Stigmergic Linking for Optimising and Reasoning Over Information Networks

Kieran Greer[1], Matthias Baumgarten[1], Maurice Mulvenna[1], Kevin Curran[2] and Chris Nugent[1]

Faculty of Computing and Engineering, University of Ulster, Northern Ireland, UK
1 School of Computing and Mathematics and Computer Science Research Institute
2 School of Computing and Intelligent Systems and Computer Science Research Institute

Abstract **-** This main contribution of this report will be to describe and evaluate a set of tests, used to measure the effectiveness of stigmergic linking with regard to query optimisation. It will be shown that the results of queries can be autonomously fed back through a network to allow it to self-optimise. The type of network that is being considered is a network of information sources, such as a 'knowledge network'. This is a construct that will self-organise distributed knowledge in a way that allows it to be efficiently retrieved and used. Key elements thereof are its lightweight, reference-based structure and its autonomous nature. The network may have permanent organisational structures based on semantics. However, the stigmergic links will add an extra dimension to the self-organisation and allow it to optimise the query process effectively. These links will in effect create temporary views that reflect the use of the system and complement any permanent organisation created through semantics. Some tests on reasoning over the stored information will also be provided and shows that higher levels of reasoning can also be obtained.

General Terms: Autonomous, Stigmergic Link, Query, Evaluation, Knowledge Network

First released in 2008.

1 Introduction

The aim of this work is to evaluate a mechanism to allow an information network to optimise itself dynamically. One objective of the work is that the network structure should remain lightweight and so the organising mechanism should also be lightweight. To this end stigmergic links are used. Stigmergy is an experience-based mechanism of learning from its environment. The context of the work is to use stigmergic links to optimise the network with regard to querying. In other words, the network will learn what nodes to link together in order to optimise the querying process. These source clusters will represent the knowledge of the users of the system and will optimise knowledge retrieval by indicating specific sources to retrieve. Thus knowledge will be entered into the system without the need for an internal knowledge-based algorithm. This is appealing because the use of the network may not be known beforehand, or may change through time and so some form of dynamic knowledge construction will be able to adapt to a changing situation. The types of network to be considered are distributed information-based networks, such as the Internet or a Mobile or Ad-hoc NETwork (MANET). A user would typically be required to query the sources to retrieve information from such a network. One example of this that has been studied in a recent project[1] is called a 'Knowledge Network'. A knowledge network is a generic structure that organises distributed knowledge of any format into a system that will allow it to be efficiently retrieved. Papers describing the initial knowledge network concepts include Mulvenna et al. [2006] or Baumgarten et al. [2006]. The rationale of the knowledge network is to act as a middle layer that connects to a multitude of sources, organises them

[1] For more information on the overall project see http://www.cascadas-project.org.

based on various concepts and finally provides well-structured, pre-organised knowledge to individual services and applications. The main features of this structure are its generic, lightweight and service-based architecture. The service-based architecture allows for services to be added that can perform any kind of functionality over the network knowledge. This is very flexible and will allow many different kinds of network to be constructed from certain base components. The knowledge network will organise itself in an autonomous manner through the results of query requests, to create temporary views that reflect the use of the system. Figure 1 is an example of a knowledge network related to the concept of Sport.

This network is clearly hierarchical in structure, where it is organised based on the semantics of the contents. Thus a Football node is a sub-node of a Sport node and Premier and Sunday-league are sub-nodes of the Football node. These semantics provide a relatively lightweight and distributed ontology in which to describe the network contents. The hexagonal structures at the bottom represent actual information sources that bring information of any kind into the network.

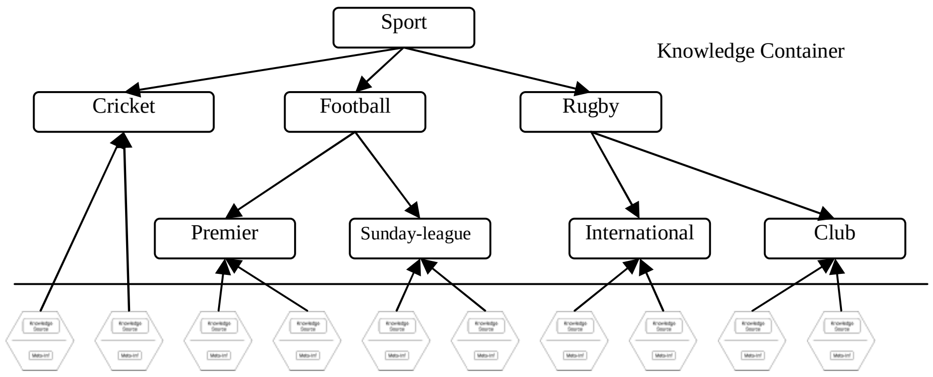


Figure 1: Sport Knowledge Network

The challenges faced for querying the knowledge in this type of network are as follows:

- For the emerging pervasive/sensor-based environment, a lightweight framework is being considered. Ontologies, typically used to represent knowledge, thus need to be created in a relatively lightweight way.
- The querying needs to be distributed, retrieving information from several sources. This requires the query to be constructed dynamically from partial results as it is executed.
- The organisation must be autonomous and should not assume any prior knowledge of the environment. The system is to be generic, stigmergic and self-organising.
- The potential size of the network could be huge, so some query optimisation, directing the search to the most relevant sources, would make a querying process more practical.

While the hierarchical structure is useful for permanent organisation of the knowledge based on semantics, it does not provide complete organisation. One problem is the potential number of similar sources. For example, Figure 1 shows two sources related to the Cricket node. Each source, for example, might represent a different team. If a user asks about Cricket, then the network might need to look at all teams. However, if through links certain sources can be indicated, then the query process can be significantly reduced. The other problem with this ontology construction is that it is still knowledge-based. Stigmergic links will also be able to link sources that are not obviously related through semantics or other comparisons and so can dynamically link what would otherwise be completely unrelated concepts. For example, maybe a Football and Rugby Club use the same stadium. This adds an extra dimension to the optimisation process and so long as the links are sensible, they can be reliably used. One potential application area might be the health service, as has already been suggested by Cameron et al. [2004]. This environment might have a large number of databases with records of patients etc. that could be electronically retrieved and linked through queries. Another area is the Semantic Web [Berners-Lee et al., 2001]. Instead of a user asking to retrieve a single web page, he may ask for information that relates to several web pages. These pages would need to be retrieved and associated together. Thus a way of selecting the best pages to use and link would be useful.

The rest of the report is organised as follows: Section 2 describes the linking structure used to self-optimise with respect to querying and introduce the idea of low level reasoning. Section 3 describes related work. Section 4 describes the test environment and methodology. Section 5 describes the results and conclusions of the tests. Section 6 discusses the report's findings and describes any general conclusions on the work.

2 Stigmergic Linking

Stigmergy is a term that is used to describe the actions of simple entities that, by reacting to their environment, can collectively achieve more intelligent behaviours. The term stigmergy was first used by the biologist Grassé [Grassé 1959] to describe the behaviour of termites as they coordinate their nest-building activities. An example of stigmergy in Artificial Intelligence is the Ant Colony Optimisation algorithm [for example, Dorigo et al. 2000]. ACO works by copying the actions of ants

as they try to find the optimal route from one position to another. They randomly select a number of routes and leave a pheromone behind indicating the route that they took. The shortest route will build up the strongest pheromone amounts and so the other ants will be stimulated to take this route. The ants do not know what the optimal route is, but rather discover it through the experience of all the routes that all ants take. Their reasoning does not require any intelligent knowledge, but only to be able to read the pheromone trail.
A standard ‘Select-From-Where’ statement can be used to retrieve information. This allows a user to select certain values from certain source types where certain conditions are true. The querying process being adopted here is a two stage process. The first stage searches for potential sources that can answer the query and the second stage then queries the sources to retrieve their values. The first stage can also be used as a search engine, where source references can be stored. Thus if the sources are dynamic (sensors for example) it is better to store the reference and then retrieve dynamically the information when needed. For example if a user asks for sources about International Rugby teams, then the first stage can provide a search of the network and retrieve the path references. A user however can also ask for specific values, for example:

Select Premier.team From Premier, Sunday-league Where Premier.goals GT Sunday-League.goals

The network needs to be fed information that it can use to manage the links and it is proposed that this can be done by using the results of the querying process. The nodes that were used to answer each query can be informed and can update their related link values. Each source can store a structure that records the sources related to it through the querying process. This structure can monitor related sources at different levels by assigning weights to them and allows it to recognise when a new source reference should be included or an old source reference removed. In the context of source linking, a weight is simply a numerical amount associated with a source. It can be incremented or decremented and compared to threshold values. If the weight value becomes greater than a threshold value, then the source reference can move up a level in a linking structure. If a source is found to be used in the same query that the current source was used in, then its reference can be added to a link table in the current source. In one set of tests there was also another entry stored for each link path. This was just a single entry that could store a new source before it was moved into the link table itself. This single additional entry was found to have a significant improvement with regard to quality of answer, but as it was stored for each new linking branch, it was difficult to control the amount of memory used when it was included. Thus it has since been made an optional feature and all new entries can be stored directly in the link structure instead if that is possible. Thus each node can know and manage the exact number of entries that it stores. Only the sources referenced in the top level structure are considered to be reliable links that should be returned as sources to look at. So there needs to be some degree of consistent association before a source is returned as a link.

2.1 Linking Methodology

The tests have used a linking structure with three levels. These levels are separate structures that represent possible sources, monitored sources and linked sources. The possible sources structure records new sources that may possibly become links. They need to be associated several more times before they can be called links. The monitor structure is an intermediary structure that stores more advanced possibilities. Having two levels here (possible and monitor) may not be necessary, but it can be helpful for resource management or may allow for initial communication when a source is shown to be related. For example, specific sources can reinforce links when they reach the middle level, even if the whole query is not answered. The linked level is the top level that then stores the references to sources that are actual links. It is these sources that are returned as possible answers when the appropriate query is executed. Each structure can be assigned a limited amount of memory, or number of allowed references, ensuring that it stays lightweight. When this allocation is reached, to add a new reference it must first remove an existing one. Source references are stored with weight values that can be either incremented or decremented. The weight values determine in which level the source references are stored. The weight values must reach a certain threshold value before they can be moved up a level. For example, for the simplest case, say we have a weight increment value of 0.1 and a threshold for the next level of 0.5. If a particular source is associated with the current source 6 times in a row, then its weight reaches the value 0.6, which is greater than the threshold and so its reference can be moved up to the next level. If the same query type is run again however and the source is not used, then its value can be decremented, when it will subsequently be moved down a level if it then falls below the threshold. Additional features are also possible. For example, sources can borrow memory from each other. This means that the total amount of memory stays the same but it is better distributed. More heavily used sources can be allocated a larger amount of memory and thus can more effectively optimise. Learning algorithms might also be included to try and autonomously learn certain parameters, such as the best weight increment or decrement values. Appendix B outlines the main linking procedures that have been implemented.

2.2 Linking Structure

The linking structure itself is meant to record related sources for similar queries. However, it is not a caching mechanism where it stores the whole query with the sources used. It is slightly more lightweight and flexible, where it matches parts of new queries to the structure. The structure can allow for any level of nesting that may be suitable to the current problem. For example, say we have a Premier-league team being linked to Sunday-league teams. If different queries have asked for comparisons related to goals or fouls, then partial query paths like the ones shown in Figure 2 can be stored. At the end of these paths there is a thresholds structure that stores the linked sources at the different levels for the specified path.

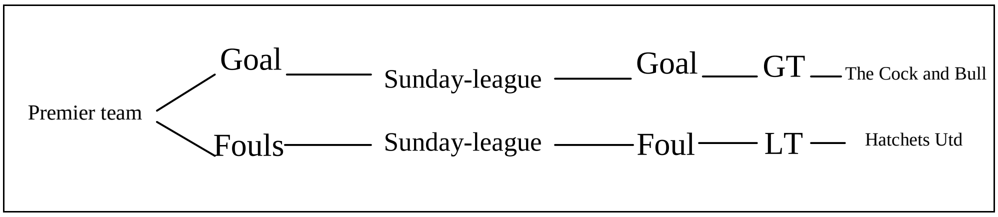


Figure 2: Example of paths relating to links for partial queries.

For a distributed environment, the source path will also include a URL to define a remote server, so that there will not be any confusion as to exactly what source is being referenced. The linking structure is a reference-based structure that stores the addresses of nodes rather than the nodes themselves and can be used in at least two different ways. One way is globally in the network itself. It can be used to link sources that typically answer the same type of query. Then when one of the sources is queried, it can return the sources it is linked to and these can be looked at instead of needing to look at all potential sources. The links stored globally thus optimise the whole network and can be shared between all users. The other possible use is as a local view. A particular application may want to store locally references to the nodes that it typically visits for the queries that it typically answers. These may be a subset of the whole network. A description of the nodes visited by a particular application can be stored locally and monitored in the same way as the source links. Then when a new query is executed, the local view can be searched first and if it contains any relevant information this can be used instead of having to search the whole network. This report will present results of tests evaluating the performance of both the global source linking and local view. Appendix A outlines the main query engine procedure, while the linking structure and methodology are described next.

2.3 Reasoning over the Information

It will also be shown that it is possible to use the links to provide some level of reasoning over the stored information. Queries could be generated to answer questions such as:

- what is the best value based on other values?
- does a value exist based on other values?

If the user is asking if a certain solution exists, then the query engine can simply check that an answer is possible. If the user is asking for the best answer, then the query engine can average all possible solutions to produce a final answer. In generating these answers, the linking mechanism can be used to provide the knowledge required to reason over the information. The links represent the knowledge of the users of the system. They show what pieces of information the users requested and grouped together. Thus it should be possible to use the links in more knowledge-intensive queries. For example, a user might ask how to get to the next game in the Winter season against a London club. The query might look like:

Select Best Transport.Type From Transport, All_Teams Where (Transport.Destination Equals All_Teams.Team_Location) And (All_Teams.Team_Location Includes 'London') And (All_Teams.Game_Season Equals 'Winter')

Thus through simple mathematical operators such as 'average', a basic distributed reasoning engine can be generated. This has been written about previously in Greer et al. [2007b].

3 Related Work

Mano et al. [2006] discuss the main mechanisms used in this report for linking (stigmergy, self-organisation, reinforcement). While this report has described the process as being stigmergic, others might argue that it is more like the Hebbian learning rule, which states that concepts that are activated simultaneously become more strongly associated. This method has been used by Heylighen and Bollen [2002], although their method of clustering seems to require manual user interaction rather than an automatic query process that compares specific values. Stigmergy has been used widely to self-organise in MANETs. These networks are highly dynamic and need a flexible and robust mechanism that can adapt and allow them to self-organise. As these systems can be on a massive scale, some centralised controlling mechanism may not be practical. Babaoglu et al. [2006] and Breukner and Parunak [2004] are papers that discuss this problem. This self-organisation may not lead directly from the querying process but would involve information sharing, though Babaoglu et al. [2006] discusses organising through queries that contain sets of keywords. The architecture of Sartiani et al. [2004] is close to the network architecture suggested in this report. Although, wherever possible, they do clustering based on schema similarity, they note that it is also possible for peers with unrelated schemas to be clustered together. Their super-peers also allow for a hierarchical structure. However, the organisation is slightly different as the super-peers do all of the autonomic clustering. The source clustering tested in this report is not hierarchical, but there is no reason that container nodes could not also cluster using the links. Dragan et al. [2005] is also relevant, as the dynamic linking of sources will also in effect reroute queries. Another example can be found in Vidal et al. [2004] or Raschid et al. [2006]. They apply linking to the problem of optimising routes through web resources in the area of Life Sciences. In this set of resources, there are known to be different routes to different resources that may answer the same query.

An example of linking based purely on the query experiences includes Koloniari et al. [2005]. They try to cluster nodes in a peer-to-peer network based on query workloads. They measure how similar a node’s content is to a type of query, which will mean that it is more likely to return an answer to that type of query. They then try to cluster nodes with similar workloads together in workload-aware overlay networks. They describe that the mechanism for calculating the workload value is still an open issue and could be based on a node storing statistics on the queries that pass through it. Michlmayr et al. [2006] is also along the lines of Kolonari et al., where they stigmergically create links between nodes based on query requests. The requests consist of sets of keywords and they try to cluster documents in repositories and optimise routes based on the requests. Another system that tries to associates nodes based on the query experience is called NeuroGrid (Joseph [2002]). Robinson and Indulska [2003] describe a service discovery protocol called ‘Superstring’. Superstring is used to discover services in an extremely dynamic environment but with a stable central core of nodes. This is exactly the environment that we envision for our system. The lightweight framework allows for a dynamic MANET environment, for example, but the stable central core allows for stigmergic principles to work. They also build up a partial hierarchy based on the semantics of the concepts being searched. These examples however may be more along the lines of a search engine than the evaluation queries tested in this report.

4 Test Environment

A test environment has been written in the Java programming language to test the querying process of the information network. The data used to evaluate the tests is generated in a random manner. This includes the network configurations and the queries to be executed. With specified configurations, networks can be randomly generated that are hierarchically structured, as shown in Figure 1. The queries are also generated in a random manner. The tests however require some control over the similarity of the queries to be executed, to determine how well the linking will do. Distribution bands thus need to be specified for the source and value types. For example, if we have 10 different source types, it is possible to specify that one of three source types should be selected 70% of the time and one of the remaining 7 source types 30% of the time. When a query is randomly constructed, this will skew the specification towards particular query types. The linking specification also needs to be entered, such as threshold values, increment and decrement weight values, or the number of allowed entries in the linking structure. The user also specifies a linking method. This can indicate the type of search process (for example, full search with links), whether to include the comparison operator or single reserve entry in the linking structure, or whether to allow borrowing of memory or learning. The user can also indicate whether to construct and use a view as part of the search process. Note that a separate linking specification can be entered for the view if desired.

4.1 Test Objectives

With parameters specified it is possible to randomly generate a number of queries and execute them on the network. Each query is evaluated on the network and the statistics that are generated, such as node count or level of accuracy, are stored. When all queries have been executed the statistics can be retrieved and analysed. The objectives of this testing are as follows:

- To show that in certain situations, stigmergic linking can be effectively used as part of a query self-optimisation process.
- To estimate at what kind of configuration, with regard to query or network variation, might allow effective linking.

- To indicate any definite conclusions that can be made from this initial set of tests, with regard to link structure, size and performance.

4.2 Testing Methodology

Appendices A and B outline the main algorithms used in the test procedures. These are essentially the query process and linking mechanism. Results from similar linking methods seem scarce, so these tests have compared node count and quality of answer between searches that use links and searches that do not (called a full search). Note that the full search is still guided by the hierarchical structure and storage of previous query evaluations as described in Appendix A. The only difference is that it will not try to retrieve linked sources. The search strategy was as follows: Two different searches were performed for each query. The first was a full search only and the second was a linked search as described in Appendix A. The node count and quality of answer from these two searches could then be compared. However, if the linked search did not return an answer and the full search did then the full search node count would need to be added to the linked search node count as part of the linked search. It was assumed that a full search would then be performed to try and return an answer.

The performance of the linking was measured as the amount of reduction in node count and also as the quality of answer returned. The percentage of reduction in number of nodes visited could be measured by comparing a node count from a full search only with a node count from a linked search. The queries requested values from sources that were of the integer type. The evaluation function then tried to maximise the sum total for all source values requested to produce the best quality of answer. As the network is service oriented, the term Quality of Service (QoS) is used to mean the same as quality of answer. The full search has access to all source instances and so should return the best answer. For a linked search, the number of sources looked at would be reduced and so you would expect this maximum value also to be reduced. However, if the correct links have been established, then a good answer should still be returned. This percentage of reduction was taken to be the reduction in quality of service. When measuring QoS reduction, only queries answered through a linked search were included. If a linked search required a further full search to find an answer, then the answer value would not be included in the statistics. The values were all of type integer, but this is equally suitable for text or concept matching as for numerical comparisons. For example, 10 different integer values can be compared in the same way as 10 different textual words. Then the equivalence matching can apply equally for numbers or for text.

5 Test Cases

Following are the results of tests that give an indication of how effective the linking is likely to be. The tests revealed that changes in configuration parameters could affect the performance and so it is not possible to give a definitive evaluation. The tests also suggested that configuration would require tuning if trying to achieve optimal performance, as linking parameters, such as memory allocation, increment/decrement weight values or threshold values need to be specified or learned. Each value measured was averaged over at least three test runs. The testing focused mainly on the type of query being executed and conclusions will be given at the end of the tests sections. The network configuration was not tested as much. For the linking, the most critical factor is the variety in the query and so this was the main evaluation consideration.

5.1 Test1: Tests with Queries skewed by a 70:30 Distribution

One set of tests was for queries that were skewed with a 70:30 distribution. The linked search either included the extra single entry in a reserve position (reserve entry), or alternatively a view. The reserve entry was a single additional entry for each branch of the linking structure and could be considered as a sort of benchmark that other setups without it should try to achieve. The number of queries executed ranged from 5000 to 50000 queries. This set of tests performed two evaluations.

- It compared the reserve entry to a view.
- It considered linking effectiveness with respect to text or concept matching in particular.

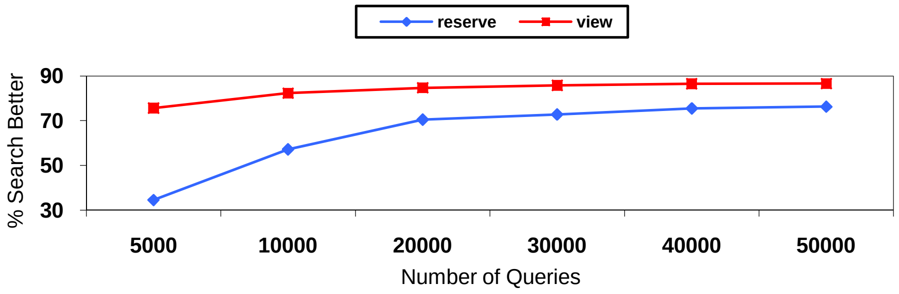
The network configuration for these tests was as follows:

- There were a total of 300 sources in a network with 315 nodes.
- There were 10 different source types and 5 different value types.
- There were a total of 30 instances of each source type, where each source instance contained all 5 value types.
- Each value type could have a range of integer values from 1 to 10.
- The distribution for the queries was 0.7:0.3. So for the sources, 70% of the time a source would be selected from one of 3 source types and 30% of the time from one of the remaining 7 source types. For the values, 70% of the time a value would be selected

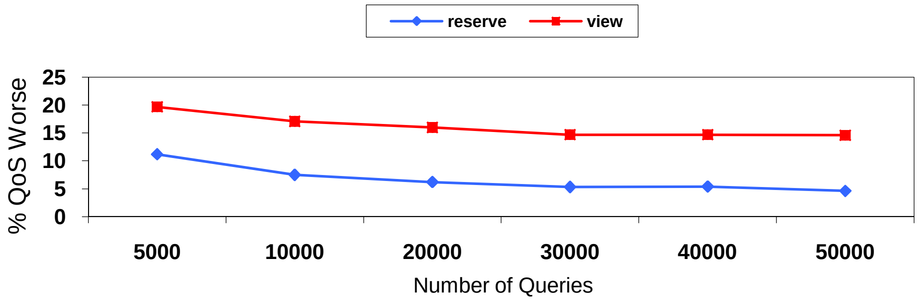
from one of 2 value types and 30% of the time from one of the remaining 3 value types.

- For the linking methods, each link level (possible, monitor and link tables) was allowed a total of 50 entries for each source. The monitor level threshold was 0.1 and the link level threshold was 0.4. The increment value was 0.1, while the decrement value was 0.05 times the increment.
- The view size allowed 100 entries for each of the three link levels.
- Each query was made up of a maximum of 2 sources in the 'Select' clause and 3 sources in the 'From' clause. If the 'Select' clause had 2 sources, the 'From' clause had 2 or 3 sources.
- The queries contained 'Where' clause comparisons with only the equivalence operator. For example A 'EQ' B. No other operators (GT, GE, LT etc) were used.
- The reduction in number of nodes searched and reduction in QoS were measured.

Graph 1 and Graph 2 are an example of the results that can be achieved. One of the trends in the graph is for a set of tests where the reserve position was included (reserve) and the other is for a set of tests where the reserve position was omitted but a view was allowed (view). While the reserve entry seems to produce a much better QoS, the view produces a better reduction in node count. The view can limit the number of nodes used as part of the first source retrieval, thus reducing the search afterwards as well.



Graph 1: Skewed Query Test (70:30). Percentage of reduction in the number of nodes searched for queries with the equivalence operator only. Link structures with either a reserve entry or a view are included.



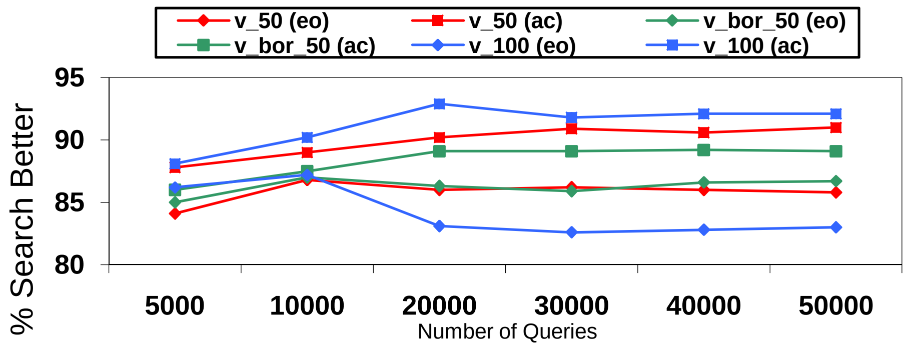
Graph 2: Skewed Query Test (70:30). Percentage of reduction in quality of service for queries with the equivalence operator only. Link structures with either a reserve entry or a view are included.

5.2 Test2: Tests with Queries skewed by a 90:10 Distribution

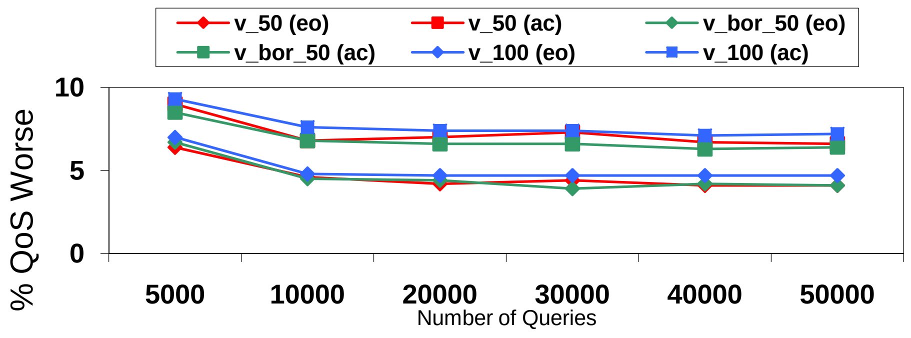
This set of tests tried to determine if more skewed queries could effectively link sources that answered queries with all types of comparison operator. If this is possible then numerical data can also be linked. But because this increases the complexity of the query, the source and value types may need to be skewed more to compensate for this. The reserve entry was not included in these tests but a view was allowed. The network configuration and linking structure was the same as for the previous tests with the 70:30 skewing (section 5.1). This set of tests:

- Compared queries with equivalence only comparisons to those with all comparison types.
- Also considered memory borrowing as part of the linking process.

To show the importance of the configuration values, tests were also run with a maximum of 100 entries at the possible links level for each source and 200 allowed link entries in the view. These results were averaged over 6 test runs each. Graphs 3, 4 and 5 give the results of these tests.

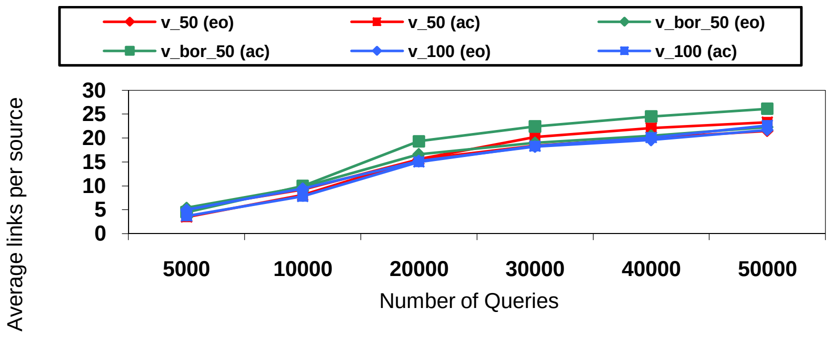


Graph 3: Skewed Query Test (90:10). Percentage of reduction in the number of nodes searched for queries with the equivalence operator only (eo) or all comparison operators (ac). Link structures with a view that do (v_bor) or do not (v) borrow memory are included. Number of entries at the possible links level either 50 or 100.



Graph 4: Skewed Query Test (90:10). Percentage of reduction in quality of service for queries with the equivalence operator only (eo) or all comparison operators (ac). Link structures with a view that do (v_bor) or do not (v) borrow memory are included. Number of entries at the possible links level either 50 or 100.

When considering search reduction, it can be seen that increasing the number of allowed entries at the possible links level has had an affect on the performance. There appear to be optimal levels that then reduce as more queries are executed. For the equivalence only queries, these graphs suggest that increasing the number of updates after 10000 queries or so might actually have a detrimental effect on the effectiveness of the linking and for the all comparison queries, it is after 20000 – 30000 queries. The QoS stays roughly the same, but the search reduction starts to decrease. The reason must be that extra links are being added that are not then helping with the search process. However, if there are only 50 allowed entries at the possible links level then there is not the same reduction in performance. Graph 5 shows the average number of link references stored for all sources, but does not give any indication of the distribution over each source. This number of links can be compared with the percentage of search reduction shown in Graph 3.



Graph 5: Skewed Query Test (90:10). Average number of source links stored for queries with the equivalence operator only (eo) or all comparison operators (ac). Link structures with a view that do (v_bor) or do not (view) borrow memory are included.

As might be expected, this comparison shows that increasing the number of stored links improves performance, but only up to a point. Graph 5 shows that the link numbers do not reach the maximum of 50 on average that is allowed; but this measurement does not consider individual sources, when more important sources might be expected to store more links. A result at the end of section 5.5 indicates that only allowing a maximum of 25 links also produces good results and so Graph 5 is not too deceiving. There appears to be an upper limit on the amount of memory that should be used, or even possible performance that this particular linking mechanism can provide. The extra entries at the possible links level must have allowed more variety in the references that make the link level, or alternatively, having fewer allowed entries at the possible links level has concentrated the variety at the link level.

5.3 Test3: Increase the Network Variability

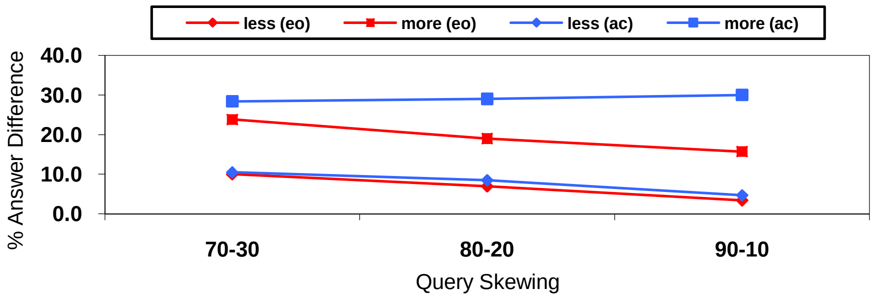
Other tests tried to test how increasing the network variability might affect the performance. This was done in two different ways. The first was to increase the query range by having more source types or value ranges. The other test increased the number of instances of each source type. The results showed that increasing the source type and value ranges from 5 types and a range of 1 to 20, to 20 types and a range of 1 to 20, showed that performance remained relatively stable. Increasing the number of source instances to 90 also showed the performance remaining relatively stable.

5.4 Reasoning Tests

This set of tests evaluated the potential of the linking mechanism to provide some level of reasoning over the information stored in the network. To test this, queries that asked the 'best' or 'exists' queries were generated. It proved relatively easy to do this. The query generator would place the keyword 'best' or 'exists' in front a standard information retrieval query. For example:

Select Best A.val1 From A, B Where (A.Val2 LT B.Val3)

The keyword would then tell the query engine to average all of the possible answers to give the best total. Thus all possible solutions would be retrieved and then their values averaged to provide the final answer. For the 'exists' queries, the engine would check that any answer existed. Tests for this used the test 1 setup but varied the skew from 70:30 – 80:20 – 90:10. Graph 6 shows the results of this.



Graph 6: Reasoning test. Percentage range of average answers for a linked search compared to a full search for varying query skewing. Equivalence only (eo) or all comparison (ac) queries tested.

The results provide an interesting conclusion. The average answers range from values larger than the best answer for a single solution, to values less than that. For the 'best' queries, results showed that the QoS ranged from maybe 10-30% better to around 5-10% worse. The QoS metric tries to maximise the answer total and so larger totals would be preferred. This suggests that the linking 'intelligently' prunes the lower total answers from the search, which the full search would also return. Thus the linked search provides a better QoS as well as node count, depending on your measurement. The assumption here is that the evaluation function considers larger totals as being better. The links are created based on this metric and all queries evaluated using it. Thus if the reasoning queries return larger totals then this is considered as being better. If some other metric is used for the reasoning queries, for example the user wants purely the average of all possible solutions without any sort of skewing, then this metric would not be helpful. There also does not appear to be a direct correlation between the number of answers returned with larger values and the overall percentage value. The number could vary by quite a large amount, with the percentage value varying by much less.

5.5 Conclusions to the Tests

It is useful to make some initial numerical calculations, to try and determine the relative size of the linking structure compared to possible query variations that could be used. These calculations can be done for the test configuration specified in test1. As there are 10 source types and 5 value types in total, this gives 50 possible combinations. For a completely random query, each source/value combination thus has a 2% chance of being selected. When the queries are skewed, they can either be skewed by

70:30 or 90:10, where the source split is 3:7 and the value split is 2:3. Table 1 gives the probability that an individual source or value will be selected for a part of the query based on Equation 1. These combinations lead to 4 different bands to which a source/value combination might belong. Table 2 indicates the relative frequency by which a combination from one of the 4 different bands will be selected. The frequency is calculated by Equation 2 and Equation 3:

Equation 1: probability of being selected = probability for band / number in band.
Equation 2: probability a source/value combination will be selected = (probability source band selected * number in band) * (probability value band selected * number in band) / (number of sources in band * number of values in band).
Equation 3: frequency = probability for the band / probability for the lowest valued band.

The percentage of all possible links that a source or view can store can also be calculated. As there are 30 instances of each source type, assuming a ‘value-source’ path for a view entry, a view can store 10 * 30 * 5 = 1500 link combinations. Assuming a ‘value1-source-value2-comparison-source_instance’ path for a source entry - if a query allows equivalence only queries, then the total number of possible links is 299*5*5 = 7,475 link combinations. If all comparison operators are allowed, this increases to 299*5*5*6 = 44,850 link combinations. From this we can tell that 200 entries in the view means 13.3% of all possibilities can be stored as actual links. For the sources – for equivalence only queries, each source can store 50 references, or 0.7% of all possibilities, while for queries with all comparison operators, this reduces to 0.1%. The actual number stored in the top level as links rose to around 25, which is 0.33% or 0.06%

respectively. These numbers indicate that the linking can be quite effective.

70:30								90:10							
Source				Value				Source				Value			
Band1:0.7		Band2:0.3		Band1:0.7		Band2:0.3		Band1:0.9		Band2:0.1		Band1:0.9		Band2:0.1	
P	N	P	N	P	N	P	N	P	N	P	N	P	N	P	N
0.233	3	0.04	7	0.35	2	0.1	3	0.3	3	0.014	7	0.45	2	0.033	3

Table 1: Table showing the probability (P) that a source or value will be selected depending on the band it is in and the probability skew. Also indicated is the number of source or value types (N) to which this probability relates.

70:30					90:10				
	Value					Value			
	Band1:0.7		Band2:0.3			Band1:0.9		Band2:0.1	
Source	F	N	F	N	Source	F	N	F	N
Band1:0.7	20.25	6	5.75	9	Band1:0.9	293	6	19	9
Band2:0.3	3.5	14	1	21	Band2:0.1	15	14	1	21

Table 2: Table showing the frequency (F) that a source/value combination will be selected from one of the 4 bands by, based on the number in each band and the skew probability. Also indicated is the number of all source/value combinations (N) to which this frequency relates.

For the 90:10 split, Table 2 indicates that a combination from the highest valued band will be selected with a frequency of 293. Although the tests show that the variation is much greater than this, if we reduce the queries to just this upper band we get 29 * 3 * 2 = 174 (source instances number * source types * value types) possibilities for the equivalence only queries, or 29 * 3 * 2 * 6 = 1044 (include different comparison operators) possibilities for the all comparison queries. Test2 thus suggests that the average number stored as links in each source is just under 15% (25 / 174) of all possibilities for the worst case. For these tests the view is used only to retrieve links for the first source evaluated. Thus, a comparison with the source links only here looks reasonable. This suggests that the linking is an effective optimisation technique, providing an appreciably greater than 1:1 relation with memory allocation. If around 15% of all possible combinations are stored as links, a much greater than 15% improvement in performance would be expected. Measurements here show 80-90% search reduction, for example. This does not mean however that adding more link entries will automatically improve performance by the same amount. As well as the upper limit that has been suggested there is also not a direct correlation. For example, the test 1 configuration with the same view but only 25 entries at the link level reduced search performance by only around 2-3%. The conclusion is simply that if you store a certain percentage of the possibilities as links you would expect to get a much better improvement than that in performance.

5.5.1 Summary of Test results

This section will summarise the test results and make some comparisons to other peoples' tests in related areas. As this type of query mechanism has not been evaluated before, the comparisons cannot be direct comparisons, but only indicate the sort of performance numbers that can be involved.

5.5.1.1 Summary of Test 1 Results - Skewed Query Test (70:30)

The test1 set of tests was for queries that considered the equivalence operator only and show a significant reduction in node count with a reasonable QoS. This would be useful for text or concept matching, for example. The reserve entry provides a good QoS, but the view provides a better search reduction. If both are combined however, rather than benefiting from both, it acts more like a view. The initial filtering from the view seems to have reduced the QoS unless the queries are sufficiently skewed. One other test tried this configuration without a reserve or view. This then performed more like the reserve option, with a search reduction of around 76% and QoS reduction of around 6% (compared to under 5% with a reserve) for the test 1 configuration.

5.5.1.2 Summary of Test 2 Results - Skewed Query Test (90:10)

When considering search reduction, the test2 tests seem to show a significant conclusion. If the equivalence only queries can achieve a certain level of accuracy after 20000 queries, then adding more links after this might only increase the search without further helping to find a better answer. But this is strongly dependent on the configuration. So the system might need to monitor this and determine an upper bound on the memory allocation. This would also be a limiting factor on the possible effectiveness provided by the links. This monitoring could be done as part of the supervision system inherent in autonomic systems. This would be an interesting area for more monitoring, to determine when to stop adding new links and has been written about in Greer et al. [2007a]. A drop in performance when new links are added could indicate an upper bound on the memory allocation. Graph 4, showing the QoS, indicates a very good quality for the equivalence only queries and reasonably good for queries with all comparison operators. A generally prevailing rule through all tests would be that a larger search would produce a better quality of service. However, as shown by the reserve results for test1, when the links being added are key, both factors can be improved. Graph 5 shows that the link numbers levels out at around 25 for all sources, although it is still increasing slowly. This is an average value however and does not consider the exact distribution over the skewed sources. Statistics also show that the proportion of references to queries for the more popular source or value types increases with increasing query numbers. Thus if these types are queried more and have more links, then this would increase the search size. This is an estimate for the whole source or value type and not individual instances. The link numbers also suggest that even with high skewing the large numbers of queries are sufficiently random to allow a large amount of variation. It might also suggest that the current parameters make incrementing links much easier than decrementing them, so that sources reach the link level relatively easily. The test2 results show that borrowing memory gives a clear improvement in QoS for the all comparison queries. This should be because of the more flexible memory distribution.

5.5.1.3 Summary of Test 3 Results – Query Range and Network Variability

Test3 tests show reasonably good consistency over increasing the value ranges, suggesting that some scaling up is possible there. However, scaling up the number of source instances is not as convincing as was anticipated. The linking performance seems to hold its own, but does not really improve. The original supposition was that a larger ratio of source instances to source types would be favourable, because the linking process selecting specific source instances would then be more effective. However, this does not seem to be the case. Having more instances with the same range of values means that there is more chance that a full search will find a source with the optimum set of random values for each query part. Thus the linking has to be more accurate in this case to provide the same level of QoS. Also, the process used to answer the queries could have a normalising effect on the number of nodes visited as it limits the number from one query part to the next.

5.5.1.4 Summary of Test 4 Results - Reasoning

The initial reasoning tests in test 4 show that using links is beneficial. The links can 'intelligently' prune the worse solutions from the search space while still leaving an answer. Thus a better average value will be calculated and so both search reduction and QoS will be improved.

5.5.1.5 Related Test Results and Other Conclusions

There appears to be little directly related test results available. Koloniari et al. [2005] indicate that their tests showed that 60% more queries could be answered in the same amount of time when nodes were clustered using their algorithm. These tests have not quite reached 60% but QoS is also indicated. Michlmayr et al. [2006a] state that in the space of 2000 to 10000 queries, the number of links travelled to find an answer reduces from 60.44 to 53.02 (12% reduction) and the number of documents retrieved increases from 1.82 to 3.95 (over twice as many). In Michlmayr et al. [2006b] they provide a different set of results for a search using taxonomies. They formulate queries consisting of single keywords taken from a taxonomy. A number of source documents are created, where each peer is an expert in a particular topic based on a percentage value. For example, a P_{expert} value of 60% means that 60% of the peer's documents relate to the concept that it is expert in and 40% are random. The hit rate for the number of relevant documents retrieved for a particular time interval is then measured. They note that their most significant result is an improvement of 39.5% for a P_{expert} value of 60%, while for a P_{expert} value of 80%, the improvement is 38.2%. These values might be compared to the query distributions with search reduction, as this also measures improvement over variety.

6 Conclusions and Discussion

The aim of this work has been to evaluate a mechanism to allow an information network to optimise itself dynamically. One objective is that the network structure should remain lightweight and so the organising mechanism should also be lightweight. To this end stigmergic links have been used. A weight value and reference between two nodes is very lightweight and the total amount of memory can also be controlled. It should also be flexible as we do not know exactly what information will be stored. Thus the results of previous queries can be used to optimise the network, as this is completely dynamic, reflecting the system use. It does however require a certain amount of consistency with respect to the type of query that is executed for it to be effective. The intention of the tests in section 5 is to show that stigmergic linking can fulfil the optimisation requirements. The links represent the knowledge held in the network and so together with some form of semantic organisation, could largely replace a centralised ontology to describe the knowledge in the network. The exact type of query being executed or the exact network structure is not so important. This sort of query, linking sources through comparisons, may be unusual in an XML-based information network, but the intention is to produce a richer or more complex type of query. The linking mechanism is generic, but the test results are tied into the query procedure that is used. Hyperlinks that link related Web pages might be a similar sort of application, or comparing values suggests some sort of record-based system. For example, Cameron et al. [2004] suggest a linking mechanism for database records, such as those in health care, retrieved from Web services. Another area could be the Semantic Web [Berners-Lee et al., 2001], when queries will retrieve information from several related web pages rather than from a single Web page.

Stigmergic linking can clearly be used as part of a self-optimisation process and introduce new knowledge into the network. Query variation rather than network configuration has been tested, as either should determine the linking effectiveness. Querying a network with more variety in its node types should lead to the same query variation. The tests suggest that retrieving links can produce a worse performance when there are either too many or too few links, and so the optimal configuration is important. With a certain level of confidence however, it can be stated that provided a good configuration is found, the linking should improve performance by something appreciably larger than a 1:1 relation with memory allocation. Some other results in similar areas have been found, but it is difficult to make direct comparisons, due to the differences in the type of query that is executed. While these test results seem new, the indexing and retrieval mechanism of query parts might be the most novel aspect of this work, providing the desired lightweight and flexible framework.

Tests have also shown that the links can be used to reason over the network's information. They represent the use of the system and thus also the knowledge of the users of the system. Thus they can be used in more knowledge-intensive queries. Through simple aggregation, higher levels of reasoning can be obtained.

Acknowledgments

Work supported by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

7 References

Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G., Ducatelle, Gambardella, F., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A. and Urnes, T. Design Patterns from Biology for Distributed Computing, ACM Transactions on Autonomous and Adaptive Systems, 1 (1) (2006) 26 – 66.

Baumgarten, M., Bicocchi, N., Curran, K., Mamei, M., Mulvenna, M.D., Nugent, C., Zambonelli, F. 2006. Towards Self-Organizing Knowledge Networks for Smart World Infrastructures, Invited Session on Service Development and Provisioning through

Situated and Autonomic Communications at International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS'2006), Erfurt, Germany, (2006) pp. 18 - 21.

Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, Scientific American, May (2001).

Breukner, S. and Van Dyke Parunak, H. Self-Organising MANET Management, G. Di Marzo Serugendo et al. (Eds.): AAMAS 2003 Ws ESOA, Lecture Notes in Artificial I ntelligence (LNAI) 2977, 2004, pp. 20 – 35.

Cameron, M. A., Taylor, K. L. and Baxter, R. Web-Service Composition and Record Linking VLDB Workshop on Information Integration on the Web (IIWeb-2004) Toronto, Canada, 2004.

Dorigo, M., Bonabeau, E., and Theraulaz, G. Ant algorithms and stigmergy, Future Generation Computer Systems, 16 (2000) 851 – 871.

Dragan, F., Gardarin, G. and Yeh, L. MediaPeer: a safe, scalable P2P architecture for XML query processing, Database and Expert Systems Applications, Proceedings. Sixteenth International Workshop on, 22-26 Aug. 2005, pp. 368- 373.

Fu, L. Knowledge Discovery based on Neural Networks, Communications of the ACM, 42 (11) (1999) 47 - 50.

Grassé P.P. La reconstruction dun id et les coordinations internidividuelles chez Bellicositermes natalensis et Cubitermes sp., La théorie de la stigmergie: essais d'interprétation du comportment des termites constructeurs, Insectes Sociaux, 6 (1959) 41-84.

Greer, K., Baumgarten, M., Mulvenna, M., Curran, K., and Nugent, C. Autonomic Supervision of Stigmergic Self-Organisation for Distributed Information Retrieval, Workshop on Technologies for Situated and Autonomic Communications (SAC), at 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems, BIONETICS 2007, December 10-13, Budapest, Hungary, (2007a).

Greer, K., Baumgarten, M., Mulvenna, M., Nugent, C and Curran, K.. Knowledge-Based Reasoning through Stigmergic Linking, International Workshop on Self-Organising Systems IWSOS'07, 11 -13 September, Lecture Notes in Computer Science (LNCS), Eds. David Hutchison and Randy H. Katz, Vol. 4725, 2007b, pp. 240 – 254, Springer-Verlag.

Heylighen, F. and Bollen, J. Hebbian Algorithms for a Digital Library Recommendation System, Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02), 2002, pp. 439-.

Joseph S. NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks. In Proceedings of the International Workshop on Peer-to-Peer Computing (co-located with Networking 2002), Pisa, Italy, May 2002.

Koloniari, G., Petrakis, Y., Pitoura, E., and Tsotsos, T. Query workload-aware overlay construction using histograms, Proceedings of the 14th ACM International Conference on Information and Knowledge Management, 2005, pp. 640 – 647.

Mano, J-P., Bourjot, C., Lopardo, G. and Glize, P. Bio-inspired Mechanisms for Artificial Self-organised Systems, Informatica, Vol. 30, 2006, pp. 55 - 62.

Michlmayr, E., Pany, A., and Graf, S. Applying Ant-based Multi-Agent Systems to Query Routing in Distributed Environments, in Proceedings of the 3rd IEEE Conference On Intelligent Systems (IEEE IS06), September 2006a.

Michlmayr, E., Pany, A., Kappel, G. Using Taxonomies for Content-based Routing with Ants, 2nd Workshop on Innovations in Web Infrastructure (IWI2006), Co-located with the 15th International World-Wide Web Conference, May 2006b, Edinburgh.

Mulvenna, M.D., Zambonelli, F., Curran, K., Nugent C.D. Knowledge Networks, In: I. Stavrakakis and M. Smirnov (Eds.), Autonomic Communication, Springer-Verlag, Lecture Notes in Computing Science, LNCS 3835, 2006, pp. 99-114. ISBN 3-540-32992-7.

Raschid, L., Wu, Y., Lee, W., Vidal, M., Tsaparas, P., Srinivasan, P., and Kumar Sehgal A. Ranking Target Objects of Navigational Queries, 8th ACM International Workshop on Web Information and Data Management WIDM '06, 2006, pp. 27 – 34.

Robinson, R. and Indulska, J. Superstring: a scalable service discovery protocol for the wide-area pervasive environment, Networks, 2003. ICON2003. The 11th IEEE International Conference on, 2003, pp. 699- 704, ISSN: 1531-2216, ISBN: 0-7803-7788-5.

Sartiani, C., Manghi, P., Ghelli, G. and Conforti, G. XPeer: A Self-organising XML p2p Database System, http://www.di.unipi.it/~ghelli/papers/SarManGhe04-p2pdb.pdf, 2004.

Vidal, M., Raschid, L. and Mestre, J. Challenges in Selecting Paths for Navigational Queries: Trade-off of Benefit of Path versus Cost of Plan, Seventh International Workshop on the Web and Databases (WebDB 2004), 2004, pp. 61 – 66.

8 Appendix A – Query Engine Procedure

This appendix outlines the main procedure used to query the network. This procedure assumes that links are included in the querying process. Individual sources are retrieved and stored through comparisons with other sources through the 'Where' clause statements. Consider the following query:

Select A.value1, B.value2 From A, B, C where A.value3 GT B.value1 AND B.value2 LT C.value2

Query Process

To evaluate this we retrieve each 'Where' clause in reverse order and evaluate to store sources that can be used for the next stage. Thus the first stage is:

Evaluate B.value2 LT C.value2

1. To evaluate this we first check if a previous evaluation has stored any 'C' sources. As this is the first evaluation this will not be the case. We thus check if the view stores any 'C' sources relating to 'value2'. If it does these are retrieved and used. If it does not then a full search is performed to retrieve all of the 'C' sources.

2. We then try to retrieve related ‘B’ sources. If any exist from a previous evaluation then they are retrieved. As this is the first evaluation they do not and so the ‘C’ sources are asked to return any links to ‘B’ sources relating to the ‘value2-B-value2-LT’ path. If these links do not exist then a full search is performed to retrieve all of the ‘B’ sources.
3. Having retrieved the ‘B’ and ‘C’ sources they are evaluated to determine the combinations that satisfy the comparison. These combinations are then stored in structures to indicate this and also indicate the only sources of this type that can be used in the rest of the query process.

Evaluate A.value3 GT B.value1

1. To evaluate this, the ‘B’ sources that are stored are retrieved.
2. Any ‘A’ sources linked to them through the ‘value1-A-value3-GT’ path are also retrieved. If none exist then a full search is performed to retrieve all of the ‘A’ sources.
3. The ‘A’ and ‘B’ sources are compared to determine the combinations that satisfy the comparison. This will produce the set of valid ‘A’ sources.

Calculate the Answer

1. The related ‘A’ and ‘B’ sources that provide the maximum combined total for the required values are then returned as the result.
2. The source links can then be updated based on the finally selected sources using the *sources to update* procedure.

Sources to update procedure

1. The query process has indicated a number of related sources used to answer the query. They are essentially produced through the ‘Where’ clause comparisons.
2. Starting with the sources actually used to answer the query. The related sources can be retrieved and links updated between related sources to indicate their association.
3. The linking is actually done in a reverse order. For example, the B source references are stored as links to the C sources and the A source references are stored as links to the B sources.
4. If a view is used, then it is also updated with the related sources, but maybe with a simpler path structure, for example, just source name and value type.
5. The view is a global view for a whole network. It is a partial re-construction of the whole network that stores nodes representing full paths to each source type. The source type node then stores a linking structure to reference actual source instances for particular value types.

9 Appendix B – Link Update Procedures

This appendix outlines the main procedures used to update the linking structures as a result of the querying. The first procedure describes the basic link update mechanism, while the others outline the additional features that can be included.

Main link update procedure

1. Select next source *nextSource*.
2. Pass *nextSource* the path to a link structure and the sources to be linked based on the last query.

3. *NextSource* retrieves the appropriate threshold structure for the specified path.
    - 3.1. Any sources already in the thresholds structure that are not part of the query answer have their weight value decremented by the decrement amount.
    - 3.2. If these sources now fall below the allowed threshold for a level then try to move them down a level.
        - 3.2.1. If the lower level is full then try to move its sources down first.
        - 3.2.2. Repeat this until the bottom level has enough sources removed to allow subsequent moves in the other levels to allow the upper level sources to be moved down.
    - 3.3. For any source specified as part of the answer, check if it already exists in the structure.
    - 3.4. If it already exists then increment its weight value by the increment amount.
    - 3.5. If the weight value now passes a threshold then try to move it up a level.
        - 3.5.1. Check if the upper level is full.
        - 3.5.2. If the upper level stores its maximum allowed number of entries:
            - 3.5.2.1. Check if the current source still has a larger weight.
            - 3.5.2.2. If this is the case then swap this source with the lowest valued source in the upper level.
            - 3.5.2.3. If this is not the case then if borrowing is allowed, try to borrow memory using the *borrow memory* procedure.
            - 3.5.2.4. If borrowing is not possible, the current source stays at its current level.

            3.5.3. If the upper level is not full then move the current source up a level.
        3.6. If the current source is new then try to add at the bottom level.
        3.7. If the bottom level is full.
            3.7.1. If borrowing is allowed, try to borrow memory using the *borrow memory* procedure.
            3.7.2. If borrowing is not possible, check the lowest valued source's weight. If this is less than the weight of the new source then remove the referenced source and replace with the new source.
            3.7.3. If the lowest valued source still has a larger weight, then if the reserve entry is allowed store the new source there. It can then be subsequently incremented again until its weight reaches a large enough value.
            3.7.4. If the reserve entry is not allowed then the new source cannot be stored.
        3.8. If the bottom level is not full then add the new reference at the bottom level.
4. If learning is allowed, adjust parameters based on a learning procedure, for example see *learning procedure*.

Notes: the memory allocation is counted for each source and is a single total for all link paths together. When determining the lowest valued source to move down or remove, the current implementation considers all threshold structures and not just the path currently being updated.

Borrow memory procedure

One of the linking options allows one source to borrow memory from another. This means that the total amount of memory stays the same but individual sources can store more or less entries depending on their use. For these tests the borrowing has been restricted to sources referenced only from the parent container – in effect only from sources of the same type. This reduces complexity and also means that a search is not required to find sources to borrow memory from. The procedure is essentially as follows:

Current memory – the current amount of memory allowed at a particular level.
Current entries– the current amount of memory used at a particular level.
Total borrowed – the total amount of memory borrowed at a particular level.
Total lent – the total amount of memory lent at a particular level.

Equ 1 – number of slots that can be spared = current memory – current entries – total borrowed.
Equ 2 – adjust allocated amount based on lent memory:

```
total lent += lent memory;
current memory -= lent amount.
```

Equ 3 – adjust allocated amount based on borrowed memory:

```
total borrowed += borrowed amount;
current memory += borrowed amount.
```

1. The source in question retrieves all related sources that it can borrow memory from.

2. It asks each source in turn for a certain number of slots at a certain level.
3. These sources reply with the number of slots that they can spare based on equation 1.
4. The source in question selects a source or number of sources and borrows memory from them.
5. The sources that lend the memory adjust their allocated amounts based on equation 2.
6. The source that borrows memory adjusts its allocated amount based on equation 3.
7. Any source that lends memory can demand it back if it runs short. The source that it lends memory to must then release that specified amount of memory and adjust its linking structure appropriately – removing low level links if required.

Learning procedure

The learning procedure tested is very simple. It adjusts the weight increment and/or decrement values based on the number of sources incremented or decremented for the last link update. There are probably many different algorithms that could be tried.