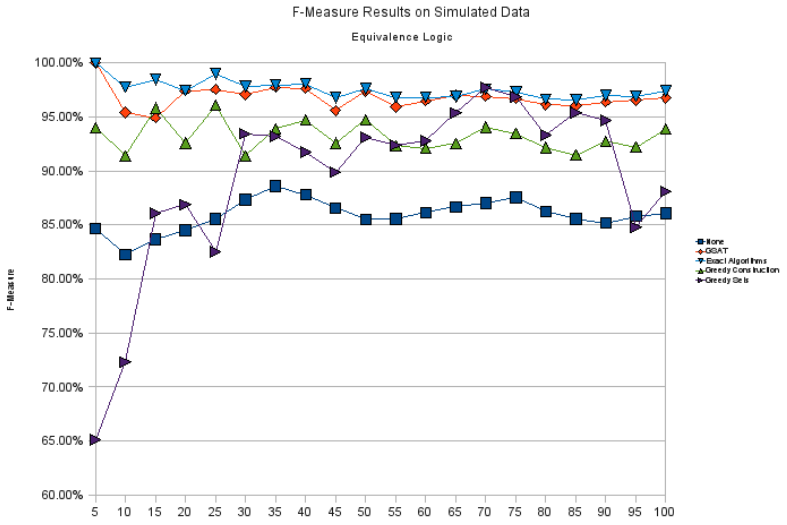

Figure 5.12: Analysis of accuracy results of Algorithms on Equivalence Logic

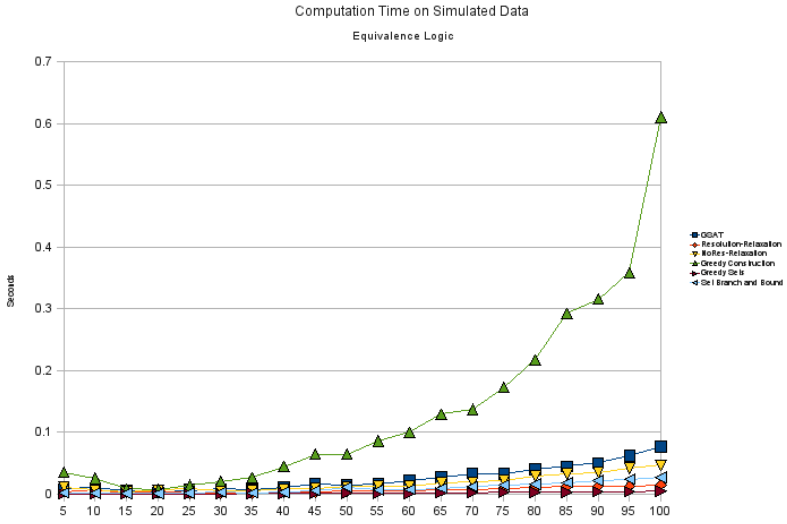

Figure 5.13: Analysis of computation times of Algorithms on Equivalence Logic

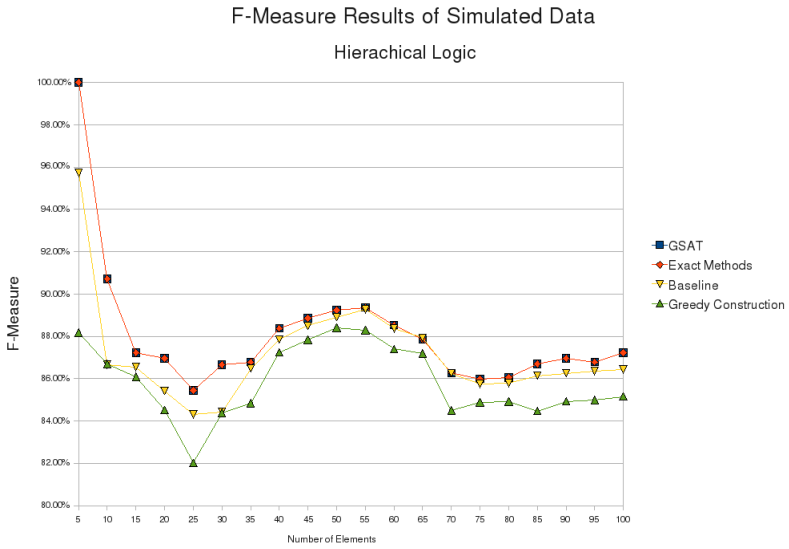

Figure 5.14: Analysis of accuracy results of Algorithms on Hierarchical Logic

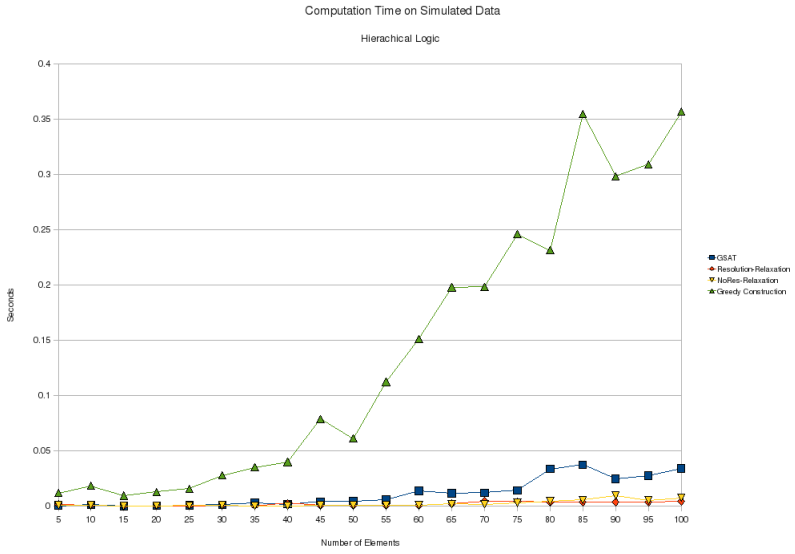

Figure 5.15: Analysis of computation times of Algorithms on Hierarchical Logic

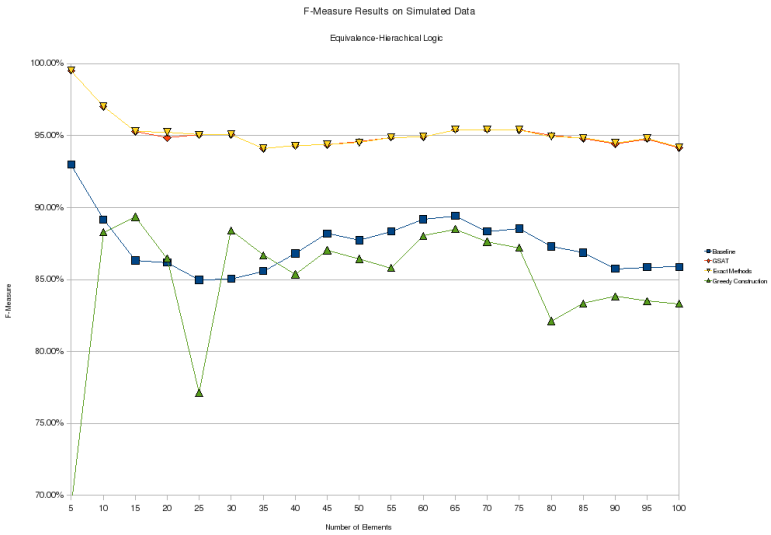

Figure 5.16: Analysis of accuracy results on Equivalence-Hierarchical Logic

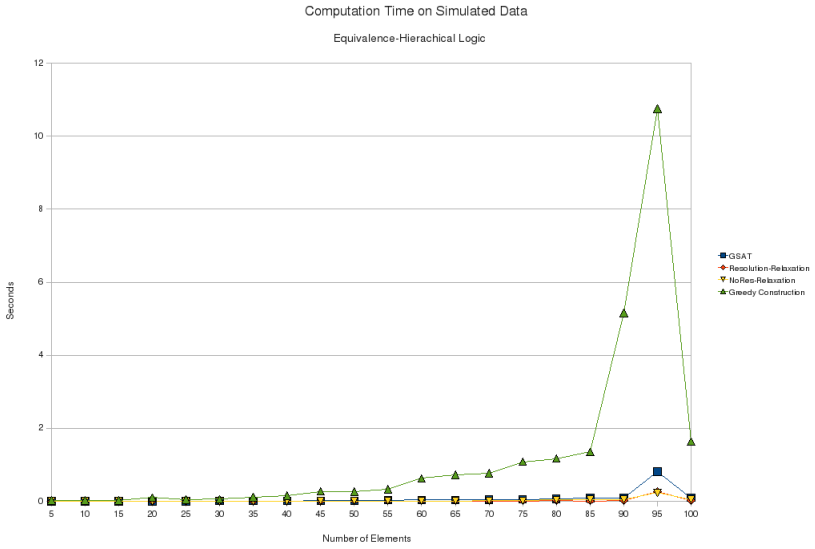

Figure 5.17: Analysis of computation times of Algorithms on Equivalence-Hierarchical Logic

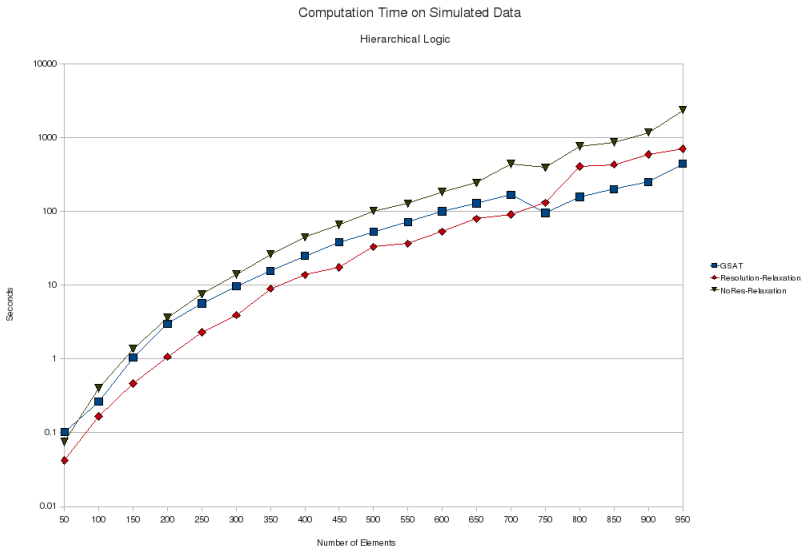

Figure 5.18: Analysis of computation times of Algorithms on large sets

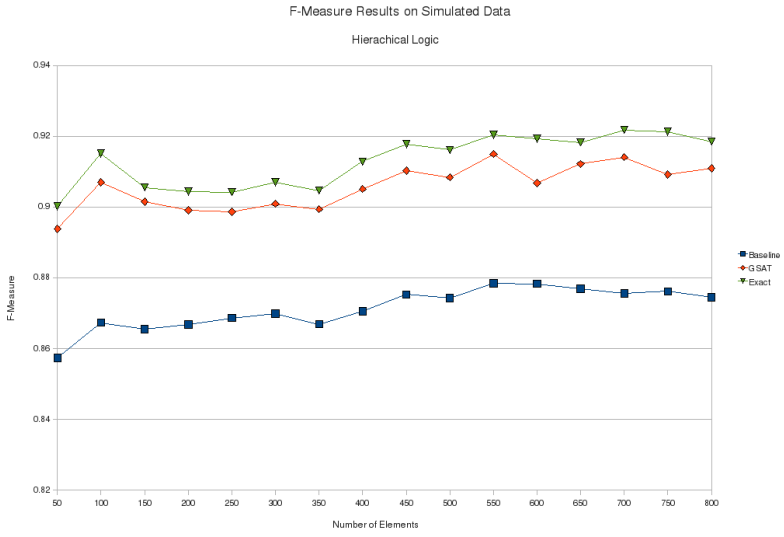

Figure 5.19: Analysis of accuracy results of GSAT on large sets

Chapter 6

Conclusion

In this paper, I considered the problem of extracting ontologies from free text, and ensuring that the extracted ontologies created have a certain structure. My methodology for this is to create patterns, and use these to create vectors which can be used with a statistical classification methodology to give a probability that a term pair is related. These probabilities give a network, however as the result is noisy and incomplete, it is unlikely that it will match the known structure of the ontology. I encode the structure of the ontology as a list of axioms, and then I define the problem of finding the correct network or ontology as finding the network which is most likely, given the probability values, and is consistent with all the axioms. I approach this by viewing this as a number of "corrections" that need to be made to the network, where the "corrections" are adding or removing links from the network with least probability until a consistent network is achieved. I show that these corrections can be tabulated into a matrix and the problem then converted to that of a *integer programming problem*, however this by itself is not sufficient to guarantee a consistent solution. I show how the matrix can be extended further by the use of resolution, which I show can generate the optimal consistent network. Finally I expanded the form of logics into a form called *Conjunctive Skolem Normal Form*, which is equivalent to first-order logic and show that this can be converted back to a simple logic form, which my algorithm, with only minor modification, can solve.

The first part of my system is the pattern extraction system, initially I based it on the WHISK system of Soderland [1999]. My system is presented in section 3.1 and I applied it to the problem of extracting synonyms from free text. As this system is not easily capable of dealing with multi-word terms I instead ran it on a pre-determined set of terms, so the system did not have to tag the corpus or use any other linguistic resource. The results from this data was very high, scoring an F-Measure of 35.7%, which was significantly higher than the result I obtained for most existing resources. In fact the only resource that out-performed it was the meta-thesaurus UMLS, which combines many thesauri into a single thesaurus, and my method still out-performed one of its component MeSH. This is quite satisfying as MeSH is itself a very complete thesaurus

and took many millions of man-hours to compile, where as my system took only 51 hours to complete. Furthermore, my system was capable of discovering synonymous pairs that were entirely absent from these resources and the recall of the system was, while low at only 29.7% still significantly higher than that of MeSH or Wikipedia and is strong enough for practical purposes. As such it is valid to conclude that this kind of methodology can be useful for creating new thesauri for domains or languages where resources of the scope of MeSH or UMLS do not exist, or for finding new terminology for the expansion of a resource like MeSH and UMLS.

The primary problem I discovered with the first system was that it found a strong pattern it would then create a huge number of minor variations of these patterns, which matched the same set or nearly the same set of contexts. As such this meant that the methodology spent a lot of its computation time on developing patterns which would not contribute to the overall effectiveness of the algorithm. I solved this by considering the match set of a pattern, that is the number of contexts from the corpus that it matches. The results of section 3.2.2 show that there is a connection between this concept of join-set and the *alignment* of two patterns, and as such I developed a new algorithm, which could find patterns by finding alignments between patterns. The results I obtained show that this method can reduce the size of the search space by 50-95%, while not removing any "useful" patterns from the search space. Furthermore I used a method based on tagging the corpus to identify terms, which allows for patterns to be developed which can find relationships between terms, not previously manually identified. This makes the system much more flexible but required a more complex rule language, for which I take the SRL language. In addition the join-set methodology allows for more complex rule matching elements based on features of the text such as orthographic elements or fixed list of words.

The next part of the problem is finding an ontology from the extracted relations. This is necessary due to the unreliable nature of the extraction systems and can be done by encoding the structure of the ontology as a set of axioms. Initially, I tackled the problem of synonym set structures and here the axioms suggest that the form of these is as a set of disjoint sets. This leads to a number of algorithms that can efficiently solve the problem. First, I present in section 4.2.1 a number of basic approaches to the problem such as *greedy search* and *branch and bound search*. These are methods, which are applicable to a wide range of problems, however the former often produces very poor solutions and the latter can quickly become intractable with the number of terms I expect to see. As such I decided to approach this by finding which sets were more likely and which sets could not possibly be part of the solution method. This leads to a much more efficient solution method. Then I present a method for representing the search space on these sets as a *integer programming problem*, which is useful as it allows for a *relaxed* version of the problem. Relaxing a problem is often highly advantageous as it allows for the "easy" parts of the problem to be solved efficiently and reduces the amount of work the non-polynomial parts of the algorithm have to perform (in fact often to zero). This algorithm was applied to the data extracted in McCrae and Collier [2008] and I found that there

was a significant increase in the performance of the result. I found that this increased not only the overall performance but was also effective for increasing the "marginal" performance, in that it removed more term pairs that were not semantically related than those which had some kind of semantic relation.

The next step is to look at different structures given by different relations. Another important relationships is hypernymy, which is a hierarchical relationship. While, it would be possible to develop a methodology adapted to hypernymy as I did for synonymy, in fact the straightforward approach is to ensure that each "chain" is complete: that is logic statements such as the following hold

$$r(e_1, e_2) \wedge r(e_2, e_3) \rightarrow r(e_1, e_3)$$
$$r(e_1, e_3) \wedge r(e_3, e_4) \rightarrow r(e_1, e_4)$$
$$r(e_1, e_2) \wedge r(e_2, e_3) \wedge r(e_3, e_4) \rightarrow r(e_1, e_4)$$

In fact it is easy to see that third rule is in fact the resolvent of the first two, as such it is clear that developing all possible "chain" conditions, is the same as resolution. Furthermore, resolving all such "chain" rules is generally not possible so instead I developed a methodology that only resolves these rules as necessary. This method as presented in section 4.3 has the benefit of not begin limited to hypernymy but applicable to a much wider range of ontology structures. As with synonymy I found it efficient to convert these rules into a matrix form so the search space could be solved as an *integer programming problem* and then relaxed to an *linear programming problem*. I also observed that the matrix could be reduced by the use of simple reduction to remove rules, which cannot contribute of the optimal solution. This essentially means that the algorithm is in fact three algorithms of increasing complexity which solve parts of the problem, first the reduction algorithms solves the trivial parts of the problem, then the linear programming algorithm solves the parts of the problem, which can be handled in linear time, and finally a branch and bound methodology solves the remainder. For comparison I developed two other algorithms, one which creates a matrix without the need for resolution and while this algorithm is of interest, it is fundamentally worse than the resolution based method, as its rule adding procedure is not noticeably more light-weight and it cannot apply the same reduction principles. Finally, I showed how the method could be approached as a *MAX-SAT* problem, and although this creates several problems, I show how a well-known algorithm called GSAT can be adapted to handle the problem.

To evaluate the effectiveness of the algorithms for logical consistency, it would not be possible to use just the data from the pattern based extraction system and for this reason, I evaluated all the systems on simulate data. The results for synonymy confirm that greedy approaches produce poor results but can be very fast, as shown by the performance of the *Greedy Sets* algorithm. The *Greedy Branch-and-Bound* method did not implement linear relaxation, to show the effect this has on the solution and as expected this means that the method takes slightly longer. Comparing the results of the *NoRes-Relaxation* and *Resolution-Relaxation* algorithms, it is clear to see that the effect of the matrix reduction produce a significant increase in performance. Perhaps the

most interesting result though was that the *Greedy Construction* algorithm always took so long to calculate. As this is a greedy algorithm, it is expected that it could efficiently find a suboptimal solution, however in fact not only was the solution poor, the algorithm was very slow. My results show this is nearly entirely to due with the cost of finding inconsistent ground instances, which takes a long time as the number of clauses that exist is often in the millions. The *GSAT* algorithm was a powerful approximation method, and I discovered that for the most part it could find a solution that was reasonably close to the optimal solution, my results show that for small sized problems its performance is worse than the exact methods, but for larger values (of the order of thousands of elements), the GSAT algorithm has slightly better performance with little degradation of its approximation ratio. The GSAT method does have a fundamental problem in that it is not guaranteed to terminate, although in no run did this occur.

I applied the systems I developed to the real world problems of extracting synonymy and hypernymy. I chose these relations as they have strong structure, interdependence and are easy to extract from raw text, although my system is also applicable to a wide range of other relations. Firstly I applied my initial system based on patterns and logical consistency by exact cover to a corpus extracted from PubMed and a small manually extracted term set. These results showed that my system was comparable in its performance to MeSH a thesaurus specifically designed for the domain of this task. As MeSH is a resource that took thousands of man-hours to compile it is impressive that my system could produce a similar quality result in a much shorter time. These results were then put into the exact cover finding system of section 4.2, and the results show that this system was capable of improving the overall performance.

The rule based on system of section 3.2 was applied to two sets of data, one was domain specific and the other was a more general resource. The first set I extracted was all disease names from WordNet and I applied it to another corpus of PubMed abstracts. These results show again that the system was capable of extracting related term pairs with reasonable precision, however the nature of the data as shown by the histogram plot, shows one of the key limitations for the method: that most of the term pairs from WordNet occurred very few times in the same contexts. I then performed a run on a general test set based of all terms from WordNet that were hyponyms of "organism" and a corpus consisting of 20,000 articles from Wikipedia, these results again had good precision. I applied the logical consistency algorithm of section 4.3 to these results, both separately for synonymy and hypernymy and using a logic that represents the inter-dependencies between synonymy and hypernymy. The results showed that by applying this method the precision of the result was improved, and although the combined logic did not produce a significant improvement over the separate logic.

In this paper I do not handle the problem of polysemy, that is of terms, which refer to several different concepts. In general this method is intended to extract terminology for a specialised domain and in that case terms are normally specific to a single concept. However I accept that there is a need to include

some sense disambiguation into the process and think this is a potential area for future study. The pattern system I have developed is applicable to a more complex rule types than my implementation has produced, for example as in the discussion of section 3.2.3. I think there is room for more investigation into the type of matching elements that would be useful for more complex patterns and possibility of using patterns based on a parsed tree structure of the texts, which I believe could be handled by the use of join-sets and alignments. The algorithm I presented for finding consistent logical ontologies, I feel is very powerful, but for very large networks (i.e., with $n > 100,000$), I suspect this exact method would not be tractable. The best approximation method I have presented is likely not significantly faster and is not guaranteed to terminate. As such there may be a need to develop a robust approximate algorithm if this methodology needs to be applied to very large networks.

The algorithms presented here have are very flexible and potentially suitable for many other tasks. The pattern extraction system I have presented can handle very general types of rules, and it is possible to extend it to be an automatic rule generator for a language such as SRL. As SRL is designed for extracting and tagging a wide variety of facts from raw text, my algorithms should be capable of handling extraction tasks than simply relations. The natural extension of this work is to improve recall by finding pairs that do not occur in the same contexts, and I feel that by using some form of reference resolution it should be possible in the future to incorporate such a system. The task of forming the result into a consistent ontology is stated in a general logical framework. The problem can be essentially viewed as finding the most likely version of structured data from noisy information about its particular components, and as such this it is possible to conceive that there are many other tasks, where this approach may prove useful.

In this paper I presented a system that is capable of extracting semantically related terms from raw text by the use of rules. I performed this by using a novel method that reduces the search space by eliminating rules, which are not different based on the data in the corpus. This method also has the advantage that it is applicable to rules using more complex matching elements. I then considered the problem of adding these extracted relations to an ontology, and I formulated this problem by defining the structure of an ontology as a set of axioms. I showed that this method gives a NP-hard problem, which requires novel algorithms to solve efficiently. I present a new algorithm based on the use of integer programming, linear relaxation and resolution, which is capable of solving this problem optimally in a very short amount of time. Finally I show that not only does making the result logically consistent allow it to be added to a structured ontology, but it can also improve the quality of the result of the extraction system.

Bibliography

Eugene Agichtein and Luis Gravano. Snowball: extracting relations from large plain-text collections. In *DL '00: Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94, 2000.

Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, USA, 2004.

Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The Berkeley FrameNet Project. In *Proceedings of the 17th international conference on Computational linguistics*, pages 86–90, 1998.

Regina Barzilay and Lillian Lee. Learning to Paraphrase: An unsupervised approach using multiple-sequence alignment . In *Proceedings of HLT-NAACL 2003* , pages 16–23, 2003.

David Bean and Ellen Riloff. Unsupervised learning of contextual role knowledge for coreference resolution. In *Proc of HLT/NAACL*, 2004.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web, 2001.

Daniel M. Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201, 1997.

Olivier Bodenreider, Anita Burgun, and Thomas Rindflesch. Lexically-suggested hyponymic relations among medical terms and their representation in the UMLS. In *TIA'2001: Proceedings of Terminology and Artificial Intelligence*, pages 11–21, 2001.

Olivier Bodenreider, Anita Burgun, and Joyce A. Mitchell. Evalutation of WordNet as a source of lay knowledge for molecular biology and genetic diseases: A feasibility study. In *MIE*, 2003.

Eric Brill. A simple rule-based part of speech tagger. In *HLT '91: Proceedings of the workshop on Speech and Natural Language*, pages 112–116, 1992.

Scott Cederberg and Dominic Widdows. Using LSA and noun coordination information to improve the precision and recall of automatic hyponymy extraction. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 111–118, 2003.

Chia-Hui Chang and Shao-Chen Lui. IEPAD: information extraction based on pattern discovery. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 681–688, 2001.

P. Cimiano, A. Hotho, and S. Staab. Comparing conceptual, divisive and agglomerative clustering for learning taxonomies from text. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 435–439, 2004.

Philipp Cimiano and Steffen Staab. Learning by Googling. *SIGKDD Explor. Newsl.*, 6(2):24–33, 2004.

Philipp Cimiano, Andreas Hotho, and Steffen Staab. Learning concept hierarchies from text corpora using formal concept analysis. *Journal of Artificial Intelligence Research (JAIR)*, 24, 2005.

Nigel Collier, Chikashi Nobata, and Jun-ichi Tsujii. Extracting the names of genes and gene products with a hidden markov model. In *Proceedings of the 18th conference on Computational linguistics*, pages 201–207, 2000.

Nigel Collier, Son Doan, Ai Kawazoe, Reiko Matsuda Goodwin, Mike Conway, Yoshio Tateno, Quoc-Hung Ngo, Dinh Dien, Asanee Kawtrakul, Koichi Takeuchi, Mika Shigematsu, and Kiyosu Taniguchi. BioCaster: detecting public health rumors with a Web-based text mining system. *OUP Bioinformatics*, 24(24):2940–2941, 2008.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2004.

George Dantzig. Application of the simplex method to a transportation problem. In *Activity Analysis of Production and Allocation*, pages 359–373, 1951.

Dmitry Davidov, Ari Rappoport, and Moshe Koppel. Fully unsupervised discovery of concept-specific relationships by web mining. In *Proc. of 45th Annual Meeting of the Assocation of Computational Linguistics*, pages 232–239, 2007.

Martin Davis and Hillary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

Katerina Frantzi, Sophia Ananiadou, and Hideki Mima. Automatic recognition of multi-word terms:. the c-value/nc-value method. *Natural language processing for digital libraries*, 3(2):115–130, 2000.

Aldo Gangemi, Nicola Guarino, Claudio Masolo, and Alessandro Oltramari. Sweetening WORDNET with DOLCE. *AI Mag.*, 24(3):13–24, 2003.

Julio Gonzalo, Felisa Verdejo, Irina Chugur, and Juan M. Cigarrán. Indexing with WordNet synsets can improve Text Retrieval. *CoRR*, cmp-lg/9808002, 1998.

Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

Jun Gu. The UniSAT problem models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(8):865, 1992.

Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (sat) problem: A survey. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 19–152. American Mathematical Society, 1996.

Nicola Guarino and Christopher Welty. Ontological analysis of taxonomic relationships. *Lecture Notes in Computer Science: Conceptual Modelling*, 1920: 273–290, 2000.

Thierry Hamon, Adeline Nazarenko, and Cécile Gros. A step towards the detection of semantic variants of terms in technical documents. In *ACL-36: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 498–504, 1998.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer Verlag, 2001.

Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, 1992.

Juraj Hromkovič. *Algorithmics for Hard Problems*. Springer Verlag, 2002.

Fidelia Ibekwe-SanJuan. A linguistic and mathematical method for mapping thematic trends from texts. In *Proceedings of the 13th European Conference on Artificial Intelligence*, pages 170–174, 1998.

Christian Jacquemin. Syntagmatic and paradigmatic representations of term variation. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 341–348, 1999.

Leonid Khachian. A polynomial time algorithm for linear programming. *Soviet Mathematics, Doklady*, 20:191–194, 1979.

Donald Knuth. Dancing links. In *Millenial Perspectives in Computer Science*, pages 187–214, 2004.

Vipin Kumar. Algorithms for constraint-satisfaction problems: A survery. *AI Magazine*, 13(1):32–44, 1992.

Douglas B. Lenat. CYC: a large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

Dekang Lin. Dependency-based evaluation of minipar. In *Workshop of the Evaluation of Parsing Systems*, 1998.

Katja Markert and Malvina Nissim. Comparing knowledge sources for nominal anaphora resolution. *Computational Linguistics*, 31:367–401, 2005.

John McCrae and Nigel Collier. Synonym set extraction from the biomedical literature by lexical pattern discovery. *BMC BioInformatics*, 9(159), 2008.

Emmanueal Morin and Christian Jacquemin. Automatic acquisition and expansion of hypernym links. *Computers and the Humanities*, 38:363 – 396, 2004.

Saul Needleman and Christain Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

Goran Nenadić, Irena Spasić, and Sophia Ananiadou. Automatic discovery of term similarities using pattern mining. In *COLING-02 on COMPUTERM 2002*, pages 1–7, 2002.

Andreas Nomengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. *Lecture Notes in Computer Science: Automatic Deduction CADE-15*, 1421:397–411, 1998.

Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of english words. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 183–190, 1993.

Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

Ellen Riloff. An emperical study of automated dictionary construction for information extraction in three domains. *Artificial Intelligence*, 85:101–134, 1996.

Gian-Carlo Rota. The number of partitions of a set. *American Mathematical Monthly*, 75(5):498–504, 1964.

Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth Annual Conference on Artificial Intelligence*, pages 440–446, 1992.

Barry Smith and Christiane Fellbaum. Medical WordNet: a new methodology for the construction and validation of information resources for consumer health. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 371, 2004.

Rion Snow, Daniel Jurafsky, and Andrew Ng. Learning syntactic patterns for automatic hypernym discovery. In *Proc. of Neural Information Processing Systems*, 2004.

Rion Snow, Daniel Jurafsky, and Andrew Ng. Semantic taxonomy induction from heterogenous evidence. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 801–808, 2006.

Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

Joel Spencer. *The strange logic of random graphs*. Springer Verlag, 2000.

Michael Q. Stearns, Colin Price, Kent A. Spackman, and Amy Y. Wang. Snomed clinical terms: Overview of the development process and project status. In *Proceedings of AMIA Symposium*, pages 662–666, 2001.

Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. SOFIE: A Self-Organizing Framework for Information Extraction. In *International World Wide Web conference (WWW 2009)*, 2009.

Ellen M. Voorhees. Using WordNet to disambiguate word senses for text retrieval. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 171–180, 1993.

Piek Vossen, editor. *EuroWordNet: a multilingual database with lexical semantic networks*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

Dominic Widdows and Beate Dorow. A graph model for unsupervised lexial acquisition. In *Proceedings of 19th International Conference on Computational Linguistics (COLING-19)*, pages 1093–1099, 2002.

Xiaofeng Yang and Jian Su. Coreference resolution using semantic relatedness information from automatically discovered patterns. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 528–535, 2007.

Mihalis Yannakakis. On the approximation of maximum satisfiability. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 1–9, 1992.

Alexander Yates and Oren Etzioni. Unsupervised resolution of objects and relations on the web. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 121–130, 2007.

Hong Yu and Eugene Agichtein. Extracting synonymous gene and protein terms from biological literature. *OUP Bioinformatics*, 19(1):340–349, 2003.

Hong Yu, Vasileios Hatzivassiloglou, Carol Friedman, Andrey Rzhetsky, and W. John Wilbur. Automatic extraction of gene and protein synonyms from medline and journal articles. In *Proc. AMIA Symp*, pages 919–923, 2002.

Bart Selman Yuejun Jiang, Henry Kautz. Solving problems with hard and soft constraints. In *Proceedings of the first internationl joint workshop on artificial intelligence and operations research* , 1995.