Department of Computer Science

Submitted in part fulfilment for the degree of
MEng in Computer Science With Artificial Intelligence.

Searching For Relationship Chains Within Web Ontologies

Daniel Banks

20th May 2020

Supervisor: Dimitar Kazakov

Contents

List of Figures

List of Tables

Abstract

In this report we assess the performance of different search algorithms to find meaningful relationships between two entities, such as individuals, companies or government bodies, comparing the performance of uninformed, informed, and bidirectional searches in their ability to find said relationships. The search is performed within web ontologies in order to find these meaningful relationships through specified relationship types, such as that between a company and its employees. The search algorithms are assessed in terms of their computational performance.

The experiment uses a series of companies and their employees in order to find a link from one person to another. Using the Statement of Changes in Beneficial Ownership documents of companies in the US, gathered from the trading of stocks by company insiders in Q1 of 2017. This data was converted into an ontology that could act as our search graph.

This study assesses the performance of several search algorithms to find these significant relationships between two people within the ontology, Iterative-Deepening, Breadth-First, and Best-First search and the bidirectional combinations of said searches. Each bidirectional search had only one of the active searches managing a fringe of visited node.

From this study we have been unable to identify which search algorithm is the most computationally efficient to use for such an endeavour. However, we have been able to assess that in the practical use of querying a web ontology, bidirectional search methods require fewer queries in order to find the optimal path at searches with a large number of intermediate nodes. This is extremely beneficial for a system with limited memory such as that used in this experiment.

Perhaps exploration of a more detailed ontology and execution in a programming language with better optimisation may yield better results.

Executive Summary

The aim of this report is to discover significant relationships between two seemingly unconnected entities using publicly available data hosted within web ontologies. In order to do this, we have assessed several search algorithms in their ability to find said relationships through querying the web ontology. The report uses data from 1000s of companies, representing the company insiders who traded their stock during Q1 of 2017. We have imported the data extracted from the Changes in Beneficial Ownership documents available into a web ontology that we can search. We use this ontology as a proof of concept for the viability of the search algorithms to find significant relationships on a much larger and detailed ontology.

Finding these significant relationships may prove useful for a range of real-world uses. A significant relationship between two entities using publicly accessible data can be used to identify and examine different companies for a high level of co-integration that could be good candidates for pairs trading, find conflicts of interest in the board members of a company, discover the presence of virtual monopolies, and other points of influence.

When examining a set of companies for high-level of cointegration for pairs trading, in 2015 during the Volkswagen emissions scandal, in which the stock price of both Volkswagen and their subsidiary company Audi fell dramatically. This parallel decline in the share prices of both companies shows a significant relationship between the two, by better understanding these significant relationships a more informed decision can be made when investing in a pair of companies. For example, if a company is doing well, we may wish to find other companies that appear to be following the same trend.

As Audi is a subsidiary of Volkswagen, they share a number of board members, this may be a contributing factor to the significant relationship between Audi and Volkswagen. From this we can assume that any companies that share board member may be good candidates for pairs trading and help us identify such companies.

Additionally, we may wish to find companies that hold a virtual monopoly for a commodity. Holding a virtual monopoly discourages competition between a few, very powerful companies and instead promotes the companies to artificially inflate their prices, because if they do the consumer has very little choice and are forced to pay higher prices. A good indicator for virtual monopolies is when a few companies in a certain field have a close relationship that dissuades competition, one factor to this relationship may be a few of the board of directors or other company insiders being shared by the two companies.

We hypothesise that the most efficient search algorithm for finding these significant relationships within a web ontology is an iterative deepening search and the implementation of a bidirectional search will significantly improve the efficiency of a search algorithm and significantly reduce the space complexity of the algorithm. Thus, we shall see a significant change in the execution of a search algorithm when paired with another for a bidirectional search.

In order to find the most efficient method of discovering these significant relationships we implemented a number of uninformed and informed searches to compare their average execution time over several specified relationships at a range of depths. As stated by Milgram[1] and Backstrom[2], most entities can be connected by between 4 and 6 intermediaries. Therefore, we concluded that significant relationship is one that has fewer number of intermediaries than the average.

We implemented searches to function on a web ontology by having the expansion of nodes executed by querying the ontology to find all the companies a person worked for and then all the other people who worked at said companies.

We then combined the searches we had implemented into a bidirectional framework and compared each combination's average execution time and average use of memory/number of queries made. We then compared the best performing singe and bidirectional search to assert whether there are any practical improvements made by implementing a bidirectional search.

Having completed the experiment, we are unable to access which of the search algorithms implemented were the most efficient. However, that is not to say that the search algorithms implemented were unable to find the significant relationships. Most of the search algorithms employed were able to find the significant relationship we had specified and performed well within the limited capabilities of the GraphDB workbench. We also found that the uninformed searches performed much better than the informed search.

Furthermore, we discovered that whilst the use of bidirectional searches did not improve the efficiency of the search algorithms, the spatial complexity, in this case the number of queries made to the ontology, was greatly reduced for searches with a greater number of intermediaries. As our implementation had memory issues when many queries were being sent to the ontology in quick succession, this allowed for a greater exploration of ontology and a more significant number of relationships were able to be explored without the need to pause between searches. However, for the purpose of this experiment the benefit from the reduction in queries made is largely irrelevant as we are only concerned with close relationships.

There were no legal, social, ethical, professional, or commercial issues identified as part of this study.

1 Introduction

Vast amounts of data are available on the Internet, scattered across large repositories, both public and private, within the public domain and other organisations. The issue with the disjointed nature of this information, and the Internet being predominantly a method of linking documents for human comprehension [3], is that finding meaningful relationships between these entities is difficult.

The aim of this project is to find these relationships between seemingly disjointed entities by unifying concepts from the Semantic Web, web ontologies, search methods, and tree expansion. We wish to evaluate different search methods to find the most efficient and reliable means of discovering the relationship chains between two entities with minimal degrees of separation. We wish for the relationship chain to be meaningful, such that the relationship expressed between these entities is non-trivial in that it has less degrees of separation than the average distance between any two entities within the ontology. A trivial relationship would be one that a significant percentage of objects of their class are linked by such a relation, i.e. two people being linked because they are homeowners, but their homes were not sold by the same company, on the same street or even in the same country.

We will develop an efficient search algorithm that aims to find the optimal path between two entities within the knowledge graph of a web ontology, the optimal path returning the shortest possible connection between the two entities we have selected. If our aim is to find a significant relationship between said entities, then we can use the proximity of the two entities to assert whether their relationship is atypical. A path between them that is greater than or equal to the average cannot be considered "meaningful" as it is not abnormal. Our algorithm will aim to discover the optimal path using an iterative deepening search, starting from each end of the path and using a bidirectional searching method.

It is my belief that the data stored within large open source web ontologies, such as FactForge, could be used to identify previously unknown relationships between companies, government bodies or people of public interest, without the need for a computationally complex natural language processing and machine learning. By searching these ontologies, we will be able to, where applicable, find a path from one entity to another in the graph with less intermediaries than the average, or to find the optimal path between said entities.

The Semantic Web and web ontologies allow for a formal definition of taxonomies and classification networks. Defined by classes of objects and the predicates, representing the relationship between said objects. Discovering significant relationships between these objects becomes far simpler as they are defined by their links to one, another. The challenge of finding a significant relationship between the two entities increases with the size of the ontology and the branching factor of each entity, i.e. the number of links one entity can have. A large ontology such as FactForge has over 1 billion objects contained within it[4] with an unfixed branching factor, ensuring any tree expansion of the entire tree to be far too computationally complex. Therefore, a method will need to be developed that significantly reduces the number of tree expansions to be carried out in order to find these relationships.

Considering this, it is the purpose of this project to develop an efficient algorithm to find a meaningful relationship between two objects within the knowledge graph of an ontology, through the optimal path between them. The significance of a relationship will be decided by the number of intermediaries in the path being less than the mean or mode of relationships in such an ontology. We shall analyse the performance of the search algorithm's ability to find said relationships, both in terms of time and space complexity. For the purpose of this project we shall restrict ourselves to one kind of relationship, that of a company and director, with the possibility of expanding to other relationships in further work.

2 Literature Review

There are a number of computer science concepts are central to this project, namely the Semantic Web as an expansion of the internet that provides methods of computer comprehension, web ontologies as the source of information that defines objects by their relation to another, search algorithms as a method of expanding the relationship tree expressed by ontologies and exploring the research done in finding relationship chains and the degrees of separation between two entities. These concepts are central to this project for in order to discover the relationship chains, we will need many predefined links between objects and some value that can be exploited in order to discover the commonality between two unique objects.

2.1 Search

Over the course of this project we shall be using several different search strategies, including uninformed, informed and bidirectional search techniques. Search algorithms are a method in which we can perform a sequence of actions necessary to solve a problem. We build a search tree containing a series of states and actions, represented as nodes, in which a goal state representing the solution to said problem is achieved by a sequence of these actions. The aim of a search algorithm is to find the least costly solution to the problem posed.

Uninformed searches decide which nodes to expand, and which actions to take, based solely on the structure of the search tree and not at the state inside the nodes. The decision on which path to take is solely based upon the actions previously taken. Such examples that will be utilised by this project are Breadth-First and Depth-First search, each one expanding the shallowest and deepest nodes available respectively.

Whereas informed searches do look at the state of the nodes in order to decide which paths and actions to take to find the optimal solution. In order to make the decision on which node to expand, informed searches use and evaluation function *f(n)* as an estimate of the cost of the path from the selected node to the goal state. Such an informed search algorithm is called Greedy Best-First search, in which the algorithm expands nodes that are closest to the goal node, in an assumption that they will lead to the goal quickly. With the evaluation function being equal to a heuristic, an estimation of the distance from the current state to the goal state.

2.1.1 Recursive Best-First Search

Recursive Best-First search is an informed searching method that searches the nodes of a graph by expanding the nodes in order of the sum of each node's estimated distance to the goal node and the cost to reach said node[5]. The recursive algorithm, as seen in the appendix figure A.3, functions much the same as a greedy best-first search, but only using linear space. The recursive algorithm expands nodes in order of their estimated distance provided by the evaluation function:

$$f(n) = g(n) + h(n)$$

The algorithm continues down this path until the cost of the next node to expand reaches an *f-limit*, the estimated cost of the best alternative path. The algorithm then rewinds back to the alternative path and then proceeds down this alternative, replacing the estimated cost of each node it passes with the best estimated cost of its children. Thus, remembering the best leaf node of the rewound subtree.

In order to provide an optimal solution, the heuristic used to estimate the distance from any node to the goal node must be consistent. A heuristic is consistent if, for every node *n* and every successor *n'*, the estimated cost of reaching the goal from *n* is no greater than the step cost of getting to *n'* plus the estimated cost of reaching the goal from *n'*:

$$h(n) \leq c(n, a, n') + h(n')$$

2.1.2 Bidirectional Search

Bidirectional search executes two search algorithms simultaneously, one form the initial node and one form the goal node, with the intention to have the two searches intersect. This can be done by either having the two frontiers, the list of expandable nodes from each search, share at lest one node, or by having one frontier contain an already expanded node of the other search. As we already know the goal state we are looking for and are simply looking for the optimal path, we can use this method to increase the efficiency of our search. The complexity of bidirectional search decreases to $O(b^{d/2})$ from $O(b^d)$ when implemented using a breadth-first search, the explored area of the two searches is smaller than one large search from the start to the goal node, as can be seen in figure 2.1.

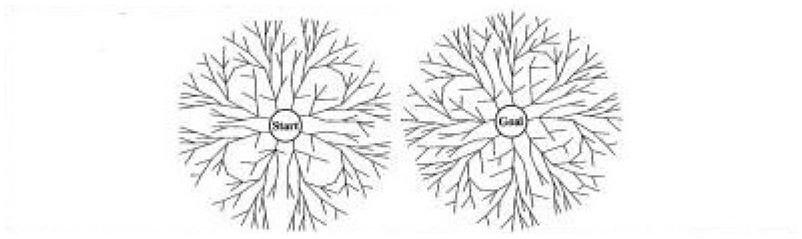


Figure 2.1: Schematic of a bidirectional search

Bidirectional search is implemented by replacing the goal test with a check if one of the frontiers intersect with another or one of the expanded nodes of the other search. However, the first intersection of these searches may not be optimal, further searches or more complex searches may be necessary. Russell and Norvig state that the biggest draw back of bidirectional search is its space complexity, however this can be reduced by half if one of the two searches uses an iterative deepening search, with at least one of the frontiers must be kept in memory. If one of the frontiers are kept in memory, the other can be searched backwards, finding the predecessors of the currently expanded node which can be done in constant time and as such does not drastically increase the time complexity of the search.

2.2 Relationship Chains

Discovering relationship chains between entities is not a new concept. In their paper discussing the so called "Small World Problem" [1], Milgram discovered that the mean average degrees of separation (i.e. the number of intermediaries) between any two people on the planet was five. However,

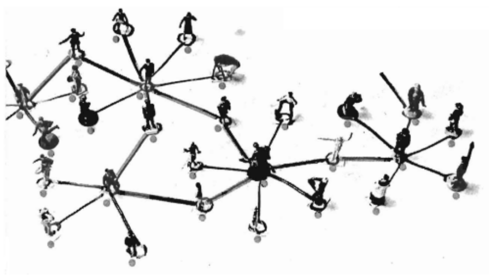


Figure 2.2: Social network depicted by Milgram

Milgram does state that several the chains performed in their experiment did not complete and were abandoned prior to completion, to account for this a mathematical model was performed that did yield slightly longer chains on average, the exact length of these modelled chains is not stated. Whilst the experiments performed by Milgram were solely focused on the individual's social network, i.e. the person's personal acquaintances, this could also be compounded by other information that links the two people as it does not take into account why the person and their acquaintance know each other, possibly through work, geographical proximity, or a school friend. This abstraction could be provided by ontologies that have a greater variety of relational links and could yield new relationship chains.

Further work conducted by Facebook in 2012 discovered that social networks, and possibly any knowledge graph of a sufficient size, tend to have only four degrees of separation [2]. This paper highlights the fact that as the size of the knowledge graph increases, the fewer degrees of separation there is between two people. This logic can be applied to knowledge graphs other than social networks; we expect that this will also hold true for ontologies as they can also be represented as knowledge graphs. Backstrom et al. concludes that networks that are primarily defined by their geographical proximity, have a smaller average distance, such is the case in Milgram's experiment.

In their paper, Milgram discusses the tendency for common pathways for any relationship chain, such as the clothing merchant in Boston being the principal point of contact between the broker and the larger world [1]. By our definition of a "meaningful relationship" this will be considered a meaningful relationship, provided that the number of intermediaries is less than average, perhaps further work could be done to avoid said connections.

Conversely, as hypothesised by Daraghmi et al., it is possible that these points of commonality may be what causes the decrease in intermediaries for a network [6]. Daraghmi et al. discovered, through the use of graph theory, that the number of intermediaries does tend to plateau with the increase in the size of the network and it may be the points of commonality that unify the "worlds" that link two people together as discussed in Milgram [1]. However, as the experiments performed by Daraghmi et al. were only performed on a small percentage of the knowledge graph used by Backstrom et al., this may be due to the limited scope of the network and its geographic bounds.

Whilst a large percentage of the research done on relationship chains has predominantly been focused on social networks, it is my belief that the increase in diversity of relations offered by ontologies may overcome the common pathways issue found in Milgram and may yield fewer intermediaries necessary than the average four-to-five found in social networks.

2.3 Web Ontologies

The Semantic Web is an extension of the World Wide Web designed to be computer understandable by enabling comprehension of the content and context of the web page [3]. The current function of the World Wide Web is predominantly a medium for the linking of a number of parsable document, the documents can be parsed in an effort to search for certain information but understanding the context is not possible without the use of machine learning or some form of natural language processing, and the information contained within the document is unable to be manipulated automatically. Contrastingly, the Semantic Web is "a web of actionable information" [7], the Semantic Web enables the performance of intelligent, automatic tasks to be executed on the web by adding a series of documents and data the current Web, targeted towards computer understanding.

In their paper, Berners-Lee et al. define ontologies as the method of which the Semantic Web can implement this understanding. Ontologies are a collection of taxonomies and a set of inference rules[3]. The taxonomy defines the classes of objects and the relations between them, such that a single entry in the ontology will consist of an entity, a predicate, and some other entity (e.g. subject: "IBM's The Great Man Challenge"; predicate: "begins"; object: "April - August"). This predefined link can be exploited by the computer to find relationship chains between entities, as each entity can have multiple predicates and a defined object for each, another entity with the same object value will have a connection to it via the unique URI for said value.

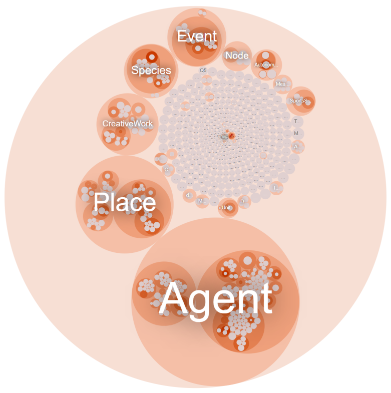


Figure 2.3: RDF graph of the FactForge linked ontology

An issue that occurs when linking ontologies, such as the data provided by FactForge [4], is that the definition of predicates may conflict and cause logical errors. In Berners-Lee et al. they describe a situation where the definitions of "address" conflicts between two intelligent agents and cause the "address" of postal addresses to be confused with the "address" physical address. Large, linked ontologies overcome this with a series of equivalence relations to better form an understanding of the context of the data, through the use of URIs.

Using these linked ontologies, we will be able to find relationship chains that were previously impossible to discover as they can be used to link data from a multitude of different domains, including but not limited to companies, management, news events and people of public interest. This information will be crucial to finding meaningful relationship chains through previously unidentifiable means, such as discovering a relationship between two competing company's CEOs through members of their family having attended the same school. Knowledge graphs, the primary way of displaying data in ontologies, can be queried by RDF query languages to find said links, as discussed in Bailey et al. [8]. Whilst the survey performed by Bailey et al. primarily focused on the wide variety of RDF and XML query languages available, and their positives and negatives, the paper highlights the ability to efficiently find an objects relationship to another uniquely defined value. The results of these queries can be used to form new ones and follow the trail of values defined by the knowledge graph. Exploiting this feature of query languages, we will be able to discover the relationship chains via uses of both uninformed and informed search algorithms, with each step of the tree expansion being performed by a new query on one of the results given.

3 Motivation and Problem Analysis

3.1 Motivation

The aim of this project is to find meaningful relationships between two companies, currently through the directors of said companies but the medium in which these links can be found could vary. These meaningful links could be used to identify and examine different companies for a high level of co-integration and also to find conflicts of interest in the board members of said companies. Companies with a high level of co-integration, that is to say their stock prices share a long-term equilibrium and have a relatively stable relationship, as the stock prices for one company increases, the stock prices of the second follow. This can be exploited by a trading method called pairs trading, in which stocks are bought and sold simultaneously from the two companies and exploiting the price difference to return a profit.

The relationships discovered by this project will aim to inform which companies should be considered for pairs trading as the close links they share could be indicative of the companies level of co-integration.

3.1.1 Volkswagen and Audi: Emissions testing

In 2015 the car manufacturer Volkswagen were found to be artificially reducing the level of emissions in their cars when under testing conditions, in an attempt to increase sales by emphasising their car's low emissions[9], when in reality the level of emissions produced by the cars was significantly higher. When this was discovered Volkswagen share prices dropped[10].

Similarly, at this time the price of Audi's stocks also began to fall. Audi is a subsidiary of the Volkswagen Group and the two share a number of board members and directors, such as Abraham Schot who is both the Chairman of the Board of Management of AUDI AG and sits on the Board of Management of Volkswagen Commercial Vehicles[11][12]. It is possible that this close relationship could have contributed to the similar movements of the two companies stock prices. This link could have possibly allowed someone to use the fall in one to predict the fall in the other.

3.1.2 Virtual Monopolies

A virtual monopoly is when a limited number of companies exist within an advantageous relationship, wherein they can work together, or at least not need to compete, to control a commodity. Without the need of competition between these few companies, they are able to price gouge the consumer without recompense or the fear of losing customers. When all the companies supplying the commodity agree to set their prices at a much higher level than would be considered fair, or would be possible in a free market, the consumer has no option but to foot the bill.

A current example of this is the American telecommunications market, which is currently dominated by three large network providers, AT&T, Verizon and T-Mobile, which combined have an estimated 428 million active subscribers[13][14][15][16], this represents an overwhelming majority of the consumer base, and in many areas it is likely that a customer only has one option for a network provider as their coverage is not identical.

If such a close relationship exists between companies that control a large portion of a commodity, they are then incentivised not to compete, but instead form a virtual monopoly. One could reasonably assert that companies in a virtual monopoly are viable for pairs trading or should be considered for possible malpractice investigations.

3.2 Problem Analysis

Having considered that significant links between companies exist, either via their directors or by other means, there may also exist a relationship between each company's stock prices and vice versa. This leads to the reasoning that if it appears that two companies seem to have some level of co-integration and could be considered for pairs trading, discovering and understanding the link between said companies will better inform one on the likelihood that the current equilibrium between the two companies is stable, as long as this relationship holds. Once an efficient algorithm has been found that can search ontologies for this kind of relationship it can then be implemented on a wider scale to include different kinds of connections, such as directors being connected via their shared Alma Mater rather than the shared intermediate directors of other companies.

We suggest that companies that are considered for pairs trading, series of companies that form a relative monopoly, and company directors with a possible conflict of interest have a significant relationship in the personnel that they share, and that these relationships can be found in publicly accessible Web Ontologies. The scope and scale of finding these relationships will grow exponentially with the size of the ontology and the number of possible relationships between entities. Therefore it is important to develop a search algorithm that will be able to scale with this increase.

The closer related two companies are, the greater the impact of this relationship and the more significant the effect it can have. That is why we will focus on trying to find the most efficient algorithm for finding the shortest path between any two entities, highlighting the relationships that are uncommon (i.e. have a path with fewer than average number of intermediaries). These meaningful relationships could be useful for identifying companies for pairs trading or discovering virtual monopolies.

Whilst we do not intend to examine if these relationships are of any use for identifying companies for pairs trading, if our experiment provides an efficient and worthwhile algorithm than perhaps one could explore its use for such in further study.

3.3 Hypothesis

As the algorithm we develop will need to be scalable to large ontologies and in use with a wide range of relationship types, to be computationally efficient the algorithm used will need to have low time and space complexity. Therefore, we hypothesise that the most computationally efficient method for finding the shortest path between any two entities in an ontology, through a specified relationship, is iterative deepening search. We also hypothesise that bidirectional searching in large ontologies have a lower time and space complexity than single searching methods, making them overall more computationally efficient.

4 Method

4.1 Method Overview

To test our hypothesis and assess the performance of different search algorithms, we require a sample ontology upon which to run our search algorithms. For our purposes a relatively small ontology with only the company-director relationship we are considering is needed. We shall retrieve a set of publicly traded companies and their directors from publicly available company reports. The ontology will act as our search graph, in which we can expand any company or director to find their coworkers and employees which shall be treated as the current directors children.

In order to test our hypothesis, we will execute each search algorithm individually on a set of predetermined relationships between individuals in the ontology of differing lengths. The relationships in question shall be incremental in size but no longer than 6 intermediaries as we are only interested in the discovery of meaningful/close relationships that could be used to indicate the presence of virtual monopolies or high co-integration. Once this has been completed we will again run the search algorithms in conjunction with another as a bidirectional search. The searches will be assessed by their time complexity, the space complexity and their reliability.

After the performance of the searches have been gathered, we will compare the performances of the individual and bidirectional searches separately, to assert which method is most efficient in the average case. We can then compare the performance of single searching to bidirectional searching methods, to assert if bidirectional search is more efficient for finding the optimal path in large ontologies.

4.2 Full Method

4.2.1 Gathering Company Information

In order to obtain the relevant information for this experiment I have retrieved a set of company reports in the US, using a publicly available Statement of Changes in Beneficial Ownership document from 2017. This document details any material change in the holdings of company insiders and is available in XML format. Using a series of these documents from Q1 of 2017, an NT file is constructed consisting of a series of triples mapping a unique identifier for said company and the information in the Statement of Change document. The NT files are then imported into a GraphDB repository that can then be queried.

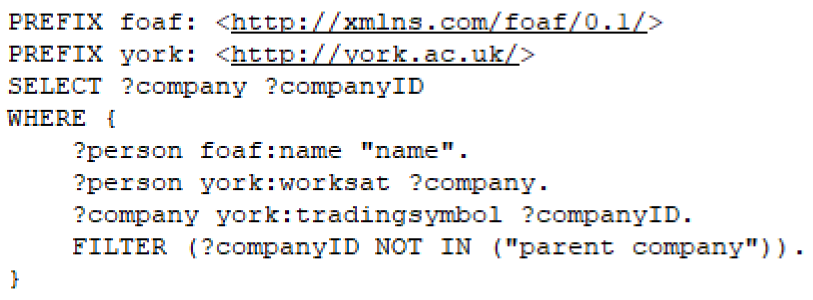
4.2.2 Searching the Ontology

In order to search the ontology for new relationships multiple search algorithms are used, the functionality of which are well established and require no alterations to their processes. Iterative deepening, Breadth-First, and recursive Best-First Search has been employed to discover the relationships in the ontology. When taking a personnel-centred approach to this experiment, that is to say we are attempting to find a relationship between two people by their shared companies, each company insider in the ontology is equivalent to a node in the search graph, with the node's children being the other employees of any company the root node is employed by. The root node for any company in the ontology becomes the first node encountered to have a link to it, meaning that the search graph is not fixed and will change each time a search is performed.

The nodes in the search graph are expanded by querying the ontology, first discovering any and all companies associated with the current node, and then executing a second query to retrieve all employees of the companies found by the first query, excluding the root node. As the ontology will could contain any number of cycles, in order to ensure the search algorithms do not repeat and get stuck in a cycle, the results of the first query are filtered for any companies that have been previously visited before being passed on to the second query. The queries for node expansion are seen in Figures 4.2 and 4.1. As a SPARQL query supplied to GraphDB has a fixed memory capacity, the filtering of companies that employ a node is done after the query is returned, instead of the ontology itself filtering the output.

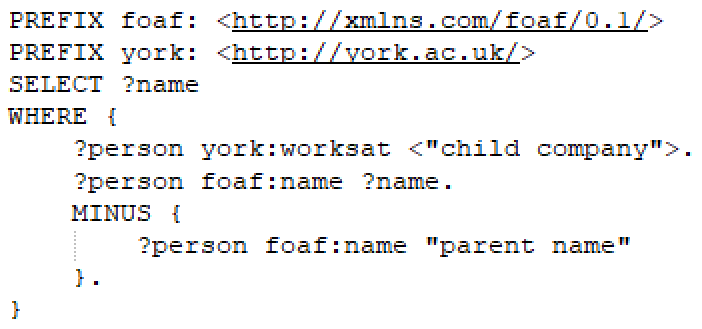
Best-First Search Heuristic

The use of a recursive Best-First Search algorithm requires the use of a heuristic. For the algorithm to find the optimal path said heuristic must be admissible. For use in this experiment an admissible heuristic is difficult to obtain, as we wish to employ this search algorithm in order to discover a path between two entities, we do not know the actual distance to the goal node. Therefore, our heuristic is only an estimate and the use of a



```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX york: <http://york.ac.uk/>
SELECT ?company ?companyID
WHERE {
    ?person foaf:name "name".
    ?person york:worksat ?company.
    ?company york:tradingsymbol ?companyID.
    FILTER (?companyID NOT IN ("parent company")).
}
```

Figure 4.1: Query template for retrieving the companies of a selected director



```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX york: <http://york.ac.uk/>
SELECT ?name
WHERE {
    ?person york:worksat <"child company">.
    ?person foaf:name ?name.
    MINUS {
        ?person foaf:name "parent name"
    }.
}
```

Figure 4.2: Query template for retrieving the directors at a selected company

Best-First Search may not result in the optimal path.

The heuristic used in our Best-First Search, as seen in Figure4.3, is based upon a node's connectivity and the current depth of the search. This is because we can reasonably assume that the deeper a node is in the search, the less likely it will lead to the optimal path. Furthermore, we can reasonably assume that the more connected a node is the greater the chance it will lead to the optimal path. As a result our heuristic will favour the shallower, more connected nodes before expanding the deeper nodes by having the connections factor decrease the heuristic and the depth factor increase it.

$$DepthFactor = \begin{cases} 1 - depth/7 & depth <= 7 \\ 0 & depth > 7 \end{cases}$$

$$ConnectionFactor = \frac{connections}{maxbranchingfactor}$$

$$h(n) = 1 - 0.6 * ConnectorFactor + 0.4 * DepthFactor$$

Figure 4.3: Best-First Search Heuristic

4.2.3 Bidirectional Search

In order to perform a bidirectional search upon the ontology that will still result in the optimal path being found, I will be running the two searches concurrently, using multi-threading techniques. Each search will have to request access to the ontology, with the ontology port being accessed using critical sections of code. The ontology is used as a shared variable as the two searches will be running asynchronously. The ontology is hosted on a local repository and is accessed via a single port, access to the port will be given during these critical sections as we cannot expect each search to be requiring access to the ontology only when the other is comparing nodes to the goal state.

To implement the bidirectional search, we will have one of the searches maintaining a fringe of visited nodes, their attached companies, and their parent node for use when reconstructing the path when the two searches intersect. This will be implemented in our python code using a dictionary, so that the indexing of the fringe will remain constant as it grows. As stated in Russel and Norvig[5], the space requirement of bidirectional search is its greatest weakness, to combat this we will only be maintaining a repository of the visited nodes of one of the two searches. Therefore, the goal test of assessing whether the two searches have intersected can only be completed by the search that is not maintaining the fringe, this may cause a slight reduction in the execution time however we do not consider it to be a significant factor in the performance of the search.

As stated in the above literature review, when a bidirectional search is implemented, often the first path discovered is not the optimal path. When a path is discovered, the two halves will be generated by connecting the mid-node to its parents held in the fringe, before signalling to the current search that the goal state has been found, allowing it to traverse backwards from the mid-node back up the search, adding to the path as it goes. To ensure our bidirectional algorithms do indeed retrieve the optimal path, some extra searching is required. Once the first path has been found, it will be stored in a global list and the search will continue until three paths have been found, the shortest of which is likely to be the optimal path. We can assume this due to experimental research upon this ontology and because we are searching for relatively small connections, i.e. less than 6 intermediaries, continuing the searches further is likely to result in one of them finding the optimal path, making the use of bidirectional search redundant.

4.2.4 Assessing Performance

The performance of each search will be assessed by two metrics, the average execution time on paths of differing lengths and the amount of memory used. The average execution time of each search will be calculated from twenty five executions of the algorithm over the same search. As we cannot reasonably assess the worst-case performance of each search, due to the formation of the search graph changing depending on the order of returned results from the ontology queries, we will have to assess the performance from the average case.

As the amount of space used by each of these searches is well defined by their theoretical limits, will be comparing the space complexity of the searches through the number of queries made to the ontology. We are only assessing the number of queries made as the amount of memory that is being used is inconsequential in practical terms, as the only data being stored is a series of strings and any practical application of this method will have more than enough memory to be able to perform these searches. The size of the fringe is being recorded so that we can interpret from the results how quickly each of the bidirectional searches were intersecting. This metric can be used to inform us further about which configurations of searches will be most effective on large ontologies, i.e. is it better to have a much broader searching algorithm maintaining the fringe or a much deeper one?

Once the average execution time of the searches has been calculated and the number of queries made recorded, we can perform a statistical test to decide if we can assert which of the searches perform best when searching ontologies. We can then compare whether the average execution of a bidirectional search performs better than that of a regular search.

5 Results

5.1 Querying the Ontology

The system used to generate the search graph by querying the ontology was very effective and was able to expand the nodes efficiently and repeatedly without issue. However, there were some issues with using the SPARQLWrapper python library and the data we had coalesced for this project. The queries are supplied to GraphDB in a string format and converted into UTF-8 format for transfer to the local repository, this proved to cause issues when the data had not been properly sanitised and the query filters exceeded a certain length, forcing the query processing to be done outside the ontology. Several the names of companies included in our ontology included a backslash in their names, in normal string operations the backslash is used to identify a real character, i.e. \n is the newline character, in order to represent a backslash in a string two need to be placed together (\\). However, when the query is supplied to the SPARQLWrapper, to represent the backslash character an additional backslash is added to each instance, this would result with the final query being sent to GraphDB having too many backslashes and result in a bad query error. Table A.1 found in the appendix give examples of the pre-sanitised data.

To overcome this issue all the ontology data had to be sanitised before a search could be performed. This is obviously an issue when using the algorithm we have developed on a significantly larger repository, the data held in larger repositories such as FactForge or DBpedia is simply too substantial to ensure that all the data has been sanitised. In order to overcome this the queries shown in figures 4.1 & 4.2 will have to be modified to use the subject's unique identifier and the search graph explored by the search algorithms will have to compare these unique identifiers rather than the names of directors. Whilst this is rather simple to implement it does reduce the understandability of the algorithm for testing and evaluation sake, but once the operation of a searching algorithm has been tested and verified this should not be a problem.

Furthermore, the GraphDB workbench is implemented using the Eclipse RDF4J Framework. This can cause issues when running many queries as the Java garbage collection is rather inefficient and the workbench can quickly run out of memory if several searches are performed in quick succession. We found under stress testing that the workbench would use up all its memory after approximately 1000 searches, with an approximate depth of 6 intermediaries for the optimal path, if not left time between searches for garbage collection.

5.2 Data Analysis

The following results were produced on a Windows 10 operating system, using an Intel Core i7-6700k processor, GTX 980 Ti graphics card, and 16GB of RAM. The results approximate their real-time average execution as they will be subject to the threading and scheduling algorithms implemented by the OS and if the algorithms used were run on a real-time system the searches' average execution will greatly decrease. However, the interference caused is constant across each search making the impact on the searches for comparison sake is negligible.

The following results are calculated from the average execution time of each search on the same start and goal nodes, with a new execution of the GraphDB workbench to ensure an empty Java memory heap and no search ends prematurely.

5.2.1 Single Search Performance

Having discovered a series of relationships on which to test the different search algorithms, an average of 25 searches on different relationship lengths was calculated which can be seen in table 5.1 and plotted in figure 5.1.

	Average Execution Time		
Number of Intermediaries	Iterative Deepening Search	Breadth-First Search	Best-First Search
0	0.013124943	0.005665503	0.21287470
1	0.095068092	0.067301645	3.92226705
2	0.218879709	0.140367708	N/A
3	0.606253357	0.406639977	N/A
4	2.123347416	1.519327297	N/A

Table 5.1: Single Search Average Execution Time

From this experimental data we can clearly see that the Recursive Best-First Search algorithm performed the most poorly, often being unable to find the optimal path. The Best-First Search algorithm struggled to find the

	Number of Queries Made		
Number of Intermediaries	Iterative Deepening Search	Breadth-First Search	Best-First Search
0	2	2	136
1	28	26	46130
2	48	40	Memory Error
3	214	168	Memory Error
4	966	752	Memory Error

Table 5.2: Single Search No. Queries Made

optimal path at a depth greater than 2-3 nodes before causing GraphDB to experience a memory error due to the Java heap running out of memory locations. At a depth of less than 3 nodes, the Best-First Search performed the worst of the three search algorithms, by a factor of 20-42 times greater. As can be seen in table 5.2, the number of queries made to the ontology by Best-First search increases dramatically, far larger than that of the other two searches. This is in part due to the heuristic needing to further query the ontology each time a node is evaluated.

Furthermore, we can see that overall the Breadth-First search performed better on average than the other two search algorithms across all search depths and consistently found the optimal path. This is consistent with the theoretical complexity of the algorithm O(b^d), with b equal to the branching factor and d equal to the depth of the optimal path. As Iterative Deepening Search shares the same theoretical complexity, we could have reasonably assumed that the Iterative Deepening algorithms would have performed as well as Breadth-First. However, this was not the case.

The Iterative Deepening algorithm performed comparatively to the Breadth-First search at depths less than 5 intermediaries deep, growing at a constant rate consistent with Breadth-First Search as can be seen in figure 5.1. However, as the depth of the search increased the two searches diverged, with the Iterative-Deepening growing faster than the Breadth-First search. This may be in part due to the optimisation of recursive algorithms in python or the number of queries being sent to the ontology. As the two searches grew at the same rate in shallower searches, I am inclined to believe the former to be the case.

Whilst the Average execution time of the Iterative and Breadth-First searches are comparable, as can be seen in table 5.2 the Breadth-First search requires far less queries to the ontology than the Iterative-Deepening search. This is due to the iterative nature of Iterative-Deepening search causing the algorithm to re-track parts of the search tree that had been explored with a lower depth limit.

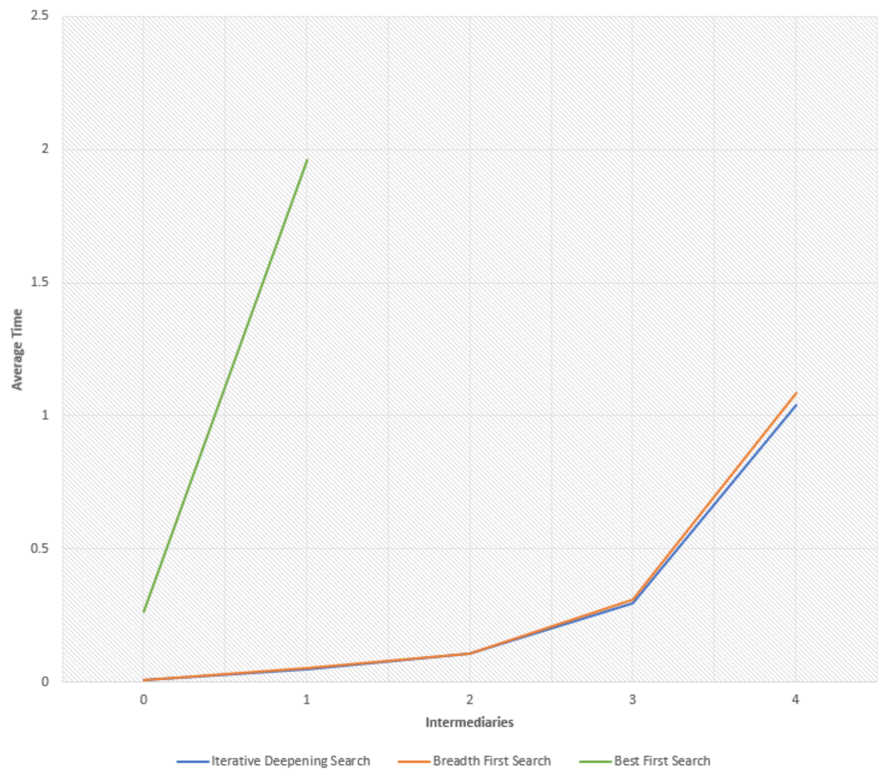


Figure 5.1: Single search average execution time

5.2.2 Bidirectional Search Performance

As can be seen from the experimental results shown in tables 5.3, 5.4, and 5.5, as with the single search method the worst performing by far was the Best-First Search algorithm. Often when executing the Best-First search the GraphDB workbench would use the entirety of its heap memory before the optimal path was found, causing the workbench to stall and be unable to receive any other queries.

	Average Execution Time		
Number of Intermediaries	IDS with IDS	IDS with Breadth-First	IDS with Best-First
0	0.026729259	0.010093031	0.127621355
1	0.153034115	0.063232279	0.255523462
2	0.507517490	0.223147745	0.487308111
3	1.176082439	0.596895828	1.253930702
4	1.820275364	1.533151026	N/A

Table 5.3: Iterative Deepening Bidirectional Search

As seen in tables 5.6, 5.7, and 5.8, the number of queries made by any bidirectional search with Recursive-Best-First makes significantly more

	Average Execution Time		
Number of Intermediaries	Breadth-First with IDS	Breadth-First with Breadth-First	Breadth-First with Best-First
0	0.049250078	0.011329718	0.173141212
1	0.173521824	0.092634716	0.366607685
2	0.384407825	0.288314905	0.646882439
3	1.396727877	0.389624777	1.251060438
4	1.696772213	0.893727493	N/A

Table 5.4: Breadth-First Bidirectional Search

	Average Execution Time		
Number of Intermediaries	Best-First with IDS	Best-First with Breadth-First	Best-First with Best-First
0	0.264057961	0.346242619	0.208945589
1	0.631984472	0.487427750	0.542858791
2	1.330609179	0.724478436	N/A
3	1.725836983	N/A	N/A
4	2.945140772	N/A	N/A

Table 5.5: Best-First Bidirectional Search

queries to the ontology than the others, and many more than any of the single search methods. This does not come about because the Best-First search is expanding more nodes in the search graph than the other algorithms, rather this is caused by the Best-First Search revisiting nodes and querying the ontology each time. When all the children of a node in the search graph have a greater estimated cost than the alternative, the recursive function backtracks to the parent node and visits another path. However, when this happens repeatedly and there is the need to backtrack from multiple depths, the number of visited nodes greatly increase, each time the children are revisited, and queries sent to the ontology.

Between the Breadth-First and Iterative Deepening bidirectional searches, the two performed comparably with the Iterative Deepening bidirectional searches performing marginally better than their Breadth-First search counterparts. The Iterative Deepening—Breadth-First algorithm, with the iterative deepening search maintaining the fringe, performed the best at low depth searches and performing better than its single search counterpart. However, it still suffered from python's unoptimized recursion functionality, as can be seen in figure 5.2 from the accelerating growth in execution time.

It appears that the greatest problem facing the iterative deepening search is the recursion optimisation. Even with the added drawback of continuing the search to find multiple paths, and twice the level of recursion overhead, the iterative deepening bidirectional search was still able to outperform its single search counterpart.

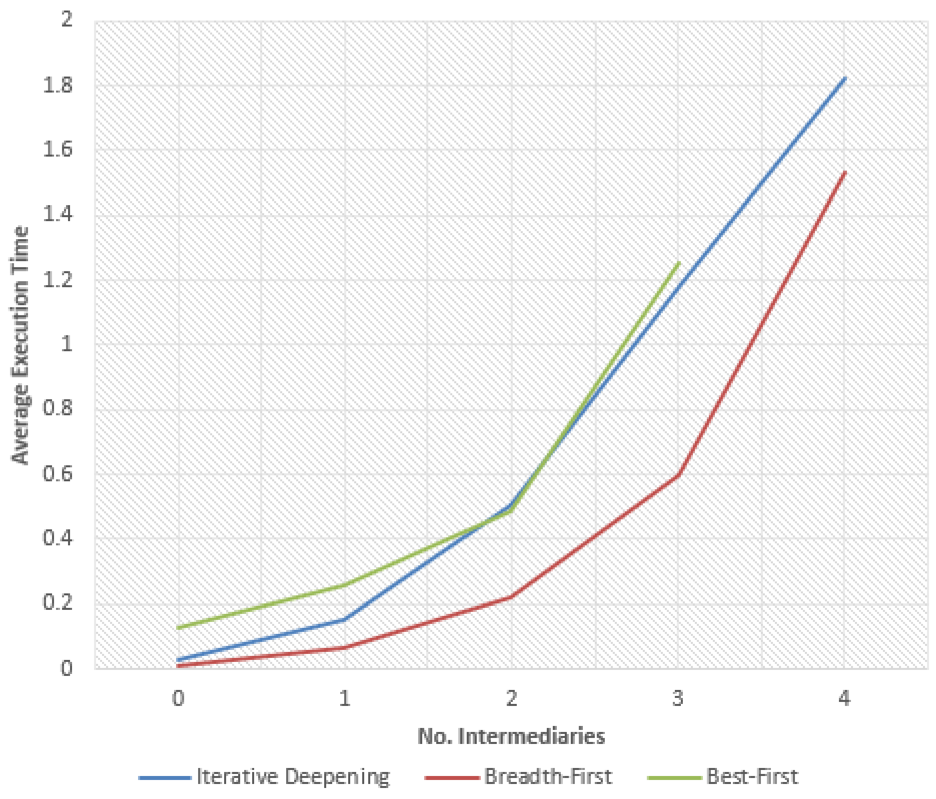


Figure 5.2: Iterative Deepening Bidirectional Search Performance

However, whilst the dual Breadth-First search algorithm performed worse than the Iterative deepening—Breadth-First algorithm in shallower searches, its execution time increased at a much slower rate than, like its single search counterpart. The dual Breadth-First algorithm was also able to outperform its single search counterpart at these greater depths. This can also be seen by the number of queries made by the dual Breadth-First search, at greater depths the number of queries made dramatically falls and remains lower than any of the other bidirectional searches.

	Number of Queries Made		
Number of Intermediaries	IDS with IDS	IDS with Breadth-First	IDS with Best-First
0	4	4	82
1	44	28	86
2	170	106	148
3	354	222	472
4	366	586	Memory Error

Table 5.6: Iterative Deepening Bidirectional Queries Made

	Number of Queries Made		
Number of Intermediaries	Breadth-First with IDS	Breadth-First with Breadth-First	Breadth-First with Best-First
0	4	4	82
1	30	42	96
2	56	116	156
3	426	52	328
4	648	322	Memory Error

Table 5.7: Breadth-First Bidirectional Queries Made

	Number of Queries Made		
Number of Intermediaries	Best-First with IDS	Best-First with Breadth-First	Best-First with Best-First
0	140	138	112
1	264	48	196
2	340	108	Memory Error
3	182	Memory Error	Memory Error
4	1318	Memory Error	Memory Error

Table 5.8: Best-First Bidirectional Queries Made

6 Evaluation

6.1 Searching the Ontology

Throughout the course of this experiment we have had a great issue caused by the GraphDB workbench. The poor java garbage collection meant that with many queries being passed to the ontology the workbench was quickly overwhelmed, causing us to pause for a few seconds between each execution of a search. Whilst this was not a problem during our searches, we feel that deeper searches of an ontology with a greater branching factor, i.e. one with more relationship types than company-employee, this will be a much more significant issue.

Furthermore, we found that the GraphDB workbench was poorly optimised and when supplied with an increase in query traffic, the workbench would be unable to resolve a query before the next was supplied to it, causing a “WinError 10048” exception to be raised (only one usage of a socket address at a time). This combined with the java garbage collection issue, means that it is our recommendation that a different graph database be used when implementing our algorithms in a practical sense.

However, from an experimental perspective, searching an ontology is relatively easy and only takes slight modification to the code implemented for the expansion of a node within the tree. Furthermore, querying the ontology allows us to perform searches on specific types of relationships, with a more detailed ontology that is not constrained to only the company-director relationship new links and relationships can be discovered that would perhaps be of more interest for the purposes of pair trading. A more detailed ontology could not only link the directors through the companies that they currently work at, but also by companies that they have previously been employed by, however this may require some additional checks to ensure they worked at said company at the same time.

Overall, I believe that we can assert that searching an ontology for these significant relationships is viable and the algorithms do perform efficiently and reliably, even with the extra constraints posed by using the GraphDB workbench. Furthermore, the viability of our algorithms extends beyond our simple ontology and could very easily be adapted to a significantly larger ontology.

6.2 Single Searches

As seen from the experimental data in table 5.1, the Best-First algorithm was not suitable for finding the optimal path between two entities. This is in part due to the heuristic being far too simplistic and not being admissible or consistent. In order for an informed search method to find the optimal path in a graph search, a heuristic needs to ensure that for every node n and every successor generated, the estimated cost of reaching the goal from n is no greater than the step cost to the successor and from the successor to the goal node[5]. Our heuristic is not consistent. As the search graph changes at the beginning of every search learning the heuristic from experience becomes almost impossible, but with enough examples and an appropriately trained neural net an effective heuristic could be found. However, this will not be transferable to a lager, more general ontology and does not necessarily provide an admissible or consistent heuristic, so we may in fact still have the issue of being unable to find the optimal path.

The iterative deepening and breadth-first search algorithms performed well and consistently were able to find the optimal path, However, each did have its problems. The primary issue with the iterative deepening search was that as the depth, and thus the level of recursion, increased so did the overhead. As Breadth-First and iterative deepening search have the same theoretical complexity, we could have expected for their average execution to remain comparable, yet the two begin to diverge at a depth greater than three intermediaries. Whilst this divergence in a practical sense of searching for these tightly connected relationships is still well within the bounds for efficiency required, it may become an issue when applied to a large enough ontology. To overcome this issue the algorithm we have developed could be performed in a programming language that performs tail recursion and remove the overhead.

Furthermore, the Iterative-Deepening search requires many more queries to be sent to the ontology due to its iterative nature in its execution. With the memory issues experienced when using the GraphDB workbench this could prove to be a significant issue. However, in a practical sense this may not be experienced when the ontology is stored locally in a system with sufficient memory allocated. Furthermore, if the system was optimised such that the algorithm does not need to query the ontology once a node has been expanded, this memory issue would decrease to the point of Iterative-Deepening surpassing Breadth-First and requiring less queries.

6.3 Bidirectional Search

The implementation of the bidirectional searches proved to not have a dramatic increase in the execution time of the search algorithms. The bidirectional searches predominantly performed worse than their single search counterparts, with the exception of Best-First search, however this is likely due to the paired search finding the optimal path rather than an improvement in the performance of the Best-First search. The recursion overhead issue suffered by the iterative deepening did not increase when running the bidirectional search. I believe this is due to the iterative search that was managing the fringe to be stopped prematurely and not require the recursion back to the top level to be executed.

Overall the main benefit from the bidirectional search was the theoretical reduced memory requirement. Whilst maintaining the fringe did increase the memory requirement of the algorithm, this is dramatically outweighed by the reduction in the number of queries made to the ontology in deeper searches. The theoretical number of nodes expanded in the singular searches were $O(b^d)$, where b is the branching factor and d is the depth of the goal node, and the number of nodes in the bidirectional search is $O(b^{d/2})$. However, due to the bidirectional search requiring to further its search beyond the first path found, for most of the searches in this experiment this was not the case. The dual breadth-first search however, did still result in a lower number of queries being sent to the ontology. The reduction in queries made also allowed the Best-First bidirectional searches to overcame the java memory heap issue and allowed more searches to be executed. Therefore the main benefit from using the bidirectional search is the reduced memory load when executing searches with a high number of intermediaries, rather than any improvement in execution time.

7 Conclusion and Further Work

Over the course of this experiment we have been able to develop a hypothesis and devise a method for testing it. From our results we can reject our hypothesis that iterative deepening search is the most effective single-search algorithm to find the optimal path between two entities through a specified relationship in an ontology. This is due to the iterative deepening search performing worse than the breadth-first search algorithm, caused by the increase in overhead with the greater depth of the optimal path.

Furthermore, we are not able to reject our null hypothesis that there is no difference in time complexity between a bidirectional and single-search algorithm, there are some small benefits produced in terms of optimisation to be gained but not significant enough to warrant rejecting the null hypothesis. This is because there is not a large enough statistical difference between the average performance of the best performing single-search and bidirectional searches to warrant rejecting the null hypothesis.

Additionally, we found that the number of queries made to the ontology, and thus the number of expanded nodes in our search, largely increased when implementing a bidirectional search fro searches with few intermediaries. This meant that the main theoretical benefit from implementing a bidirectional search was largely of no benefit for our needs, as this report is largely only considering meaningful relationships, i.e. those with few intermediaries.

Whilst we were unable to assert the best search algorithm to perform on an ontology, the project has provided a few efficient and reliable search algorithms for exploring an ontology. The algorithms and queries developed result in a reliable method to, in order to find companies for pairs trading, or find the companies a few people have influence over, or to discover any virtual monopolies over a certain commodity, find significant relationships between two companies via their directors.

In future work it may be possible to extend the algorithms we have developed to find these significant relationships through other types of relationships, not simply the company-director relation. An extension to this process may be to also look at influential company director's family or other places of business to find new relationships that otherwise may have been very distant or non-existent. Furthermore, now that we have found a significant relationship, the project could be furthered to investigate which kind of relationships lead to good candidates for pairs trading.

Further work could be conducted to develop a suitable heuristic for a more detailed ontology, one that encourages relation chains to entities belonging to concepts that are more closely related to that of the target individual, e.g. going from Apple to the place of business if the target entity in the chain is an architect, rather than following the chain of company directors of unrelated fields to the target.

A Appendix

A.1 Search Algorithm Pseudocode

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node $\leftarrow$ a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier $\leftarrow$ a FIFO queue with node as the only element
  explored $\leftarrow$ an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node $\leftarrow$ POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child $\leftarrow$ CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier $\leftarrow$ INSERT(child, frontier)
```

Figure A.1: Breadth-First Search pseudo code

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? $\leftarrow$ false
      for each action in problem.ACTIONS(node.STATE) do
          child $\leftarrow$ CHILD-NODE(problem, node, action)
          result $\leftarrow$ RECURSIVE-DLS(child, problem, limit $-$ 1)
          if result = cutoff then cutoff_occurred? $\leftarrow$ true
          else if result $\neq$ failure then return result
      if cutoff_occurred? then return cutoff else return failure
```

Figure A.2: Iterative Deepening Search pseudo code

```
function RECURSIVE-BEST-FIRST-SEARCH($problem$) returns a solution, or failure
    return RBFS($problem$, MAKE-NODE($problem$.INITIAL-STATE), $\infty$)

function RBFS($problem$, $node$, $f\_limit$) returns a solution, or failure and a new $f$-cost limit
  if $problem$.GOAL-TEST($node$.STATE) then return SOLUTION($node$)
  $successors \leftarrow [\,]$
  for each $action$ in $problem$.ACTIONS($node$.STATE) do
      add CHILD-NODE($problem$, $node$, $action$) into $successors$
  if $successors$ is empty then return $failure$, $\infty$
  for each $s$ in $successors$ do  /* update $f$ with value from previous search, if any */
      $s.f \leftarrow \max(s.g + s.h,\ node.f))$
  loop do
      $best \leftarrow$ the lowest $f$-value node in $successors$
      if $best.f > f\_limit$ then return $failure$, $best.f$
      $alternative \leftarrow$ the second-lowest $f$-value among $successors$
      $result, best.f \leftarrow$ RBFS($problem$, $best$, $\min(f\_limit, alternative)$)
      if $result \neq failure$ then return $result$
```

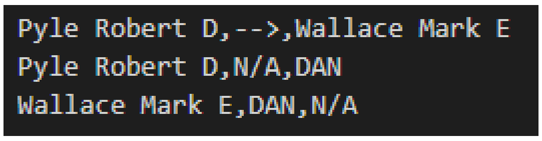
Figure A.3: Recursive Best-First Search pseudo code

A.2 Raw Ontology Data

Subject	Predicate	Object
http://sec.com/0000036104	rdf:type	http://york.ac.uk/Company
http://sec.com/0000036104	http://xmlns.com/foaf/0.1/name	US BANCORP \DE\
http://sec.com/0000036104	http://york.ac.uk/tradingsymbol	USB
http://sec.com/0001449732	rdf:type	http://xmlns.com/foaf/0.1/Company
http://sec.com/0001449732	http://xmlns.com/foaf/0.1/name	Willbros Group, Inc.\NEW\
http://sec.com/0001449732	http://york.ac.uk/tradingsymbol	WG
http://sec.com/0001012019	rdf:type	http://xmlns.com/foaf/0.1/Company
http://sec.com/0001012019	http://xmlns.com/foaf/0.1/name	RUSH ENTERPRISES INC \TX\
http://sec.com/0001012019	http://york.ac.uk/tradingsymbol	RUSH
http://sec.com/0001005284	rdf:type	http://xmlns.com/foaf/0.1/Company
http://sec.com/0001005284	http://xmlns.com/foaf/0.1/name	UNIVERSAL DISPLAY CORP \PA\
http://sec.com/0001005284	http://york.ac.uk/tradingsymbol	OLED
http://sec.com/0000897448	rdf:type	http://xmlns.com/foaf/0.1/Company
http://sec.com/0000897448	http://xmlns.com/foaf/0.1/name	AMARIN CORP PLC\UK
http://sec.com/0000897448	http://york.ac.uk/tradingsymbol	AMRN

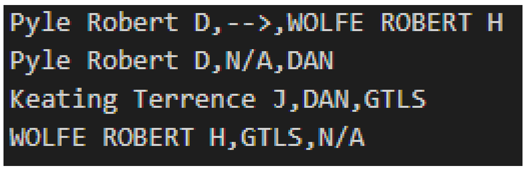
Table A.1: Example ontology entries

A.3 Trial Relationships



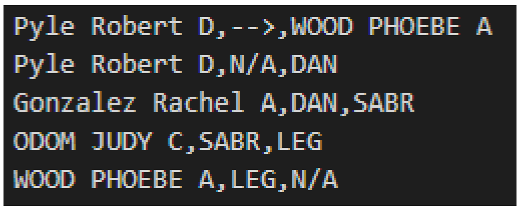
```
Pyle Robert D,-->,Wallace Mark E
Pyle Robert D,N/A,DAN
Wallace Mark E,DAN,N/A
```

Figure A.4: 1st Relationship used for testing



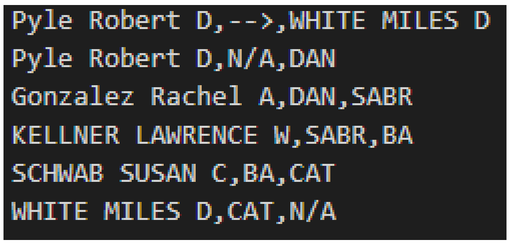
```
Pyle Robert D,-->,WOLFE ROBERT H
Pyle Robert D,N/A,DAN
Keating Terrence J,DAN,GTLS
WOLFE ROBERT H,GTLS,N/A
```

Figure A.5: 2nd Relationship used for testing



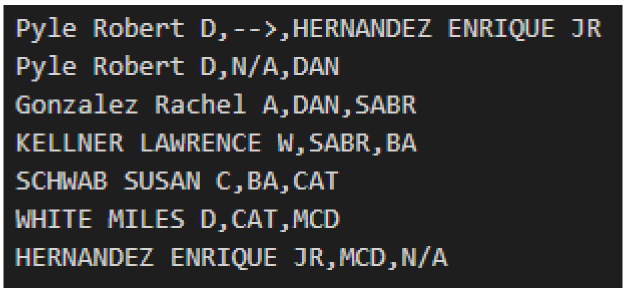
```
Pyle Robert D,-->,WOOD PHOEBE A
Pyle Robert D,N/A,DAN
Gonzalez Rachel A,DAN,SABR
ODOM JUDY C,SABR,LEG
WOOD PHOEBE A,LEG,N/A
```

Figure A.6: 3rd Relationship used for testing



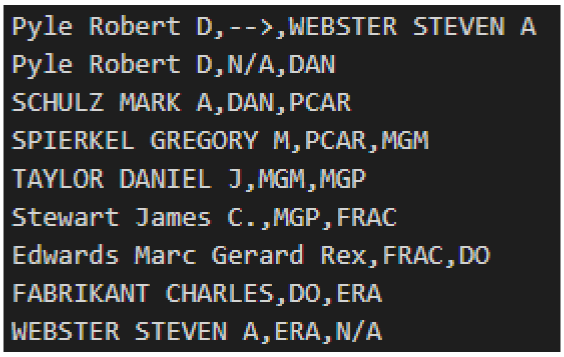
```
Pyle Robert D,-->,WHITE MILES D
Pyle Robert D,N/A,DAN
Gonzalez Rachel A,DAN,SABR
KELLNER LAWRENCE W,SABR,BA
SCHWAB SUSAN C,BA,CAT
WHITE MILES D,CAT,N/A
```

Figure A.7: 4th Relationship used for testing



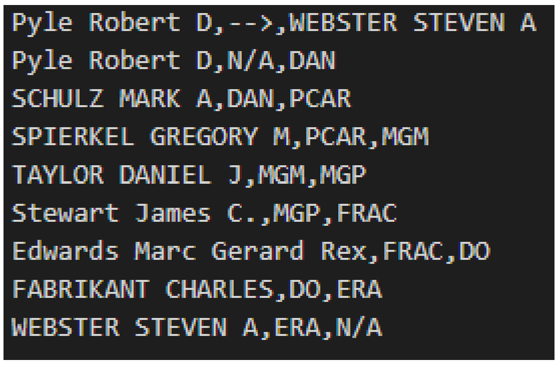
```
Pyle Robert D,-->,HERNANDEZ ENRIQUE JR
Pyle Robert D,N/A,DAN
Gonzalez Rachel A,DAN,SABR
KELLNER LAWRENCE W,SABR,BA
SCHWAB SUSAN C,BA,CAT
WHITE MILES D,CAT,MCD
HERNANDEZ ENRIQUE JR,MCD,N/A
```

Figure A.8: 5th Relationship used for testing



```
Pyle Robert D,-->,WEBSTER STEVEN A
Pyle Robert D,N/A,DAN
SCHULZ MARK A,DAN,PCAR
SPIERKEL GREGORY M,PCAR,MGM
TAYLOR DANIEL J,MGM,MGP
Stewart James C.,MGP,FRAC
Edwards Marc Gerard Rex,FRAC,DO
FABRIKANT CHARLES,DO,ERA
WEBSTER STEVEN A,ERA,N/A
```

Figure A.9: 6th Relationship used for testing



```
Pyle Robert D,-->,WEBSTER STEVEN A
Pyle Robert D,N/A,DAN
SCHULZ MARK A,DAN,PCAR
SPIERKEL GREGORY M,PCAR,MGM
TAYLOR DANIEL J,MGM,MGP
Stewart James C.,MGP,FRAC
Edwards Marc Gerard Rex,FRAC,DO
FABRIKANT CHARLES,DO,ERA
WEBSTER STEVEN A,ERA,N/A
```

Figure A.10: 7th Relationship used for testing

Bibliography

[1] S. Milgram, ‘The small world problem’, *Psychology today*, vol. 2, no. 1, pp. 60–67, 1967.

[2] L. Backstrom, P. Boldi, M. Rosa, J. Ugander and S. Vigna, ‘Four degrees of separation’, in *Proceedings of the 4th Annual ACM Web Science Conference*, 2012, pp. 33–42.

[3] T. Berners-Lee, J. Hendler and O. Lassila, ‘The semantic web’, *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[4] (2020). About, [Online]. Available: http://factforge.net/about.

[5] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 3rd ed. Malaysia; Pearson Education Limited, 2016, ch. 3.4-3.5.

[6] E. Y. Daraghmi and Y. S. Ming, ‘Using graph theory to re-verify the small world theory in an online social network word’, in *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*, 2012, pp. 407–410.

[7] N. Shadbolt, T. Berners-Lee and W. Hall, ‘The semantic web revisited’, *IEEE intelligent systems*, vol. 21, no. 3, pp. 96–101, 2006.

[8] J. Bailey, F. Bry, T. Furche and S. Schaffert, ‘Web and semantic web query languages: A survey’, in *Reasoning Web*, Springer, 2005, pp. 35–133.

[9] J. Ewing. (Jun. 2017). The volkswagen scandal - what really happened at vw, [Online]. Available: https://www.economist.com/books-and-arts/2017/06/01/what-really-happened-at-vw. [Accessed: Feb. 19, 2020].

[10] B. Snyder and S. Jones. (Sep. 2015). Here’s a timeline of volkswagen’s tanking stock price, [Online]. Available: https://fortune.com/2015/09/23/volkswagen-stock-drop/. [Accessed: Feb. 19, 2020].

[11] (Mar. 2019). Board of management of audi ag, [Online]. Available: https://www.audi-mediacenter.com/en/overview-of-audi-5702/board-of-management-of-audi-ag-5710. [Accessed: Feb. 19, 2020].

[12] (Jan. 2019). Volkswagen executive bodies - management board, [Online]. Available: https://www.volkswagenag.com/en/group/executive-bodies.html#. [Accessed: Feb. 19, 2020].

[13] A. Mobility, *2020 AT&T Earnings*, https://investors.att.com/~/media/Files/A/ATT-IR/financial-reports/quarterly-earnings/2020/Q1_2020_INVESTOR_BRIEFING_Final.pdf, Apr. 2020.

[14] (Jul. 2019). T-mobile sets more records in q2, [Online]. Available: https://www.t-mobile.com/news/t-mobile-q2-2019-earnings. [Accessed: Apr. 8, 2020].

[15] (Jan. 2020). Sprint reports fiscal year 2019 third quarter results, [Online]. Available: https://newsroom.sprint.com/sprint-reports-fiscal-year-2019-third-quarter-results.htm. [Accessed: Apr. 8, 2020].

[16] (Sep. 2019). Verizon q3 earnings report, [Online]. Available: https://verizon.com/about/file/31827/download?token#FxCcxBBC. [Accessed: Apr. 8, 2020].