

2000

Semantic software scouts for information retrieval

John J. Rehder

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Rehder, John J., "Semantic software scouts for information retrieval" (2000). *Dissertations, Theses, and Masters Projects*. Paper 1539623977.

<https://dx.doi.org/doi:10.21220/s2-8zyp-mq20>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

SEMANTIC SOFTWARE SCOUTS FOR INFORMATION RETRIEVAL

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

by

John J. Rehder

2000

UMI Number: 9982019

UMI[®]

UMI Microform 9982019

Copyright 2000 by Bell & Howell Information and Learning Company.

**All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.**

Bell & Howell Information and Learning Company

300 North Zeeb Road

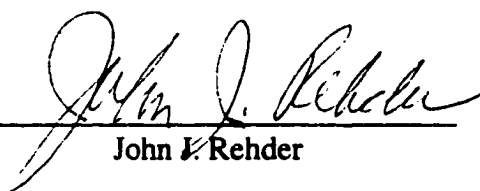
P.O. Box 1346

Ann Arbor, MI 48106-1346

APPROVAL SHEET

**This dissertation is submitted in partial fulfillment of
the requirements for the degree of**

Doctor of Philosophy



John J. Rehder

Approved, April 2000



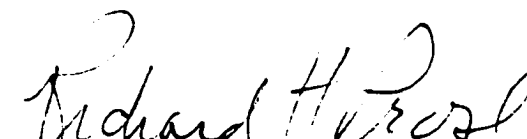
William L. Bynum, Dissertation Advisor




Stefan Feyock



Phil Kearns



Richard Prosl



George Rublein

DEDICATION

This thesis is dedicated to my wife, Shirley Confino-Rehder. Without her love, encouragement, and support, this work would have not been achieved. My appreciation is boundless.

Because thou hast the power and own'st the grace
To look through and behind this mask of me
(Against which years have beat thus blanchingly
With their rains), and behold my soul's true face,
The dim and weary witness of life's race,
Because thou hast the faith and love to see,
Through that same soul's distracting lethargy,
The patient angel waiting for a place
In the new Heavens.--because nor sin nor woe,
Nor God's infliction, nor death's neighbourhood,
Nor all which others viewing, turn to go,
Nor all which makes me tired of all, self-viewed.--
Nothing repels thee, . . . dearest, teach me so
To pour out gratitude, as thou dost good!

Elizabeth Barrett Browning.

TABLE OF CONTENTS

| | Page |
|---|------|
| ACKNOWLEDGEMENTS | vi |
| LIST OF TABLES | vii |
| LIST OF FIGURES | viii |
| ABSTRACT | ix |
| CHAPTER 1. INTRODUCTION | 2 |
| CHAPTER 2 MOTIVATION AND BACKGROUND | 6 |
| 2.1 Motivation | 6 |
| 2.2 Background | 6 |
| CHAPTER 3. RELATIONSHIP TO OTHER RESEARCH | 11 |
| 3.1 Related Research Activities | 11 |
| CHAPTER 4. SEMANTIC HYPERTEXT | 16 |
| 4.1 Knowledge Retrieval with Hypertext | 16 |
| 4.2 Cooperative hypertext retrieval | 17 |
| 4.3 Construction of Semantic Hypertext | 19 |
| 4.4 Semantic Characterization | 20 |
| CHAPTER 5. DOCUMENT BASE CONSTRUCTION | 28 |
| 5.1 Document Source | 28 |
| 5.2 Link Taxonomy | 30 |
| 5.3 Similarity Determination | 31 |
| 5.4 Link Syntax | 35 |
| 5.5 Document Base Statistics | 36 |
| 5.6 The Problem with Typed Links | 37 |
| 5.7 Construction of a simulated document base | 38 |

| | |
|--|-----------|
| CHAPTER 6. SCOUT METHODOLOGY | 42 |
| 6.1 Supporting Technology | 42 |
| 6.2 Individual Scout Function | 44 |
| 6.3 Scout Taxonomy | 45 |
| 6.4 Scout Implementation | 47 |
| | |
| CHAPTER 7. SCOUT CONSTRUCTION | 55 |
| 7.1 Scoutmaster | 55 |
| 7.2 Broadcaster | 59 |
| 7.3 Melee | 63 |
| CHAPTER 8. TESTS AND RESULTS | 65 |
| 8.1 Setup of document bases | 65 |
| 8.2 Scout Setup | 66 |
| 8.3 Scoutmaster performance results | 67 |
| 8.4 Broadcaster performance results | 79 |
| 8.5 Melee performance results | 86 |
| CHAPTER 9. CONTRIBUTION AND FUTURE RESEARCH | 93 |
| 9.1 Contribution | 93 |
| 9.2 Future Research | 94 |
| BIBLIOGRAPHY | 99 |

ACKNOWLEDGEMENTS

I would like to thank my mother, Patricia Rehder, for instilling in me, at an early age, a love of learning that has sustained me through this endeavour.

The author deeply appreciates the time, support, and ideas given by Dr. William L. Bynum, my thesis advisor. Thanks are also given to those willing to serve on my committee: Stefan Feyock, Phil Kearns, Richard Prosl, and George Rublein.

Gratitude is owed the NASA Langley Research Center, the Vehicle Analysis Branch, and the Computational AeroSciences Office for allowing me the opportunity to continue my education and for providing superior computer facilities.

LIST OF TABLES

| | Page |
|--|------|
| Table 1. Statistical characterization of document-to-document similarity | 33 |
| Table 2. Characteristics of test document bases | 65 |
| Table 3. ScoutMaster, Effect of machine type, BaseA, single scout, w=0.6, topic=14 | 68 |
| Table 4. ScoutMaster, Effect of topic selection, BaseA, single scout, w=0.6 | 70 |
| Table 5. ScoutMaster, Effect of weighting, single scout, BaseA, topic=14 | 71 |
| Table 6. Scoutmaster, Effect of number of scouts, BaseA, topic=14, weight=0.6 | 72 |
| Table 7. Scoutmaster, Effect of number of scouts, BaseB, topic = 14, weighting = 0.95 | 76 |
| Table 8. Scoutmaster, Effect of number of scouts, BaseA, topic = 14, weighting = 0.95 | 76 |
| Table 9. Document parsing, no links, BaseA, topic =14 | 77 |
| Table 10. Broadcaster, Effect of number of scouts, BaseA, topic = 14, weight = 0.6 | 81 |
| Table 11. Broadcaster messages, BaseA, topic = 14, weight = 0.6 | 83 |
| Table 12. Broadcaster, Effect of number of scouts, BaseB, topic = 14, weight = 0.6 | 85 |
| Table 13. Melee, Effect of number of scouts, BaseA, topic = 14, weight = 0.6 | 86 |
| Table 14. Melee, Message activity, BaseA, topic = 14, weight = 0.6 | 87 |
| Table 15. Melee, Effect of number of scouts, BaseB, topic = 14, weight = 0.6 | 90 |
| Table 16. All scout types, Comparison of results, topic=14, weight=0.6; | 90 |

LIST OF FIGURES

| | Page |
|--|------|
| Figure 1. Similarity characterizations. | 24 |
| Figure 2. Excerpt from document DOC0004.html around paragraph 3. | 39 |
| Figure 3. An excerpt of DOC0004.html around paragraph 8. | 40 |
| Figure 4. Excerpt from DOC0017.html showing return link to DOC0004.html | 41 |
| Figure 5. Screenshot of a scout applet conversing with the AMR | 52 |
| Figure 6. Screenshot of a scout applet conversing with the ScoutMaster | 53 |
| Figure 7. Example log file generated by a scout. | 54 |
| Figure 8. Scoutmaster. Effect of number of scouts on efficiency, BaseA, topic=14, w=0.6 | 73 |
| Figure 9. Scoutmaster. Messages sent by each scout, number of scouts=10, Base A, topic 14, weight=0.6. | 74 |
| Figure 10. Scoutmaster. Effect of number of scouts on speedup, all cases. | 78 |
| Figure 11. Scoutmaster. Degradation of speedup improvement with message rate, all cases. | 79 |
| Figure 12. Broadcaster, Effect of number of scouts on message traffic, BaseA, topic=14, weight=0.6. | 82 |
| Figure 13. Melee, Effect of number of scouts on message traffic, BaseA, topic=14, weight=0.6. | 89 |

ABSTRACT

A new concept for information storage and retrieval is proposed that links chunks of information within and among documents based on semantic relationships and uses those connections to efficiently retrieve all the information that closely matches the user's request. The storage method is *semantic hypertext*, in which conventional hypertext links are enriched with semantic information that includes the strength and type of the relationship between the chunks of information being linked. A retrieval method was devised in which a set of cooperating software agents, called *scouts*, traverse the connections simultaneously searching for requested information. By communicating with each other and a central controller to coordinate the search, the scouts are able to achieve high recall and high precision and perform extremely efficiently.

An attempt to develop a document base connected by semantic hypertext is described. Because of the difficulties encountered in the attempt, it was concluded that there is no satisfactory method for automatic generation of semantic hypertext from real documents. For this reason, the collection of semantically linked documents used in this research was generated synthetically. The constructed document base has many of the characteristics desired in a genuine semantically linked document base.

A Java-based agent framework used to develop three types of software scouts. In the simplest scout implementation, **Scoutmaster**, the paths of the scouts through the document base were specified by a central controller. The only task of each scout was to follow the links specified by the central controller. In the next level of autonomy, **Broadcaster**, the controller was used strictly as a conduit for scouts to exchange messages. The controller received information from the scouts and immediately broadcast it to all of the other scouts to use in determining their next moves. In the final implementation, **Melee**, the central controller was used only to inaugurate the scout searches. After initialization, the scouts broadcast their own messages to all the other scouts.

Experiments were performed to test the ability of the scouts to find information in two synthetically created document sets. All scout types were able to find all of specified information, i.e. high recall, without searching many documents that did not contain the information, i. e. high precision. Using groups of scouts, the best time to search document sets with up to 3000 documents and 2.5 million links was about thirty seconds.

Using single scouts, the Scoutmaster method took the most time, because the scout had to ask the Scoutmaster for its next move and wait for a response. However, this method benefitted the most from the use of additional scouts in a coordinated search. The Broadcaster and Melee types were faster than the Scoutmaster for low numbers of scouts, but did not improve as much with higher numbers of scouts. The message traffic for the Scoutmaster experiments was constant with the number of scouts. For the Broadcaster, the message traffic increased linearly with the number of scouts. For the Melee, the increase in traffic was greater than linear. In the latter two cases, the large increases in message traffic limited the number of scouts to ten.

SEMANTIC SOFTWARE SCOUTS FOR INFORMATION RETRIEVAL

CHAPTER 1

INTRODUCTION

Effective information retrieval (IR) has been a long sought after goal ever since computers became widely used. While great progress was made in early IR research, the last two decades has seen only incremental improvement and fine tuning of methods. There is a tension between finding all documents related to a user's query, termed *recall*, and retrieving only documents that are related to the query, called *precision*. In trying to improve both of these parameters, advancement in IR methods in general seems to have reached a plateau.

In the current research a new concept for information storage and retrieval is proposed that links chunks of information within and among documents based on semantic relationships and uses those connections to efficiently retrieve all the information that closely matches the user's request. The storage method is *semantic hypertext*, in which conventional hypertext links are enriched with semantic information that includes the strength and type of the relationship between the chunks of information being linked. A retrieval method was devised in which a set of cooperating software agents, called *scouts*, traverse the connections simultaneously searching for requested information. By communicating with each other and a central controller to coordinate the search, the scouts are able to achieve high recall and high precision and perform extremely efficiently.

Although research by others has led to methods for retrieving information through hypertext and to techniques for using parallel computing to speed up indexing in conventional IR, no other known research uses parallel techniques and semantic hypertext links

together to speed up hypertext searching.

It is useful at this point to compare the methods developed in this research to the most familiar use of hypertext, the World Wide Web (WWW). Documents on the web are connected through one-way hypertext links that are usually intended to provide relationships between source and destination documents. These links are generated by the authors of the documents according to whatever logic they feel is appropriate. As a result, the links may be arbitrary and the relationships obscure. Also, these links point to entire documents which then must be searched to find the desired information.

The search engines available on the web use programs commonly called web-crawlers to find documents. However, these programs do not traverse hypertext links in response to a user query. They traverse the links simply to provide path names to documents and excerpts of the documents to a facility that indexes the document excerpts using conventional IR techniques. These techniques tend to sacrifice precision to attain high recall.

Chapter 2 provides motivation and background for this research. Chapter 3 describes previous research by others that is related to this dissertation. The semantic hypertext concept is described in Chapter 4 as well as methods proposed by others for automatic or semi-automatic generation of semantic hypertext links.

In Chapter 5 an attempt to develop a document base connected by semantic hypertext is described. Because of the difficulties encountered in the attempt, it was concluded that there is no satisfactory method for automatic generation of semantic hypertext from real documents. For this reason, the collection of semantically linked documents used in this research was generated synthetically. The constructed document base has many of the

characteristics desired in a genuine semantically linked document base. Chapter 5 describes the process used to generate the document base.

Chapter 6 describes the methodology and the Java-based JATLite agent framework used to develop the software scouts. The original intent was to implement four different scout types to determine the effects of relative degrees of autonomy on scout performance and network message traffic. In the simplest scout implementation, called **Scoutmaster**, the paths of the scouts through the document base were specified by a central controller. The only task of each scout was to follow the links specified by the central controller. In a modification of the Scoutmaster protocol, called **Blackboard**, the central controller posted the names of already visited documents in a central location that the scouts could check to determine their next move. In the next level of autonomy, called **Broadcaster**, the controller was used strictly as a conduit for scouts to exchange messages. The controller received information from the scouts and immediately broadcast it to all of the other scouts to use in determining their next moves. In the final implementation, called **Melee**, the central controller was used only to inaugurate the scout searches. After initialization, the scouts broadcast their own messages to all the other scouts. When the scouts were implemented, it became apparent that the Scoutmaster and Blackboard types were essentially equivalent so these two protocols were merged. All subsequent research was based on the Scoutmaster, Broadcaster, and Melee protocols.

Chapter 7 discusses the details of the implementations of the scouts for the three protocols. The operations of the scouts and central controllers, the communications protocols, and the JATLite implementations are described. Chapter 8 describes the results of a number of experiments performed to test the ability of the scouts to find information in

two synthetically created document sets, one containing many documents of small size and the other containing fewer documents of substantially larger size. All three scout types were able to find all of specified information, i.e. high recall, without searching many documents that did not contain the information, i. e. high precision. Using groups of scouts, the best time to search document sets with up to 3000 documents and 2.5 million links was about thirty seconds. This value could go no lower because the agent framework employed could not handle the resulting communication congestion.

Using single scouts, the Scoutmaster method took the most time, because the scout had to ask the Scoutmaster for its next move and wait for a response. However, this method benefitted the most from the use of additional scouts in a coordinated search. The Broadcaster and Melee types were faster than the Scoutmaster for low numbers of scouts, but did not improve as much with higher numbers of scouts. The message traffic for the Scoutmaster experiments was constant with the number of scouts. For the Broadcaster, the message traffic increased linearly with the number of scouts. For the Melee, the increase in traffic was greater than linear. In the latter two cases, the large increases in message traffic limited the number of scouts to ten.

Chapter 9 describes the contribution of this research and discusses possible future research.

CHAPTER 2

MOTIVATION AND BACKGROUND

2.1 Motivation

The ubiquitous use of computers has made the creation and storing of vast amounts of knowledge easy. Because of this ease, the rate of information growth, already prodigious, will continue to increase. Most information is now being generated electronically and there are many efforts to put legacy knowledge into electronic form. Someday, virtually all the information created by the human race will be, in some sense, "accessible." One is tempted to imagine that the utopian dream of universal access to universal knowledge is close at hand. It isn't.

While our ability to generate and store information has progressed rapidly, our ability to access it is still relatively primitive. Knowledge is classified and indexed in much the same way as it has been for centuries. Although the computer can deliver the information almost instantly, our methods for locating and displaying it would seem familiar to a medieval monk. We should be able to do better.

2.2 Background

2.2.1 Information retrieval

There has been active research in information storage and retrieval since the 1940's. In principle, the problem is simple. All one has to do is go to where all the documents are, read them, keep the ones that are relevant to the question being asked, and discard the rest. This is perfect retrieval.

Obviously, this solution is not practical. When computers became widely used, it was thought that the documents could be scanned by a machine, which would then compare the query and the documents and select the relevant ones from all of those in storage. Again, practicality quashed this idea. The next phase of research was to automatically index all the documents *a priori* rather than scan all the documents for each query, so that only the index was scanned to satisfy the query. The difficulties are to extract information from the documents to make the index and use the information to determine relevance to the query. Even today, after the efforts of a small army of researchers, these problems remain largely unsolved.[1, p. 3]

To illustrate this point, let us use the example of search engines on the WWW. Submitting a query to one of these engines usually results in the presentation of thousands of documents of which only a small fraction are truly relevant to the query.[2, p. 4] On the positive side, the list will somewhere contain the most relevant documents, and without automatic indexing, there would be no search capability at all.

The most common type of retrieval has been Boolean retrieval, in which a user formulates a query as a Boolean combination of keywords or index terms, that is, terms linked by the relationships AND, OR and NOT. While this type of retrieval is efficient, it tends to be intimidating to the user, there is no control over the number of documents returned, and all retrieved documents are assumed to be of equal relevance to the query.[3] In addition, there is no effective way to indicate relative importance of different parts of the query.

An alternative type of retrieval is to measure the similarity between a query and each document and then rank the documents in terms of this similarity. The advantages to

this method are that the user does not have to specify boolean combinations of keywords, the documents are ranked, and it is possible to take into account weights associated with index terms. For example, the more times a document contains a given term, the more that document is about the concept the term represents, and the fewer documents in which that term appears, the more that term is a good discriminator. Also, relevance feedback from the user can be used to adjust the weights or reformulate the query.

There has recently been increased emphasis on conceptual or knowledge-based information retrieval (IR).[3] The response to a user's query is not simply a document ranking, but is instead an answer generated from a semantic analysis of document texts. Systems performing conceptual IR operate in narrow domains.[4.5] The research in this thesis addresses the need for efficient retrieval from a document base in which semantic analysis has been used to generate hypertext links among the documents.

2.2.2 Hypertext and the World Wide Web

Associative links among documents were first described as a concept in 1945 by Vannevar Bush.[6] His idea was to construct a device, called a "memex," that allowed a user to read documents and tie together information found in those documents. After a time, an individual would construct an elaborate set of links among the documents. These links could be shared with others. The memex device was to use microfiche as a base, but the technology was not advanced enough to actually implement the device. According to Bush, "The human mind. . . operates by association. . . . Man cannot hope fully to duplicate this mental process artificially, but he certainly ought to be able to learn from it. . . . Selection by association, rather than by indexing, may yet be mechanized." [6, p.107]

This insertion of associative links into text was first achieved by Douglas Engle-

bart.[7] In 1968, Englebart demonstrated his “Augment” system to create a hypertext document. In this system, hypertext links were inserted into the text by hand. The Augment system was the forerunner of modern user interaction, being the first use of the mouse for the selection of text.

2.2.3 An example

Suppose, for example, that there is a team that is designing a passenger airplane. Over the years, other groups have studied this kind of airplane. The results of these studies, each of which may have generated many candidate designs, are located in various places such as bookshelves, filing cabinets, and even computers. The current team may not know that the earlier projects existed, or may not have access to the previous results that are scattered around the world. Although the people working on these teams may not realize it, through their efforts over time a body of knowledge exists about the type of airplane. This knowledge base contains the physical, operational, and economic characteristics of various vehicle concepts, sources of data about subsystems, technology readiness of components, funding plans for future technology development, results of trade studies, and more.

It would be desirable for the latest group studying the airplane design to be able to use this knowledge to avoid duplication of effort, compare design strategies, and make executive decisions. For instance, a program manager may want to know the difference in cost between designing the airplane to carry 150 or 200 passengers or the return on investment on a technology program. A design team leader may want to compare the aerodynamic drag of several different wing shapes or learn the effect of using a lighter material for the structure of the vehicle. An individual engineer may need to know the dimensions,

weight, and power consumption of a piece of avionics equipment that is projected to be available after four years of technology development. To answer these questions, knowledge must be drawn from a large, diverse, and distributed group of information sources. In spite of differences in design philosophies and groundrules related information among the sources must somehow be connected.

The World Wide Web could be used in this airplane design example previously described. A first step may be to enforce that any new information generated by study participants contain hypermedia and that links be placed in all of the legacy documents. That would at least make current results accessible to all. Unfortunately, the previously described problems with searching the WWW remain.

Aircraft design is conducted in a relatively narrow domain. Thus, it is a candidate for conceptual retrieval described previously. By using knowledge-based techniques, links could be constructed between chunks of information that are related semantically. While the domain of the knowledge base may be narrow, it may also be very large, so the search through it must be efficient to be useful. One way to achieve the necessary efficiency would be to have a number of software scouts acting in concert and in parallel to follow the semantic links.

CHAPTER 3

RELATIONSHIP TO OTHER RESEARCH

Semantic hypertext linking combines several positive qualities demonstrated by other means of information retrieval. Determining links among related chunks of information is conceptually similar to computing the relevance between a query and a set of documents in traditional IR. Also, semantic hypertext linking, like conventional hypertext, has the characteristic of direct connections between chunks of information; that is, there are no links to irrelevant information. The method avoids both the need to compare a query with every document in the knowledge base and the arbitrary connections of traditional hypertext.

Simultaneous traversing of these links by a number of software scouts makes retrieval of information in even a large system tractable. Of course, this efficiency can only be achieved if the scouts do not duplicate, or even overlap, the search spaces of the other scouts. Such cooperation requires communication. To achieve high performance, the time used in communication must not greatly exceed the time used in the actual processing of individual knowledge chunks. In addition, the content of the messages passed among the scouts must contribute to the ability of the scouts to find information germane to the query.

3.1 Related Research Activities

There are a number of research activities around the U.S. whose relationship to the current project is close enough to warrant further discussion. The government sponsored Digital Libraries Initiative is the major one; there are others in academia and in the com-

mercial world.

3.1.1 Digital Libraries Initiative

The first Digital Library Initiative was a \$24 million, four year program, 1994-1998, funded jointly by National Science Foundation, Advanced Research Projects Agency, and the National Aeronautics and Space Administration (NASA).[8] This activity was performed by six academic/commercial consortia headed by the University of California, Berkeley; University of California, Santa Barbara, Carnegie-Mellon University, University of Illinois at Urbana-Champaign, University of Michigan, and Stanford University are the other participants. Each consortium focused on an aspect of digital libraries. Santa Barbara and Carnegie-Mellon concentrated on specific applications, such as digital video or cartographic libraries, not directly related to the proposed research. The relationship of the other groups will be discussed in more detail.

The Berkeley effort concentrated on collecting the environmental documents that will make up the testbed, improving Optical Character Recognition technology needed to digitize the documents, and extending and merging the capabilities of the POSTGRES database manager, the SQL3 query language, and the Z39.50 information retrieval protocol and a distributed search capability is due shortly.

The Illinois project was geared toward developing a digital library for a university engineering community. The Illinois testbed consists of engineering reports and journal articles in Standard Generalized Markup Language (SGML) format. The initial implementation was Mosaic combined with an SGML viewer and commercial software for full-text search capability. The research was centered around scale and functionality. The Illinois project investigated ways of automatically classifying documents according to conven-

tional schemes, such as Dewey Decimal and Library of Congress. The depth of semantic retrieval methods was to be enhanced. The results will migrate into a next-generation implementation in some unspecified way. There was no description of parallel, coordinated search in any of the Illinois project documentation.

At Michigan, the testbed implementation was based on three classes of software agents. User interface agents interview and receive requests from users and pass them off to mediation agents, which coordinate searches of distinct, networked information sources. The searches are performed by collection agents, which are specialized agents residing at the information sources. It is the task of the collection agents to convert the user's query language into the native query languages of the information sources. The library was partitioned into the collections of information and the conspectus, which comprises abstracted descriptions of the collections. This paradigm is fundamentally different from the parallel coordinated searches by agent clones studied in this thesis.

The Stanford project proposed an ambitious "information bus" architecture that mediates among different clients, sources, and services. The bus is a common set of protocols and representation models for information items that will allow the components of the digital library to interoperate. The "driver" of the bus is the information bus agent engine that provides creators of information services with the analog of a scripting facility. The finding of information is conducted by distributed servers that are analogous to the collection agents used in the Michigan project.

A second Digital Library Initiative began in 1999. In this phase, there are more, but smaller, projects being funded. About 25 projects have been selected, ranging in length from 1 year to 5 years and in funding from \$287,000 to \$5.4 million. Although several of

the projects will investigate methods for knowledge representation, none specifically address construction of hypertext. One project, at Indiana University, proposes the use of a set of distributed Java agents.

3.1.2 Academic Research

An example of academic research is the WebAnts project at Carnegie-Mellon [9], the only example of cooperative information retrieval in a hypertext document base. This project was similar to the current study in some aspects. It featured cooperating, communicating agents that simultaneously traversed standard Web pages looking for those that contained user-selected keywords. One of the principal differences, compared to the present study, is the problem domain: the WebAnts project is aimed strictly at searching and indexing World Wide Web pages. Other differences are the amount of research and the current status of the project. Early work in the WebAnts project concentrated on developing and testing simple protocols for message-passing among the agents. It appears, however, that the WebAnts project is defunct. The Web pages that described the project remained several years without update and have now been removed altogether: WWW search engine queries concerning WebAnts return no results.

At the University of Massachusetts Center for Intelligent Information Retrieval (CIIR), the Agent-Based Architecture project is aimed at developing an agent for gathering information from a large, distributed network of information sources.[10] The information gathering method is based on exploiting the advanced IR and information extraction techniques that are becoming available as a result of the work at CIIR, allowing an agent not only to locate but also to extract necessary information from unstructured textual documents. The Agent-Based Architecture proposal states that the development of

such an agent will be the first step toward building cooperative information-gathering agents that coordinate their activities. The research is just beginning, and there is no provision for using hypertext.

3.1.3 Commercial Research

InfoSleuth is a multimillion dollar, multi-year commercial research project at the Microelectronics and Computer Technology Corporation (MCC) to develop and deploy new technologies for finding information in enterprise-wide knowledge bases.[11] InfoSleuth is based on the existing MCC-developed Carnot technology that uses autonomous agents dispatched to remote sites both to access databases and cooperate to merge resulting information. InfoSleuth will use semantic agents to carry out distributed, coordinated, self-adapting search algorithms. MCC is soliciting for industrial sponsors; the charge is about one million dollars for full participation. The levels of participation, as well as the details of the InfoSleuth development, are sketchy, but there is some information available on Carnot. The Carnot agents appear to be full-blown inference engines specialized to specific rule-based domains. This is a substantially different paradigm from the scouts in this thesis.

CHAPTER 4

SEMANTIC HYPERTEXT

4.1 Knowledge Retrieval with Hypertext

In this research, the methods used for information retrieval differ from those usually found in hypertext retrieval in that they use typed and weighted links and avoid user browsing, and they differ from traditional IR in that they use direct associations among documents.

4.1.1 Comparison of IR and Hypertext

The collection of documents stored and managed by a traditional IR system is usually large and the documents all have the same importance. When a user makes an inquiry against the collection, all documents are potentially equally relevant to the inquiry. Associated with each document is the indexing term structure representing its semantic content. This structure is used to select and retrieve documents during query processing that proceeds linearly through the document set. The index terms themselves constitute access points to the collection. The primary differences in IR systems are the ways in which they determine the index terms and measure similarity.

On the other hand, with a hypertext system the user navigates the collection by means of direct associations or links among the documents. The user has the opportunity to construct dynamically an information path through the document base. The retrieval of information in hypertext documents proceeds in a sequential, though nonlinear, fashion. Hypertext links that appear to be related to the particular query are traversed one at a time.

The documents at the other ends of the links may contain additional links that can also be followed. In contrast to direct search methods, not all of the documents are considered, only those that are the targets of links relevant to the inquiry.

4.1.2 Hypertext Information Retrieval

Most of the work in hypertext information retrieval (HIR) has been devoted to the presentation of new retrieval models based on both navigation and direct search features.[12] An HIR system that combines query-based and browsing-based retrieval methods was presented by Lucarella and Zanzi.[13] Within this model, two networks are maintained. The document network contains the original hypertext documents that are connected with structural links. The concept network contains nodes that represents concepts that appear in the documents. The nodes have links to documents that contain the concepts and links to other related concepts. These links are created using conventional IR similarity determination methods. A user query is matched to a concept node and links are then followed to other concepts and to documents. Of particular interest to the current research is the structure of the concept links. These links are labelled by the name of the relationship that exists between the two connected nodes and a weight is associated with the link to express the strength of the relationship. The document links point to entire documents; the current research makes connections between paragraphs, within and among documents. The typed, weighted link concept was employed in the present research and will be discussed in depth in the next chapter.

4.2 Cooperative hypertext retrieval

A hypertext document base and its corresponding network of links could be very

large. A user navigating through the documents one link at a time may take an unacceptable amount of time to find the desired information. If the computer resources are available, having a number of agents navigate the documents simultaneously on behalf of the user may substantially improve the performance. You may want to insert "time," "quantity," "quality," or something similar here. of a hypertext retrieval system. The requirements of such a cooperative search, as well as some previous attempts by others, are reviewed below.

4.2.1 Hypertext Retrieval on the WWW

Most WWW searches use directories or search engines to satisfy their specific inquiries. Avoid slash combinations wherever possible. In cases where we use "and/or," one or the other will almost always suffice. The search engines employ programs, such as Webcrawler or WebSpider, to access the millions of documents on the WWW by following the hypertext links that these documents contain. These programs are not performing searches, they are simply finding and listing documents and their locations that are then indexed by standard IR methods. A user's query is compared with the index terms. Documents determined to be relevant to the query are displayed in ranked order. The Webcrawler type programs do their work prior to any user query being made.

In the literature, the only example of cooperative information retrieval in a hypertext document base was the aforementioned WebAnts project at Carnegie-Mellon University.

4.2.2 Requirements for Cooperative Retrieval

In order to achieve cooperative information retrieval from a linked document base,

several requirements must be satisfied. Because the scouts act in parallel, they, as well as the documents, reside on a distributed network of computers. The scouts have permission to execute on the various nodes of the network and have access to the documents. The locations of the documents, both machine and path names, are made available to the scouts. Also, for communication, the machine names and port numbers for the scouts are made known to all scouts and to a central controller.

The scouts require certain information in order to perform their tasks. To avoid duplication of effort, the most basic requirement is the knowledge of whether a specific link already has been traversed. The scouts detect links in a document and determine the target text and the type and strength of the connection. The scouts keep a history of traversed links in case backtracking is necessary.

4.3 Construction of Semantic Hypertext

In conventional information retrieval, entire documents are delivered in response to a user's query. If the retrieved documents are large, it may be difficult for the user to find the specific desired information. Although a small amount of information may be relevant, even crucial, to a user, the fact that it takes up only a small part of the document may mask its relevance and cause the document to be ranked low on the relevance list. For these reasons, it is beneficial to break documents into small chunks. For the current study, paragraphs within documents were used as the information chunks.

The use of small chunks of information is ideally suited to hypertext. Because both ends of hypertext links can be represented by an arbitrary amount of text, a chunk of information in one document, whether it is a word, phrase, sentence, or paragraph, can be linked directly to another entire chunk in another document. With hypertext, these links

can be one-to-one, one-to-many (but not on the WWW), many-to-one, bi-directional (also not on the WWW), and distributed over a network. Because the chunks of information on each end of the link are relatively small, the semantic relationship between them can easily be characterized. Once the semantic relationship is determined, it can be represented by assigning a type and a weight to the link.

4.4 Semantic Characterization

Methods for determining similarity between text items have been investigated since the beginnings of information retrieval. Some recent attempts have been made to automatically characterize the similarity in terms of semantics. [14, 15] Even more recent studies have attempted to use the semantic relationships to generate hypertext links.[16, 17]

4.4.1 Text Analysis

Two terms in IR research are used to characterize the quality of a retrieval. One is *recall*, which is the ratio of the number of relevant documents retrieved to the total number of relevant documents in the collection, i. e., a measure of how good the information retrieval process is at finding candidate relevant documents. The other is *precision*, the ratio of the number of relevant documents retrieved to the total number of retrieved documents, a measure of the ability not to retrieve irrelevant documents. By these definitions, commonly used automatic indexing methods, such as the SMART system[18], tend to have high recall and low precision. It is generally true that methods that have high recall have low precision and those that have high precision also have low recall. Much of the effort in IR research is devoted to improving the trade-off between precision and recall.

Some popular methods for finding similarities between documents or parts of documents use the vector space model.[14] In that approach, documents are assumed to exist as vectors in a t -dimensional space, where there are t terms used to represent documents. A vector's magnitude in any particular dimension depends on the importance of the corresponding term in the document. Two documents are determined to be similar if their corresponding vectors lie near each other in the t -dimensional space. Typically, the similarity of two documents is measured by the cosine of the angle between document vectors.

The key is to determine the importance of the terms in a document. An obvious way would be to count the number of times a term appears in the text. This term-frequency approach mimics the conventional index, where all of the occurrences of a term are listed in the index. Two problems with this approach are that documents having many common terms may not actually be similar and that large documents are almost always ranked higher than small ones. Using only term-frequency weighting will result in high recall and low precision. More factors must be taken into account when weighting the terms.

One method for term-weighting has emerged as somewhat of a standard. Called TFIDF (for term frequency, inverse document frequency), this method attempts to increase the precision without losing the high recall of term frequency alone.[15] In TFIDF, the weight of term j in document i is

$$w_{ij} = \frac{tf_{ij} \cdot \log(N/n_j)}{\sqrt{\sum_{k=1}^t (tf_{ik} \cdot \log(N/n_k))^2}}$$

where tf_{ij} is the frequency of term j in document i , n_j is the number of documents containing term j , and N is the total number of documents in the collection.[14, p. 976] The numerator is product of the term frequency and the inverse document frequency (the TFIDF). The denominator provides a cosine normalization to the vector so that large documents do not overshadow small documents. The concept is that a term that occurs frequently but occurs in only a few documents is a good term to use to determine relevance. When a common term appears many times in many documents, it ceases to be a discriminator between documents. In this formulation, such a term would be given a low weighting.

An extension of the vector space model is *latent semantic indexing* (LSI), a technique in which the dependencies between terms and between documents are explicitly taken into account.[19] Statistical techniques are used to estimate the latent structure in the pattern of word usage across documents. The latent semantics are then used to match documents that have similarity even though, due to factors like synonyms, they may share few actual terms. LSI uses singular-value decomposition, a technique closely related to factor analysis, to decompose a large term-document matrix into a smaller set of orthogonal factors from which the original matrix can be approximated by linear combination. Since the number of factors is much smaller than the number of unique terms, words will not be independent. The idea is that this dependency reflects the latent semantics in the document collection. Two different terms used in the same context will have similar vectors in the reduced LSI representation. Although LSI has shown up to 30% better performance than more conventional vector space methods, in the annual TREC demonstration, its performance was average or slightly below.[20]

Other, more empirical, techniques have been developed to improve the determination of similarity among documents. For example, NASA has developed a thesaurus [21] that explicitly states similarities among various terms used by NASA researchers.

4.4.2 Similarity and Semantics

Although the vector space model and its kin are widely used, very little improvement in their performance has been made in recent years. These methods, however, represent the state of the art in determining similarity among bodies of text and have been used in embryonic attempts at automatic hypertext construction.

Homonyms and coincidental use of common terms can cause two documents to have an incorrectly high similarity. In such a case, attention must be paid to the usage of the terms, i.e., their context. An approach for analyzing context has been developed by Allan and Salton.[15]

First, documents are compared using the TFIDF method described above. If a pair of documents has sufficiently high similarity, the documents are then broken into smaller portions, such as sections, paragraphs, or sentences. If these smaller portions are not found to be similar, the overall document similarity is judged to be the result of coincidence, because it is unlikely that two documents that have no similar portions will cover the same topic. If parts of a document are similar, further analysis is performed to determine the nature of the similarity.

In Allan [16], a novel graphical technique is presented. Two documents are represented by arcs and the smaller portions of the documents are denoted by segments within the arcs. Similar portions of the documents are connected with lines. Figure 1 shows how this technique is used to determine the characteristics of the connections.

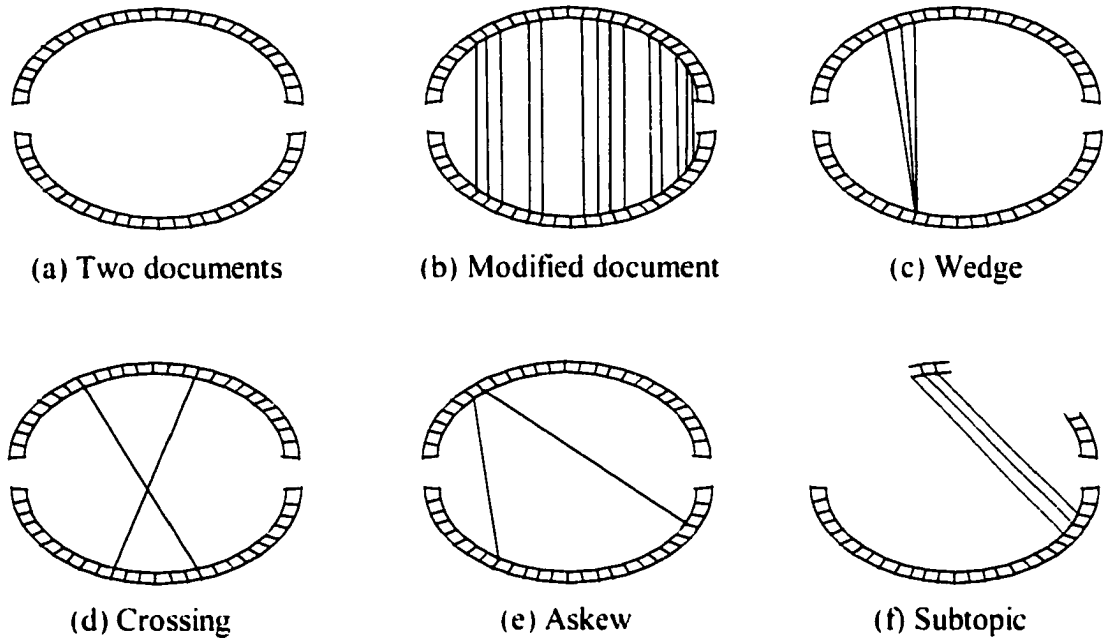


Figure 1. Similarity characterizations.

In the figure, (a) shows how two documents are represented by arcs that are divided into segments corresponding to the parts (e. g., paragraphs) of the documents. In (b) through (f), segments that are determined to be very similar are connected by lines. For example, two identical documents would show as a series of parallel lines connecting all the segments. Part (b) of the figure illustrates the case where one document is a modification of another in that it contains a number of parallel lines. The appearance of a wedge, as in (c), indicates that part of one document is a summary, or conversely, an expansion of part of the other. In (d), the lines cross, possibly indicating a difference of emphasis of a topic in the documents. Another indication of difference in emphasis is shown in (e), where two topics that are close together in one document are separated in the other. Part (f) of the figure illustrates how a subtopic from one document can appear itself as a small

document.

4.4.3 Automatic Generation of Semantic Hypertext

It would be ideal if a large number of documents could be analyzed and links could automatically be generated among the documents. These links would be typed based on the semantic relationships between the texts at either end of the links. Thus, a large knowledge base could be searched with semantic scouts that have knowledge of the link types. Several attempts have been made toward this goal and are described below.

The various manifestations of link characteristics described above are used by Allan [16] to develop the following set of link types.

Revision links indicate ancestor and descendent relationships among texts. They are those in which the two documents have many linked paragraphs in the same order as illustrated by Figure 1(b).

Summary and **expansion** links are inverses of one another. The summary link begins at the discussion of a topic and has as its destination a condensed version of that same topic, whereas the expansion link begins with the synopsis of a topic and ends at a more developed version. The wedge in Figure 1(c) illustrates the summary/expansion link.

Equivalence links represent strongly related discussions of the same topic. In that case there are many links between the documents, but the subtopics may not be in the same order as with the revision link.

Contrast links occur when documents have strong connections among small parts but are otherwise not strongly related.

Tangent links indicate that texts are globally similar but locally dissimilar, as

with a poem about clouds and a technical description of how clouds are formed.

Comparison links are used when documents are similar but fit into none of the other categories.

In Allan's work the similarities among documents and parts of documents were calculated with the SMART system [18], and the semantic characterizations were made with the graphical techniques. Unfortunately, although a technique for identifying and characterizing links was developed, the characterization in the end was performed by the human eye looking at the diagrams. No links were ever placed in the documents, either by hand or automatically, and no testing of link navigation was done.

In an advance over the text analysis in Allan's work, Green [17] describes a method in which hypertext links are constructed with of lexical chaining, a technique for extracting sets of words that occur in texts. A lexical chain is a set of semantically related words in a text. The lexical chains in a document tend to delineate the parts of the text that discuss the same topic. Green used the WordNet [22] database of synonym sets to determine the semantic relationships.

Once the lexical chains are extracted, they are compared with paragraphs of text and the fractions of the content words in a paragraph that are in each chain are calculated, resulting in a chain density vector. After these vectors are weighted and normalized, the similarity of the paragraphs is determined by computing the similarity between the corresponding chain density vectors by a variety of similarity functions. Two paragraphs are linked if the similarity between their chain density vectors is substantially greater than the mean similarity across the entire text. To generate links between documents, Green uses a

vector space model similar to that employed by Allan in the SMART program.

Green's method efficiently and automatically generates hypertext links in a large document base. However, unlike Allan's work, there is no attempt to characterize the nature of the links. While semantics are used to determine similarity, they are not used to type the links. Green compared his technique based on lexical chaining with a conventional method that uses the vector space model provided in a software package called *Managing Gigabytes (MG)*. [23] For the purposes of Green's research, MG is essentially the same as SMART, except that it automatically breaks the documents into small chunks, i. e., paragraphs. In user testing, the test subjects were able to navigate the linked documents slightly more efficiently with the links generated by lexical chaining.

Because the research described in this dissertation depends on having a semantically linked document base, it was hoped that methods such as those developed by Allan and Green could be used to generate the links among the documents. However, the capability of SMART to divide documents into chunks is neither documented nor supported and Allan's method does not include the construction of automatic linking. Green's implementation of his method operates only on documents that exist in a specific format that was not utilized in the documents available to the current study. The method applied herein and described later in this chapter borrows concepts from Allan and Green and uses a combination of SMART, MG, and manual editing to produce the required, semantically linked document base.

CHAPTER 5

DOCUMENT BASE CONSTRUCTION

The main objective of the current study was to investigate the cooperation of a number of software scouts acting simultaneously to retrieve information from a semantically linked document base. Based on the literature just described [16, 17], it was assumed that such a document base existed or that techniques existed to automatically create one. However, because of incomplete or nonexistent implementation, no capability for generating a semantically- linked document base existed. Therefore, though unintended, the development of such a capability temporarily became a major element of this study.

This effort, described in sections 5.1 through 5.5, was also unsuccessful. The reasons that generating a semantically- linked document base proved difficult are described in section 5.6. The remainder of the chapter, section 5.7, describes the development of a capability to synthetically generate the document bases that were used by the scouts in their searches during the testing phase of the research.

5.1 Document Source

5.1.1 Standard Collections

To facilitate IR research over the years, a number of standard document collections have been generated. The most prominent was assembled by the National Institute of Standards (NIST). The total collection, called TREC [24], is divided into subcollections that contain newspaper article archives, aerospace abstracts, and journal articles. New subcollections are added every year, and after five years the total collection consists of thou-

sands of compressed documents and is about 2.5 gigabytes in size. The documents come in a variety of formats, the most popular being SGML. The archive of articles from the Communications of the Association for Computing Machinery (CACM) [25] is another well-known collection. The CACM collection consists of a large number of articles archived from a set of journals.

5.1.2 Chosen Collection

The standard collections have a broad coverage of topics. For the purpose of an initial implementation of semantic scouts, a more restricted domain of knowledge was more suitable. Therefore a smaller, more specialized source of documents was utilized. The collection of NASA reports from the Langley Technical Report Server (LTRS) [26] consists of technical papers, memoranda, and contractor reports published by the NASA Langley Research Center from 1961–1999.

The entire collection is about 2000 documents in size and covers a range of topics. To restrict the domain even further, only the documents that were returned by searching on the keyword 'aircraft' were used. All of the documents on the LTRS are stored in Postscript format, which is very difficult to handle by IR indexing schemes. A utility program, called prescript, was obtained from the Managing Gigabytes project [23] in New Zealand. This program converts Postscript to either ASCII or HTML text. These two formats are identical except that with the HTML some header tags are placed in the document as well as a tag to denote each paragraph. For about half of the documents, the prescript program aborted with an internal error, so many documents were not used. The total number of documents actually used in the document base was 40.

5.2 Link Taxonomy

The hypertext links placed in the document base for the present study are typed and weighted. The type definitions are adapted from Allan [16], with the principal differences being that Allan's definitions pertain to entire documents rather than paragraphs and the addition of new types. The type definitions used herein are as follows:

Identity - The two paragraphs are essentially identical or at least so similar that to have one means to have the other. It is expected that identity links will not be followed because they offer no new information and that any further links located in the target will also be in the source. Allan had equivalence and revision link types that are conceptually similar to this identity type. However, Allan's types accounted for variations in the order in which components of the documents appeared, an idea not applicable when paragraphs are used.

Expansion-Summary - These links are inverses and will always occur together, pointing in opposite directions. The information pointed to by the expansion type link is larger than the source of the link and contains similar information to the source link. It also contains additional, but related, information. The reverse is true for the summary link type.

Complement - Often, paragraphs exhibit strong similarity but upon inspection are seen to be far from identical. Also, if they contain about the same amount of information, their relationship cannot be considered as expansion-summary. In this case, the two paragraphs may be discussing two aspects of the same concept. Paragraphs exhibiting this relationship are connected with complement type links. The complement link is a new type; there was no counterpart

defined by Allan.

Contrast - Paragraphs may have strong similarities among smaller parts, but be otherwise not strongly related. Thus, if two information chunks share some similarity between their parts, the remainder of the chunks will be analyzed. If then no further related information is found, the link is of contrast type.

Tangent - These links indicate that texts share similar terms, but with different meanings or contexts. Although this type of link may never be followed, it is included to allow some experimentation.

5.3 Similarity Determination

The relevance among paragraphs in the document base was determined by means of a conventional vector space model. Two available programs were used to calculate paragraph similarity: the SMART program from Cornell [18] and the MG program from a consortium headed by the University of Canterbury in New Zealand.[23] The programs give numerical relevance scores between a segment of text, called the query, and a set of documents and were used to give indications which paragraphs might be semantically linked in the document base. The values provided by the programs were evaluated, and a determination was then made as to whether a link should exist and what its type and weight would be.

To reduce the work in creating the document links, the process was broken into three phases. The first two phases used automatic indexing and the last phase used manual editing. First, each document was compared with the rest of the documents using the SMART program, and a list of the documents most similar to the first were listed. A typical output from SMART follows. "Num" is the document number, "Sim" is the value of

the similarity parameter, with 1.0 meaning two documents are exactly the same (the first line is the result of a document being compared to itself), and "Title" is the title of the document being analyzed.

| Num | Sim | Title |
|-----|------|---|
| 37 | 1.00 | A LABORATORY STUDY OF SUBJECTIVE ANNOYANCE RESPONSE TO SONIC |
| 32 | 0.50 | Abstract Economic viability of a supersonic commercial tran |
| 9 | 0.32 | FLUTTER CLEARANCE FLIGHT TESTS OF AN OV-10A AIRPLANE MODIFIED F |
| 52 | 0.30 | XV-15 Tiltrotor Low Noise Terminal Area Operations David A. Con |
| 18 | 0.24 | ROTONET Primer Devon Prichard Lockheed Engineering & Sciences C |
| 13 | 0.23 | FLIGHT CREW TASK MANAGEMENT IN NON-NORMAL SITUATIONS Paul C. Sc |
| 25 | 0.20 | A New Approach to Aircraft Robust Performance Analysis Irene M. |
| 49 | 0.19 | NASA Contractor Report 201607 DOT/FAA/ND-96/2 Performance of th |
| 26 | 0.19 | NASA WAKE VORTEX RESEARCH FOR AIRCRAFT SPACING R. Brad Perry*, |

The results of a cursory statistical characterization of the document-to-document comparison are shown in Table 1. The first column is an identifier for each document that was successfully processed from the LTRS. Each document in the table was compared to all of the other documents and those that yielded a value of Sim greater than 0.25 were considered similar enough for further investigation. The sum of all the Sim values for the documents above the 0.25 threshold appears in the second column of the table, and the documents are sorted by that Sim value. Because a straight sum was used, a high value may be the result of a document being slightly similar to many other documents or very similar to only a few. To provide further insight the numbers of documents that occur within ranges of Sim values are shown in columns 4 through 11. As an example, the document with the largest value of total similarity had no Sim value greater than 0.39. In other words, document 25 was the most similar to the entire body of documents, but was not very similar to any one document. In contrast, document 56 was rated lower in total similarity, but was closely similar to several individual documents. Although not done thoroughly in this study, this type of analysis may be useful in characterizing large document collections.

Table 1: Statistical characterization of document-to-document similarity

| Doc No | Sim | .25-.29 | .30-.39 | .40-.49 | .50-.59 | .60-.69 | .70-.79 | .80-.89 | .90-.99 |
|--------|------|---------|---------|---------|---------|---------|---------|---------|---------|
| 25 | 7.28 | 8 | 15 | | | | | | |
| 41 | 6.95 | 11 | 9 | 2 | | | | | |
| 14 | 6.65 | 7 | 10 | 2 | | | | 1 | |
| 26 | 6.54 | 7 | 10 | 1 | | | | 1 | |
| 56 | 6.17 | 9 | 6 | 1 | 1 | | | 1 | |
| 58 | 5.97 | 10 | 9 | 1 | | | | | |
| 5 | 5.82 | 6 | 9 | 0 | 1 | 1 | | | |
| 33 | 5.74 | 10 | 7 | 2 | | | | | |
| 45 | 5.43 | 8 | 9 | 1 | | | | | |
| 52 | 5.38 | 5 | 12 | | | | | | |
| 58 | 4.86 | 6 | 9 | 1 | | | | | |
| 22 | 4.77 | 13 | 3 | 1 | | | | | |
| 44 | 4.52 | 7 | 5 | 1 | | | | | |
| 7 | 4.49 | 8 | 4 | 2 | | | | | |
| 47 | 4.28 | 7 | 4 | 1 | | 1 | | | |
| 27 | 3.64 | 3 | 3 | 1 | 1 | | | | |
| 18 | 3.58 | 7 | 4 | 1 | | | | | |
| 38 | 3.58 | 7 | 4 | 1 | | | | | |
| 9 | 3.20 | 5 | 6 | | | | | | |
| 39 | 3.17 | 3 | 6 | | | | | | |
| 2 | 3.15 | 2 | 7 | | | | | | |
| 62 | 3.14 | 5 | 3 | 1 | | | | | |
| 68 | 3.14 | 7 | 4 | | | | | | |
| 43 | 2.97 | 3 | 3 | | 2 | | | | |
| 36 | 2.71 | 3 | 3 | 2 | | | | | |
| 49 | 2.66 | 4 | 5 | | | | | | |
| 13 | 2.59 | 4 | 2 | 2 | | | | | |
| 42 | 2.58 | 4 | 3 | 1 | | | | | |
| 6 | 2.42 | 5 | 2 | 1 | | | | | |
| 48 | 2.31 | 6 | 1 | 1 | | | | | |
| 1 | 2.12 | 4 | | 1 | 1 | | | | |
| 35 | 2.04 | 5 | 2 | | | | | | |
| 38 | 2.02 | 6 | | 1 | | | | | |
| 29 | 2.02 | 2 | 3 | 1 | | | | | |

In the next step, paragraphs in each document were compared with all of the paragraphs of the documents that were most similar to the one containing the selected paragraph, i. e., those documents with a value of Sim greater than 0.25. To reduce the time-consuming effort of comparing thousands of paragraphs, no comparisons were made between paragraphs of dissimilar documents. An example of the output of the paragraph-to-paragraph comparison follows.

| Num | Sim | Title |
|-----|------|---|
| 52 | 1.00 | There are several forms of the FW-H equation to be found in the |
| 56 | 0.48 | The effects of compressibility on the acoustic field of a rotor |
| 45 | 0.42 | The IBA(LIFT), IBA(MOMENT), and IBL(DRAG) tables should be plot |
| 21 | 0.35 | Acoustic problems are typically described in terms of the sour |
| 60 | 0.33 | There are many mechanisms which contribute broadband noise to r |
| 25 | 0.27 | Waves traveling in a fluid can also be subject to refraction (a |
| 22 | 0.27 | The behavior of sound waves in a fluid can be modeled using the |
| 63 | 0.25 | As sound waves pass through the atmosphere, there are losses in |
| 23 | 0.25 | Various forms of the linear governing equation for sound have b |
| 48 | 0.24 | Some aerodynamic phenomena on rotorcraft are : unsteady self- |
| 55 | 0.23 | Loading noise effects are strongest out of plane with the rotor |
| 57 | 0.21 | Blade-Vortex Interaction (BVI) is an impulsive aerodynamic phen |
| 49 | 0.19 | The Lighthill acoustic analogy is used to separate the aerodyna |
| 86 | 0.16 | The rotor performance modules create airloads tables for the ro |
| 20 | 0.16 | For practical purposes, unsteady pressure disturbances in a com |

The similarity values obtained from the indexing programs were then used to decide the linking between paragraphs. Every pair of paragraphs deemed similar by the programs was evaluated manually to determine the similarity with respect to type and strength. For example, the following two typical paragraphs were calculated to be very similar.

The sensitivity of the grid with respect to the vector of design parameters can be obtained by direct differentiation of the grid equations. As a consequence of using algebraic grid generation technique in which the boundary grid has the dominant effect on the interior grid, the boundary grid sensitivity coefficient would also be essential in influencing the interior grid sensitivity coefficient. Therefore, evaluation of the surface grid sensitivity coefficients are the most important part of the analysis and are directly dependent on the surface parameterization.

Among two major classes of grid generation systems (Algebraic, Differential), algebraic grid generation systems are ideally suited for achieving this objective. The explicit formulation, resulting in a fast and suitable grid, enables direct differentiation of grid coordi-

nates with respect to design parameters. The underlying effort here is to avoid the time consuming and costly numerical differentiation. In addition, the analytical derivatives are exact, a desirable feature for sensitivity analysis.

The first paragraph highlights the importance of one component of an analysis technique. The second paragraph explains the characteristics of a particular implementation of the analysis component. At first glance, the second paragraph could be viewed as an expansion of the first in that it gives further information about a concept introduced in the first paragraph. On the other hand, the first paragraph cannot be considered a summary of the second, and the paragraphs contain about the same amount of information. It is questionable whether the two paragraphs, although quite deserving of a strong link between them, display a reciprocal relationship as in the case of the summary-expansion link type described earlier. Therefore, the complement type link is the appropriate choice.

5.4 Link Syntax

Once the character of the similarity was decided, a link was created by inserting an HTML tag with appropriate attributes into the text of the two paragraphs. For the use of the semantic scouts the following attributes were added to the standard <A> tag of the HTML 4.0 standard [27]:

TYPE = I | E | S | M | C | T where I, E, S, M, C, T mean Identity, Expansion, Summary, Complement, Contrast, and Tangent, respectively.

WEIGHT = x where x is a floating point number between 0.0 and 1.0.

As an example, an expansion-summary link pair would look like this:

In document 1

```
...
<A NAME="PAR013" HREF="DOC0002#PAR026" TYPE="E"
WEIGHT=0.68>
the text of the paragraph </A>
...
```

In document 2

```
...
<A NAME="PAR026" HREF="DOC0001#PAR013" TYPE="S"
WEIGHT=>0.68>
the text of the paragraph </A>
....
```

Each paragraph in a document is labelled via the <A> tag as $PARA_n$, where n is the number of the paragraph in the document. In the above example, paragraph 26 of document 2 is an expansion of the topic in paragraph 13 of document 1. Conversely, paragraph 13 of document 1 is a summary of paragraph 26 of document 2.

Because the tag syntax is very close to standard HTML, initial <A> tags were inserted with Composer, the HTML editor built into Netscape Communicator. Then the values of nonstandard attributes were added with a common text editor.

5.5 Document Base Statistics

The document base devised for scout testing is characterized by the following:

Number of documents = 40

Number of paragraphs = about 3000

Total number of potential links = 619

Total size = 4 Megabytes

5.6 The Problem with Typed Links

While calculating a value of link strength between two paragraphs was relatively straightforward with the similarity calculations of the vector space model, determining the type of the link was extremely burdensome, requiring that each candidate be manually read and evaluated. The reading was very tedious and time-consuming and the characterizations of the links were often far from obvious. In addition, in some preliminary experiments, the scouts were not able to use the link type information to any great advantage.

The problems are that the categories of link types are too broad to be useful and user queries often do not reflect the kinds of relationships implied by the link types. Allowing only a few link types makes the task of automating link construction easier, but forcing every link to be one of only several types means there will be many links of a given type which will lead to poor retrieval precision. Providing for a large number of link types will greatly increase the difficulty of automatic link generation. If semantic linking is ever to prove useful in retrieving information from hypertext, the fundamental problem of automatic link typing must be solved. Developing new methods of analyzing documents and automatically generating semantic hypertext links is a fruitful area for further research. Conventional methods for determining text similarity will probably be inadequate for this task. Other approaches will be discussed in Chapter 9.

Fortunately, tests were run after only a few documents had been marked up with link types. Adding types to all the links in the document base was abandoned and a simulated document base was created. Although not having a group of actual semantically-linked documents is a disappointment, using a simulated document base allows more documents and a larger set of link types. Since the principal object of this research is to deter-

mine the efficiency of parallel scout execution and the characteristics of various communication methods, and because the scouts are not analyzing the text but only following links, a large set of documents consisting only of randomly generating typed links provided an excellent testbed for the scout experiments.

5.7 Construction of a simulated document base

A capability was developed for generating an arbitrarily large document base in which the number of paragraphs in each document, the type and weighting of each link, and the topic of each paragraph are randomly selected from normal distributions between minima and maxima. The generated documents contain links in the same modified HTML described above, with the only text being an integer that represents the topic of the paragraph. All links are one of a pair connecting two paragraphs that have the same topic. In other words, every link is symmetrical, using the idea that if paragraph A is similar to paragraph B, then B is also similar to A. The idea of link symmetry seems almost obvious but is seldom, if ever, used in conventional hypertext documents.

The program first generates file names for the number of documents specified. For each document, the number of paragraphs for that document is randomly selected, and for each paragraph a topic value, also randomly selected, is assigned. Then, each paragraph in each document is compared to every other paragraph in every other document and where the two topics are the same a pair of links is inserted, one in each paragraph that points to the other. The number of links in any paragraph depends on the random number of paragraphs with the same topic. Since the subject comparison is done between paragraphs, some of the links generated may be between paragraphs within the same document.

During the generation of the document base, all paragraphs that have the same

topic were connected by links, although the link weights and types varied. If this full connectivity in the document base did not exist, then during search there would be no way to know that all of the information had been found. In that case, the only way to guarantee that all of the paragraphs that matched the topic have been found would be to search all of the documents, thereby defeating the purpose of having the links.

Any number of document bases can be quickly constructed with varying sizes and characteristics and used as testbeds for scout experiments. If automatic semantic hypertext construction does become feasible in the future, it would be an interesting experiment to determine if such simulated document bases actually reflect real document bases. As an example of the output of the simulator, excerpts of two documents in a collection of 1000 documents are shown below for a case in which the number of paragraphs per document ranges from 10 to 20, the number of links per paragraph from 4 to 7, the weighting of each link is between 0.4 and 1, and the number of paragraph topics is 50. The boxes indicate particular links that we will look at as examples.

```

<A HREF="DOC0082.html#PAR000" WEIGHT=0.919487 TYPE=16> </A><BR>
<A HREF="DOC0087.html#PAR004" WEIGHT=0.443453 TYPE=10> </A><BR>
<A HREF="DOC0088.html#PAR001" WEIGHT=0.592511 TYPE=10> </A><BR>
<A HREF="DOC0088.html#PAR005" WEIGHT=0.748802 TYPE=24> </A><BR>
<BR>
<A NAME="PAR003"></A>
<P>
Paragraph 3<BR>
28 ← TOPIC
<A HREF="DOC0001.html#PAR005" WEIGHT=0.949880 TYPE=15> </A><BR>
<A HREF="DOC0003.html#PAR004" WEIGHT=0.721383 TYPE=20> </A><BR>
<A HREF="DOC0004.html#PAR008" WEIGHT=0.403744 TYPE=21> </A><BR>
<A HREF="DOC0010.html#PAR003" WEIGHT=0.569521 TYPE=24> </A><BR>
<A HREF="DOC0011.html#PAR004" WEIGHT=0.593057 TYPE=4> </A><BR>
<A HREF="DOC0016.html#PAR002" WEIGHT=0.698320 TYPE=2> </A><BR>
<A HREF="DOC0017.html#PAR000" WEIGHT=0.862954 TYPE=20> </A><BR>
<A HREF="DOC0019.html#PAR001" WEIGHT=0.711326 TYPE=2> </A><BR>

```

Figure 2. Excerpt from document DOC0004.html around paragraph 3.

Figure 2 shows part of paragraph 3 in document 4; note that the topic of the paragraph is 28. The link information is contained within the <A> tags. The first large box highlights a link to paragraph 8, designated PAR008, in the same document and the second indicates a link that points to paragraph 0 in document 17. Figure 3 is another excerpt of DOC0004.html that incorporates paragraph 8, the other end of the first highlighted link. The box in Figure 3 highlights the companion link. Notice that the topics of the paragraphs and the weights and link types are the same, illustrating the symmetry of the links.

```

<A HREF="DOC0087.html#PAR006" WEIGHT=0.521700 TYPE=11> </A><BR>
<A HREF="DOC0094.html#PAR005" WEIGHT=0.631772 TYPE=2> </A><BR>
<A HREF="DOC0095.html#PAR007" WEIGHT=0.916176 TYPE=10> </A><BR>
<A HREF="DOC0099.html#PAR004" WEIGHT=0.617879 TYPE=2> </A><BR>
<BR>
<A NAME="PAR008"></A>
<P>
Paragraph 8<BR>
28
<A HREF="DOC0001.html#PAR005" WEIGHT=0.621065 TYPE=24> </A><BR>
<A HREF="DOC0003.html#PAR004" WEIGHT=0.888180 TYPE=14> </A><BR>
<A HREF="DOC0004.html#PARC03" WEIGHT=0.403744 TYPE=21> </A><BR>
<A HREF="DOC0010.html#PAR003" WEIGHT=0.642424 TYPE=14> </A><BR>
<A HREF="DOC0011.html#PAR004" WEIGHT=0.759506 TYPE=4> </A><BR>
<A HREF="DOC0016.html#PAR002" WEIGHT=0.525494 TYPE=11> </A><BR>
<A HREF="DOC0017.html#PAR000" WEIGHT=0.559202 TYPE=23> </A><BR>
<A HREF="DOC0019.html#PAR001" WEIGHT=0.868547 TYPE=4> </A><BR>
<A HREF="DOC0021.html#PAR001" WEIGHT=0.505518 TYPE=16> </A><BR>

```

Figure 3. An excerpt of DOC0004.html around paragraph 8.

The second large box in Figure 2 indicates a link to a paragraph in a separate document. An excerpt of that document, DOC0017.html, is shown in Figure 4.

```

HTML>
<HEAD>
<TITLE>DOC0017.html</TITLE>
</HEAD>
<BODY>
<BR>
<A NAME="PAR000"></A>
<P>
Paragraph 0<BR>
28
<A HREF="DOC0001.html#PAR005" WEIGHT=0.745084 TYPE=10> </A><BR>
<A HREF="DOC0003.html#PAR004" WEIGHT=0.468975 TYPE=23> </A><BR>
<A HREF="DOC0004.html#PAR003" WEIGHT=0.862954 TYPE=20> </A><BR>
<A HREF="DOC0004.html#PAR008" WEIGHT=0.559202 TYPE=23> </A><BR>
<A HREF="DOC0010.html#PAR003" WEIGHT=0.513752 TYPE=22> </A><BR>
<A HREF="DOC0011.html#PAR004" WEIGHT=0.726203 TYPE=19> </A><BR>
<A HREF="DOC0016.html#PAR002" WEIGHT=0.555541 TYPE=19> </A><BR>
<A HREF="DOC0019.html#PAR001" WEIGHT=0.632944 TYPE=7> </A><BR>
<A HREF="DOC0021.html#PAR001" WEIGHT=0.466975 TYPE=1> </A><BR>
<A HREF="DOC0022.html#PAR001" WEIGHT=0.782399 TYPE=10> </A><BR>
<A HREF="DOC0023.html#PAR004" WEIGHT=0.472779 TYPE=23> </A><BR>

```

Figure 4. Excerpt from DOC0017.html showing return link to DOC0004.html

Again, by comparing the highlighted link in Figure 4 with the second highlighted link in Figure 2, we see that the corresponding paragraph topics and link weights and types are the same.

This tool for generating document bases can be used to generate a wide variety of document base characteristics. The number of documents is selectable. The sizes of the documents vary according to the selected minimum and maximum number of paragraphs per document. If the minimum and maximum numbers are close to one another, the documents are more uniform in size, while values that are far apart lead to a mixture of sizes. The number of topics that are allowed determines the connectedness of the document base. That is, fewer topics available leads to documents that are very similar and thus heavily linked. A higher number of available topics tends to produce documents that are more independent of one another.

CHAPTER 6

SCOUT METHODOLOGY

6.1 Supporting Technology

The parallel execution of the scouts sped up the search and the coordination was achieved by using the knowledge contained in the semantic links and passing messages among the scouts and a central controller over the network that contained the documents. The efficiency demonstrated by the scouts is due not only to parallel execution, but also to data contained in the semantic links. Once some information that satisfies the user's query is found, the scouts simply follow the links to find more.

6.1.1 Meta-Knowledge Representation

In the present study, a group of identical semantic scouts cooperate to solve a particular information retrieval query. To work together, the scouts have a shared understanding of the problem domain. Because the scouts are identical, interoperability issues that arise with agents in differing domains do not arise. The knowledge required for a scout to perform is not complex and the communication among scouts is straightforward. Thus it was sufficient to develop simple protocols consisting of only a few commands that will be presented in the next chapter. Any required message is made up of these commands.

Each scout type has a protocol that is used to decipher the meanings of the messages being shared among the scouts. The messages themselves are carried via the Knowledge Query and Manipulation Language (KQML).[28, 29] The KQML is used by programs to communicate attitudes about information, such as querying, starting, believ-

ing, requiring, achieving, subscribing, and offering. This language is indifferent to the format of the information itself, and KQML expressions will often contain subexpressions in different content languages. A KQML message is called a performative, since the message, by virtue of being sent, is intended to cause some action to be performed. The performatives are grouped by the type of action that is expected. In all, there are about two dozen reserved performative names that fall into seven basic categories. Each expression in KQML contains a single performative and a list of parameters in the form of keywords or value pairs. For example, one KQML performative, "tell", is used simply to pass some information; no reply is expected. Another performative, "ask-one", is used when one agent is requesting information from another and expects a reply. The basic KQML performatives were sufficient for the scout communication in this study.

6.1.2 Communication

The KQML acts over the network that connects the computers being used. In the case of the present study, the network is the Internet and the communication protocol is TCP/IP. A number of methods for establishing communication between processors on different machines are available. For instance, the scouts in early experiments conducted for this study used the Parallel Virtual Machine (PVM), a commonly used library of message-passing functions. Another standard becoming widely used is the Message Passing Interface (MPI), which has less overhead and is faster than PVM, but lacks some of the process control features necessary in an implementation such as the semantic scouts. Lower level methods, such as sockets, are also available.

Because this study is not concerned with low level network protocols, a higher level language—Java—which has built-in capabilities for network communication, was

used. At an even higher level of abstraction, agent frameworks have been built in Java.

One of these agent frameworks, JATLite, was used in this study and is described in more detail later in this chapter.

6.2 Individual Scout Function

The principal function of a scout is to find information relevant to a user query. A scout operates in the milieu of a knowledge base that has been preprocessed, to a degree, in that it contains typed and weighted hypertext links among its constituent parts. Therefore, rather than scan entire documents looking for relevant information, the scouts need only find one segment of one document that matches the query. Further search, if any, consists of following links from that segment.

Although three scout types were developed, Scoutmaster, Broadcaster, and Melee, they all have some characteristics in common. They all begin their search at a document assigned to them by a central controller and scan this document for information that is relevant to the user query. If the information is found the scout will find links to similar information elsewhere in the document base. If the supplied document does not contain the information, the scout will request another from the central controller. Once any scout finds a link, then the search proceeds according to method being used, either Scoutmaster, Broadcaster, or Melee.

6.2.1 Method of Scout Traversal

Each scout traverses to the next location in the document base using the target information in the link data. With the Scoutmaster type, the link data is provided to the scout by the Scoutmaster in response to a request. With the other types, Broadcaster and

Melee, the link information comes from the other scouts, either directly or relayed through the Broadcaster. In the case of the latter two, the scout maintains an updated list of links and selects the first on the list to follow.

When the scout reaches the target of the link, it sends the text of the target back to the central controller for reporting back to the user and scans the text for additional links. With the simulated document based being used in this study, no real text is returned by the scouts, only the fact that it found relevant information as well as its location. The link information at the scout's current location is then extracted and sent to either the Scoutmaster, Broadcaster, or, in the Melee case, all the other scouts.

6.3 Scout Taxonomy

The aim of this study was to ascertain the effect that the characteristics of the scouts have on the effectiveness and efficiency of the information retrieval process. To this end, several types of scouts were developed, each with a distinct set of characteristics. Because the activities of the scouts is simple, consisting principally of following hypertext links, only a small number of scout types is required to achieve a complete picture of scout behavior. The intent was to run the gamut of scout autonomy between scouts that communicated only with a central controller that provided the scouts with direction and scouts that determined their own courses by communicating among themselves. A type that combined the attributes of the two extreme cases was also developed. The following spectrum of scout classes was designed to explore the question of scout autonomy.

6.3.1 Scoutmaster

The most basic type of scout is one that makes simply follows links essentially

unaware of what the other scouts are doing. To follow a link to another document the scout must have permission a central controller, hereafter called the **Scoutmaster**. As the retrieval process is progressing, the Scoutmaster receives from each of the scouts the link that the scout is currently investigating and the additional link information that the scout has found. After processing its current location, the scout asks the Scoutmaster for the next link to traverse. The Scoutmaster examines the link information it has received from all the scouts, forms a response to the scout, and sends it. The messages required for scout cooperation are short and relatively few in number. They consist only of queries from the scout and responses to traverse requests from the Scoutmaster.

6.3.2 Broadcaster

At the next level of complexity, the scout makes the decision about its next move based on information it is carrying rather than permission it gets from a Scoutmaster. The method of communication is that each scout sends link information to the central controller which immediately broadcasts it to all the other scouts. The scouts update their own local knowledge bases to reflect the changes and use the information to decide which links to navigate, with the navigation being performed in the same manner as in the Scoutmaster method. This method is labeled the **Broadcaster** because the same information is broadcast by the central controller to all the scouts. In this method, the message traffic will consist of many more transmissions, because each message sent by a scout is repeated to all the other scouts. One set of messages, though, is omitted; the scouts need no longer send queries to a Scoutmaster.

6.3.3 Melee

In the two previous scenarios, all of the information transmitted to the scouts passes through or originates at a single point. In this last case, called **Melee**, the communication is direct from scout to scout. Functionally, the scenario is the same as that of the **Broadcaster**. The difference is that each scout broadcasts any newly found link information directly to all the other scouts. The trade-off is high traffic volume versus a potential bottleneck at the central controller. Another aspect to be considered is that the scouts may spend an inordinate amount of time sending and receiving messages rather than analyzing documents.

6.4 Scout Implementation

The scouts are implemented within an agent framework that provides many of the required logistical services, such as nameserver and socket communications. The particular scout types are developed in an object-oriented manner, inheriting their capabilities from simpler forms.

6.4.1 Agent Framework

An agent framework provides basic services required by any system of agents that communicate across a network. Because a number of these frameworks had been developed by others, there was no need to duplicate that effort here. A survey was made of candidate agent frameworks for this project. These were SodaBot from MIT [30], Aglet Software Development Kit from IBM [31], and Java Agent Template from Stanford University. [32]

SodaBot is a general-purpose user environment and system for the construction of

software agents.[30] It consists of an agent operating system that provides a framework for building software agents and an agent programming language that allows users to implement a wide range of agent applications. The framework handles the low level operational tasks, such as accessing TCP/IP among distributed agents and providing architecture compatibility. The agent language, SodaBotL, offers high level primitives and control structures designed around human level descriptions of agent activity. A graphical user interface allows editing of agents and monitoring of agent activity. Although intriguing, SodaBot was not suitable for use in this research because the system was designed to provide personal services, e.g., scheduling, to a human user, and the code was not made available outside a small number of preselected testers.

The IBM Aglet Software Development Kit (ASDK) began as the Aglet Workbench in an IBM development center in Japan. The ASDK is an environment for programming mobile Internet agents called aglets. Aglets are Java objects that can move from one host on the Internet to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch itself to a remote host, and resume execution there. Arriving at the remote computer, aglets present credentials and obtain access to local services and data.

Using the Aglet Java class is a convenient way for user-defined agents to inherit default properties and functions of mobile agents. These properties and functions include (1) a globally unique naming scheme for agents, (2) a travel itinerary for specifying complex travel patterns with multiple destinations and automatic failure handling, (3) a white board mechanism allowing multiple agents to collaborate and share information asynchronously, (4) an agent message-passing scheme that supports loosely coupled asynchronous as well as synchronous peer-to-peer communication, (5) a network agent class loader that

allows an agent's Java byte code and state information to travel across the network, and (6) an execution context that provides agents with a uniform environment independent of the actual computer system on which they are executing.

The visual agent manager for ASDK, Tahiti, provides a graphical user interface to monitor and control aglets. Through a drag-and-drop interface, two aglets can be made to communicate with each other, or an agent can be dispatched to a particular site. According to the documentation [31], Tahiti is "more than a system administration tool; it is a desktop tool for agent users in the same way that the Web browser has become the fundamental tool for Web users." The ability to launch aglets via a Web browser is also part of the ASDK.

The original Aglet Workbench was not sufficiently developed for use in this project. The current ASDK, however, is a capable, reasonably mature system for agent development. Because of the agent mobility that would allow searches over sets of documents that were not mounted locally, the synchronous peer-to-peer communication, and the graphical user interface for agent development, ASDK would probably be the system of choice for implementing the scouts if the project were just beginning.

Initial scout implementation began with the Java Agent Template (JAT) from Stanford University.[32] A newer version of JAT, called JATLite [33], quickly replaced JAT and was selected as the framework for this project.

JATLite is a package of programs, written in the Java language, that allows users to quickly create new software agents that communicate robustly over the Internet. JATLite provides a basic infrastructure in which agents register, using a name and password, with an Agent Message Router (AMR) facilitator, connect or disconnect from the

Internet, send and receive messages, transfer files, and invoke other programs or actions on the various computers where they are running. JATLite especially facilitates construction of agents that send and receive messages using the emerging standard communications language, KQML, described previously. The communications are built on open Internet standards: TCP/IP, SMTP, and FTP.

JATLite provides a template for building agents that utilize a common, high-level language and protocol. This template provides the user with numerous predefined Java classes that facilitate agent construction. Furthermore, the classes are provided in layers so that the developer can easily decide what classes are needed for a given system. Each added layer adds more capability but also creates a more restrictive environment. For instance, the KQML layer added on the top of base layer, restricts that the users should use KQML as their language. Furthermore, the user must use communication protocol for connection and disconnection. JATLite supports the full KQML syntax suggested in 1993.[28] However, if that layer is included, parsing and other KQML-specific functions are automatically included. This built-in capability to communicate via KQML was a major influence in the selection of JATLite for this study.

The most unique feature of the JATLite is the agent infrastructure packaged with it. Traditional agent systems use an Agent NameServer (ANS) for making connections between agents. An agent uses an ANS simply to look up the IP address of another agent and then uses that address to make a TCP socket connection directly to that agent for the purpose of exchanging messages. With the JATLite infrastructure, all agents make a single connection to the AMR. The AMR forwards all agent messages by name to the last known IP address. Further, just like an email system, the AMR buffers all messages and saves

them until the receiving agent acknowledges receipt with a delete message to the AMR.

The advantages of this implementation are that an agent does not need to track the (possibly changing) other agents' addresses or worry about temporary message delivery problems, asynchronous communication is possible even when the receiver agent can not receive a message, and an applet agent can communicate with other agents in spite of the applet security features in web browsers.

Another advantage of this Java-based system is that scouts can be written either as standalone applications or as applets. In general, applications are more efficient than applets, so the bulk of elements of the research, including scouts and central controllers were written as standalone applications. The applet versions allow interactive presentation of a scout using an appletviewer or web browser. The user can be a scout and type in messages that are sent via the agent framework and read messages that are sent to it. Applet-based scouts were very useful for debugging during the development of this project. A user can send any legal KQML message to any scout or central controller and evaluate whether the responses are correct. A screen shot of a scout applet in use is shown in Figure 5. The figure shows a scout named ScoutA just after sending a message to the AMR asking for a list of all agents that have registered. A list of messages received by ScoutA appears in the upper right box and selecting a message causes the message content to appear in the lower right box. In this case, the content consists of a list of registered agents, including the JATLite Router, ScoutA, and the ScoutMaster.

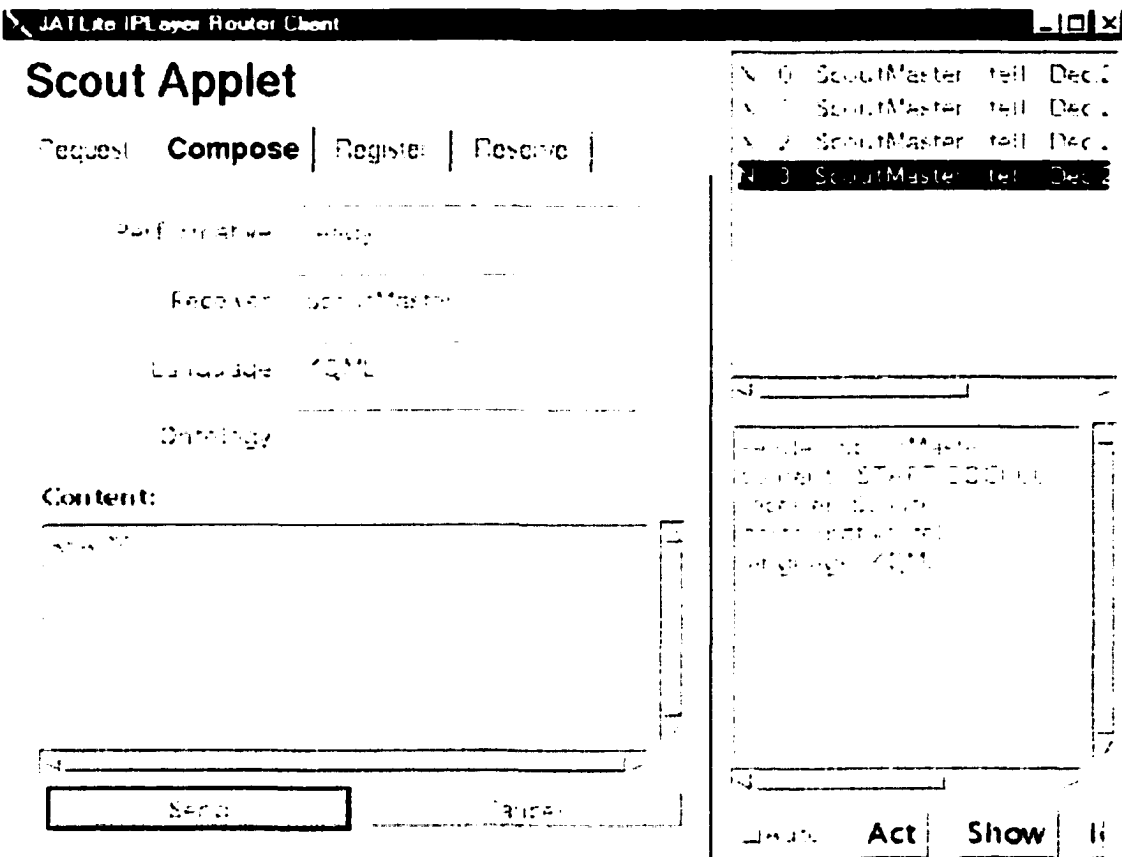


Figure 6. Screenshot of a scout applet conversing with the ScoutMaster

The scout applet allowed the author to be inserted into the search process as a monitor. From the applet, messages were sent to the central controller or scouts. Code was added to the scouts that generated appropriate responses to the messages received from the applet. This ability provided an interactive debugging facility that was invaluable during the development phase of this project.

The system can also be monitored through log files generated by the scouts and central controller. Every time a message is sent or received, this event is written to a log file. In this way, the number of messages sent during the entire process could be counted. Also, significant actions taken by the central controller and scouts were also logged. An

example of a scout log file is shown in Figure 7. The actions of scout00 after the start initiated by the applet ScoutA above are illustrated.

```
scout00-received START DOC0002.html 31 13 0.667
scout00-Could not find topic, Asking SM for another document
scout00-received new document DOC0003.html from ScoutMaster
scout00-Could not find topic, Asking SM for another document
scout00-received new document DOC0004.html from ScoutMaster
scout00-Could not find topic, Asking SM for another document
scout00-received new document DOC0005.html from ScoutMaster
scout00-Found topic 31 in DOC0005.html
scout00-Link, DOC0000.htm#PAR001
scout00-Link, DOC0001.htm#PAR004
scout00-Link, DOC0007.htm#PAR000
scout00-Link, DOC0024.htm#PAR002
scout00-Link, DOC0029.htm#PAR001
scout00-Link, DOC0032.htm#PAR003
scout00-Link, DOC0035.htm#PAR002
scout00-Link, DOC0038.htm#PAR000
scout00-Link, DOC0039.htm#PAR004
scout00-Link, DOC0040.htm#PAR002
scout00-Link, DOC0042.htm#PAR001
scout00-Link, DOC0042.htm#PAR000
scout00-Link, DOC0044.htm#PAR000
scout00-Link, DOC0047.htm#PAR001
scout00-Link, DOC0052.htm#PAR003
scout00-Link, DOC0062.htm#PAR002
scout00-Link, DOC0064.htm#PAR003
scout00-Link, DOC0070.htm#PAR004
scout00-Link, DOC0073.htm#PAR001
```

Figure 7. Example log file generated by a scout.

The first line reports that scout00 received the START command. This scout will begin its search with DOC0002.html, look for topic 31, and follow links of type 13 and weighting not less than 0.667. No paragraph in DOC0002.html had a topic of 31, so the scout asks for another document, the ScoutMaster responds with DOC0003.html, which also does not have a paragraph with topic 31. The process is repeated until DOC0005.html is provided. The scout finds a paragraph with the desired topic and begins to read in the link information. The scout will later follow each of the links from DOC0005.html.

CHAPTER 7

SCOUT CONSTRUCTION

The basic characteristics of the various types of scouts and their associated central controllers have been established. These characteristics have been expanded into specific functions which were implemented in the JATLite framework.

7.1 Scoutmaster

In the Scoutmaster scenario the scouts have no ability follow a link on their own in the navigation of the document base. The controller, or Scoutmaster, receives messages from all the scouts and provides the next move for each one.

7.1.1 Initial operation

When the Scoutmaster starts executing, it first registers with the Agent Message Router (AMR). It then reads a file of initialization data that includes the number of documents being searched and the file names of the documents, the number and names of the scouts as they are registered with the AMR, and the user query. The user query consists of the paragraph topic to search for, the link weighting threshold, and the link type. Once the initialization is complete, the Scoutmaster waits for its first message. The scouts read no initialization data; they get all the operational information they need from the Scoutmaster. When the scouts begin execution, they take no action until they receive a message from the Scoutmaster. The Scoutmaster and scouts will send receive KQML messages whose content will conform to the communication protocol.

7.1.2 Communication Protocol

As in described in the previous chapter, a scout applet can be used to manually send KQML messages. Once the Scoutmaster and scouts are invoked and ready to receive messages this scout applet is used to send one message to the Scoutmaster that starts the search process. It consists of the “ready” performative followed by the command READY with no arguments. See figure 6 of the preceding chapter for a screen shot of the applet being used to start the operation.

Two of the legal messages sent from the Scoutmaster to the scouts use the “tell” performative. The message contents and their syntax are as follows.

```
START did topic type weight
  tells a scout to begin a search
where
  did    - filename, string
  topic  - query text, string
  type   - link type, string
  weight - threshold link strength, float
```

```
STOP
  Sends termination order to scout
```

The only other kind of message sent by the Scoutmaster is a response to a scout’s request for a new document and uses the “reply” performative.

```
NEW_DOC did
  Send new document identifier to scout
where
  did    - filename, string
```

The scout sends three types of messages to the Scoutmaster. The first uses the KQML performative “ask-one.”

REQ_DOC

request a new document or link to traverse from the Scoutmaster

The other two messages use "tell"

LINK_DOC did pid

Notify the Scoutmaster that a link has been found

where

did - document name, string

pid - paragraph identifier, string

FOUND did pid

send document and paragraph of found relevant information

where

did - document name, string

pid - paragraph identifier, string

7.1.3 Search Operations

The search process begins when the Scoutmaster receives the **READY** message from the scout applet whereupon it takes a document name from its list for each scout and sends a **START** to each of the scouts. From this point on the Scoutmaster reacts to messages sent by the scouts.

When a scout receives a **START**, it opens the file corresponding to the document name and parses the file looking for a paragraph with the specified topic. If no paragraph is found, the scout sends a **REQ_DOC** to the Scoutmaster. The Scoutmaster checks to see if any scouts have sent it link information. If it has the link data, it sends the next available link destination to the scout. If not, it sends the name of the next available document on its master list to the scout. In either case, a **NEW_DOC** message is used. If the **NEW_DOC** has only a document identifier, the scout recognizes that it must parse the document and look for a paragraph with the specified topic. If the message also has a paragraph identifier, the scout traverses the link by opening the document and going directly to the para-

graph identified in the message.

When the scout arrives at a paragraph with the correct topic either by parsing or by following a link, it sends a FOUND to the Scoutmaster indicating that the paragraph is indeed one that satisfies the user's query. The scout then examines each of the links contained in the paragraph and compares the weight and type of each link to the values specified in the query. If both parameters match, the link information is sent to the Scoutmaster using LINK_DOC. After all the link data at the current location is sent back to the Scoutmaster, the scout sends a REQ_DOC and the process begins anew.

The Scoutmaster receives the link data from the scouts and adds the link to a list if it is not there already. When the Scoutmaster receives a REQ_DOC from a scout and replies, the index pointer to the list is decremented. When no more links are on the list, the Scoutmaster sends a STOP to any scout that sends a REQ_DOC. When all the scouts have been halted, the Scoutmaster terminates itself and writes a file with the results of the search that includes elapsed time, numbers of messages sent and received, and the locations of the found topics.

7.1.4 JATLite implementation

JATLite is made up of four layers, each of which adds some capability as it is added. A developer of agent software may use whatever layer suits the problem at hand. The lowest layer is called the AbstractLayer and provides abstract classes that are to be implemented in the higher layers. The most basic facilities, such as TCP/IP communications are provided by the next layer, BaseLayer. The KQML layer is the third one and provides facilities, such as message parsing, for handling KQML messages. The Scoutmaster and scouts were implemented in the JATLite Router layer that sits on top of the KQML

layer. This layer inherits all of the functionality of the KQML layer, thus allowing the KQML handling facilities. In addition, the Router layer has a more robust messaging system that can handle situations in which the scouts may be offline for some reason. Also, within the Router layer, scout applets may be executed by web browsers.

The two Java classes developed herein, one each for the Scoutmaster and scout, extend a JATLite provided template called RouterClientAction from the AgentClient package of the Router layer. The Scoutmaster and scout classes used two basic methods inherited from the template for handling messages. The method for sending messages, sendMsg, accepts three parameters, the name of the intended receiver of the message, a KQML performative and the message content. The method then packages the message into legal KQML syntax, and sends the message to the designated receiver.

The other inherited method, called Act, runs continuously and processes the KQML messages. The first part of the Act method, identical between the scout and Scoutmaster classes, receives a message and parses the KQML, picking out the sender of the message, the KQML performative, and the content of the message. Once the message is parsed, the method acts on the content. Additional methods were written by the author to perform the various operational functions described above. The values of the KQML performative and protocol commands determine which of the functional methods are invoked.

7.2 Broadcaster

In the case of the Broadcaster, the link information is again received from each scout, but then it is relayed to all the other scouts after some minor processing. The Broadcaster does not wait for a scout to request new link information.

7.2.1 Initial operation

The initial operation of the Broadcaster is identical to that of the Scoutmaster.

7.2.2 Communication Protocol

Three of the legal messages sent from the Broadcaster to the scouts use the “tell” performative. The message contents and their syntax are as follows.

START did topic type weight
 tells a scout to begin a search
 where
 did - filename, string
 topic - query text, string
 type - link type, string
 weight - threshold link strength, float

NEW_LINK did pid
 Give a scout a link found by another scout
 where
 did - document name, string
 pid - paragraph identifier string

VIS_DOC did
 Broadcaster sends a document name to add to scout’s visited list
 where
 did - filename, string

As in the case of the Scoutmaster, the response to a scout’s request for a new document uses the “reply” performative.

NEW_DOC did
 Send new document identifier to scout
 where
 did - filename, string

The scout sends five types of messages to the Broadcaster. The first uses the KQML performative “ask-one.”

REQ_DOC

request a new document from the Broadcaster

The remaining four messages use "tell."

DOC_VIS did

scout sends a document name that it has visited to be broadcast to other scouts where

did - filename, string

LINK_DOC did pid

Send the Broadcaster a link for broadcasting to other scouts where

did - document name, string

pid - paragraph identifier string

FOUND did pid

send document and paragraph of found relevant information where

did - document name, string

pid - paragraph identifier string

DONE

scout informs Broadcaster that its list of links is empty and it is halting

7.2.3 Search Operations

As in the case of the Scoutmaster, the scout applet is invoked to send the **READY** message to the Broadcaster to begin the search process. Then the Broadcaster takes a document name from its list for each scout and sends a **START** to each of the scouts. From this point on the Broadcaster reacts to messages sent by the scouts.

In a similar manner as with the Scoutmaster, the Broadcaster feeds new documents, using the **NEW_DOC** command, to the scouts until one scout finds a paragraph that matches the specified search topic. In contrast to the Scoutmaster case, once link data becomes available, the scouts never send a **REQ_DOC** to the Broadcaster and the Broad-

caster never again sends a NEW_DOC.

The principal differences between Scoutmaster and Broadcaster operations occur when a scout finds link data. Each scout maintains a list of links that could potentially be traversed. When new links are found, those that meet the query criteria of link weight and type are added to the list if they are not there already. They are also sent to the Broadcaster using the LINK_DOC command. The scout then traverses the next link on its list and reports that it has found information that matches the query.

When the Broadcaster receives the links, it adds them to a list that it maintains after removing duplicates. The links that are added are also sent to all the scouts except the one that provided the link data. Because the scouts maintain their own lists of links to traverse, they need to know which documents have been visited by other scouts. When a scout traverses a link, it notifies the Broadcaster which document it is visiting with the DOC_VIS command. The Broadcaster relays this information to the other scouts with VIS_DOC. The scouts delete the document referred to in the VIS_DOC command from their list of links.

The purpose for having the scout add newly found links to its own list and for having the Broadcaster refrain from sending link information back to the scout who found it is not to reduce the number of messages but to prevent the scouts from developing identical lists of links to traverse. Because of latency in the DOC_VIS/VIS_DOC message combination, the lists would not be updated soon enough and the scouts would likely march in lockstep through the links, with every scout traversing every link.

When a scout depletes its list of links, it sends a message, DONE, to the Broadcaster, writes its performance data to a file, and terminates. After all the scouts have fin-

ished, the Broadcaster follows suit.

7.2.4 JATLite implementation

Structurally, the implementation of the Broadcaster is the same as that of the Scoutmaster. Additional functional methods were added under the Act method to handle the Broadcaster-specific commands in the communication protocol.

7.3 Melee

After initialization, the central controller in this case is out of the loop as far as link information is concerned. The scouts communicate with each other directly.

7.3.1 Initial operation

The initial operation of the Melee is identical to that of the Broadcaster.

7.3.2 Communication Protocol

The communication protocol of the Melee is identical to that of the Broadcaster with the exception that the scouts send the `LINK_DOC` and `DOC_VIS` commands to the other scouts instead of the central controller.

7.3.3 Search Operations

There is one difference between the Broadcaster and Melee search operations. When a scout finds link data that matches the query criteria, it checks the link against its link list. If the link is not a duplicate, it adds it to its own list and sends the link data to the other scouts. Again, to help prevent list duplication, the scout adds new link information to the opposite end of the list from which it takes its new link to traverse.

7.3.4 JATLite implementation

The only changes made in the Melee implementation from that of the Broadcaster are that the functional method that handles link information receipt and resend in the Broadcaster was removed and the method that sends link data from the scout was modified to send the data to the other scouts.

CHAPTER 8
TESTS AND RESULTS

8.1 Setup of document bases

To determine the effect of document base characteristics on scout performance, two document bases with different characteristics were established for the scouts to explore. Using the program described in Chapter 5, one set was generated having a large number of small documents, and the other has a smaller number of larger documents. The minimum and maximum allowable number of paragraphs per document controlled the sizes of the documents. The number of possible topics for the paragraphs controlled the link density. For both sets of documents, the number of link types was set to 25. Both document bases are fully connected. The values for all of the variables, as well as the calculated values of total numbers of paragraphs and links are shown in the following table.

Table 2: Characteristics of test document bases

| Case Name | # of Docs | Min # of Pars/Doc | Max # of Pars/Doc | Topics | Total # of Pars | Total # of Links | Links/Par |
|-----------|-----------|-------------------|-------------------|--------|-----------------|------------------|-----------|
| BaseA | 3000 | 2 | 5 | 75 | 9,007 | 1,080,278 | 119.9 |
| BaseB | 1500 | 6 | 10 | 50 | 11,240 | 2,526,066 | 224.7 |

Some preliminary tests indicated that the scouts would perform very quickly, so large numbers of documents and links were generated. It may seem unrealistic to expect that real documents will contain hundreds of links in every paragraph, but the purpose of

the linking is to facilitate information retrieval, not casual browsing. The documents may also be useful for browsing if alternate ways of presenting the links were developed. The investigation of more compact link representation schemes was not part of the current study, but is discussed later in Chapter 9.

8.2 Scout Setup

The experiments were performed on a set of networked UNIX workstations. All of the workstations shared a common file system using NFS. The code for the scouts was always compiled on the same machine and the directory where the compiled Java classes were stored was available to each machine via NFS. Each computer also had access to the directories where the sets of documents were kept.

A number of computer types were available on the network and experiments were set up to determine the effect of machine type on the scout performance. Each type of machine was able to execute each of the single scout program compilations, thus validating the compile once, run anywhere capabilities of Java, at least for this application.

Scouts of each type, Scoutmaster, Broadcaster, and Melee, were used to find the paragraphs that matched a specified topic by following links that had weightings that were greater than a specified value. The effect of using link type was also investigated. For each scout type and for each of the document bases, various numbers of scouts were employed.

To begin each experiment, the central controller and its associated scouts were started and registered with the JATLite router using unique names. A scout applet as described in Chapter 6 was brought up and a start message was typed in and sent to the central controller that subsequently sent start messages to the scouts. During execution,

the central controller and scouts wrote entries to log files corresponding to significant actions, such as messages received or sent, paragraph topics found, etc. The data reported by the scouts is tabulated below.

8.3 Scoutmaster performance results

The initial experiments were performed using a single scout so that its performance could be verified before adding the complexity of additional scouts.

8.3.1 Effect of computer type

The first experiment was to compare the performance of the Scoutmaster on several machine types. The results of this experiment were then used to decide which computers to use in subsequent experiments. The workstations to be compared included SGI Octane with two R10000 processors, SGI Indigo with one R10000, SGI Indigo with one R4400, Sun Sparc 10, Sun Ultra with two processors, and an Intel 400MHz Pentium III with the Linux operating system. The SGI and Sun machines used the manufacturers' versions of Unix. The following table shows the results of operating a single scout looking for paragraphs with topic 14, following links with weight greater than or equal to 0.6 and of any type. The numbers of messages sent and received are for the Scoutmaster only, the numbers would be reversed for the single scout.

Table 3: ScoutMaster, Effect of machine type, BaseA, single scout, w=0.6, topic=14

| Machine Type | Elapsed Time, min:sec | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | Pars Found |
|--------------|-----------------------|-------------------|--------------------|--------------------|------------|
| SGI Octane | 8:16 | 496 | 8765 | 129 | 119 |
| SGI R10K | 9:53 | 593 | 8765 | 129 | 119 |
| SGI R4400 | 12:00 | 720 | 8765 | 129 | 119 |
| Sun Sparc | 13:12 | 792 | 8765 | 129 | 119 |
| SunUltra | 9:32 | 572 | 8765 | 129 | 119 |
| Linux PC | 8:28 | 468 | 8765 | 129 | 119 |

The spread in the elapsed times for the various machines is modest compared to the differences in the rated processor speed. The slowest computer took about 60% more time to search the document base, but the processors used varied in speed by a factor of ten. This small variance in elapsed time is an indicator that the most of the scout activity involves the sending of messages across the network, and not the crunching of data. Note that the Linux PC outperformed all but one of the other machines, at about 5% of the cost.

The scout sends the Scoutmaster all of the link data that it finds so the number of messages received by the Scoutmaster is large. Conversely, the Scoutmaster only sends the scout the link that the scout should follow next. Thus the scout receives a message for each document that it visits. In Table 3, the number of documents visited exceeds the number of paragraphs with topic 14 by ten, reflecting the fact that the Scoutmaster had to give the scout ten documents before the first desired paragraph was located. From then on, only documents pointed to by links were followed. Thus, the scout visited only 129 documents,

not all 3000 that were in the set of documents. All of the paragraphs with the specified topic in all of the documents were found giving high recall. Only 10 documents that did not have the specified topic were found, giving high precision.

The scout located all 119 paragraphs with topic 14 that were in the document base even though it was only following links with a weighting of 0.6 or greater. For the 119 documents that had links to follow, over 8000 messages were sent to the Scoutmaster, for about 75 links transmitted for every document. From Table 2 above, the average document had about 120 links per paragraph, an indication that links were being discarded. Although the scout discarded weak links, it eventually located every desired paragraph indirectly through different sets of links. Since the link strengths were determined randomly and were uniformly distributed, the sheer number of links and full connectedness of the documents insured that for this case every paragraph was eventually reachable.

From this point on, all of the experiments were performed using the two processor SGI Octane workstations. Not only were the Octanes the most numerous, but having two processors made them less susceptible to variations in performance since all of the workstations were used on a day-to-day basis by a single individual. In addition, the experiments were performed at night and on weekends to minimize the influence of extraneous factors. From time to time, earlier tests were repeated later to spot check the consistency of the results. In all cases, the performance of the original and repeated tests were within 2% of each other.

8.3.2 Effect of topic selection

The next set of experiments tested the scouts' ability to find different topics. Since each topic is represented in the same manner, differing only in value, the scout operation

was very similar to the previous tests. Six topic values out of a total of 72 were selected at random. The number of paragraphs represented by one topic varied about the average of 120 (from Table 2) and in each case all of the paragraphs in the document base were found. The results are shown in Table 4.

Table 4: ScoutMaster, Effect of topic selection, BaseA, single scout, w=0.6

| Topic | Elapsed Time, min:sec | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | Pars Found |
|-------|-----------------------|-------------------|--------------------|--------------------|------------|
| 14 | 8:16 | 496 | 8765 | 129 | 119 |
| 33 | 8:40 | 520 | 7597 | 131 | 111 |
| 40 | 9:30 | 570 | 10775 | 139 | 132 |
| 56 | 8:09 | 489 | 7898 | 131 | 113 |
| 61 | 8:43 | 523 | 10339 | 138 | 129 |
| 72 | 9:27 | 567 | 7724 | 151 | 112 |

As before, the difference between the number of documents visited, as represented by the sent messages, and the number of paragraphs found, is the number of documents that had to be searched to obtain a starting location. The elapsed time for scouts to find all of the paragraphs does not follow from the amount of message traffic. For example, while the search for topic 40 gave the largest elapsed time and the largest number of messages, the time to find topic 72 took almost as long but generated the second fewest number of messages. Looking closer, we see that, in the case of topic 72, the scout had to parse 39 documents (151-112) to find the first paragraph. Parsing a document is a relatively time-consuming process and contributed the large value for elapsed time for topic 72.

8.3.3 Effect of link weighting

As stated above, even though many links are discarded because their weights are below the specified threshold, all of the paragraphs with the desired topic were eventually found through indirect paths. A set of tests were set up to determine at which value of link weight paragraphs with the specified topic are not found. The results from those tests are shown in Table 5.

Table 5: ScoutMaster, Effect of weighting, single scout, BaseA, topic=14

| Link Weight | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Pars Found | # of Docs Visited |
|-------------|--------------|-------------------|--------------------|--------------------|-----------------|-------------------|
| 0.6 | 8:16 | 496 | 8765 | 129 | 119 | 128 |
| 0.9 | 7:53 | 473 | 1736 | 129 | 119 | 128 |
| 0.95 | 7:50 | 470 | 994 | 129 | 119 | 128 |
| 0.975 | 7:23 | 443 | 618 | 122 | 112 | 121 |
| 0.99 | 4:15 | 255 | 273 | 73 | 63 | 72 |
| 1.0 | 2:55:56 | 10556 | 3121 | 3001 | 119 | 3000 |

Using a link weight of 0.9 all of the paragraphs with topic 14 are again found, but because many more links are discarded, the scout sends far fewer messages to the Scoutmaster and the elapsed time drops slightly. With a weight threshold of 0.95, this result is repeated. It is not until the cutoff link weight has been increased to 0.975 that some of the paragraphs with the specified topic are not found. And with 0.99, the number found is about half that of the original case. It is anticipated that, in a real semantically-linked document base where the link weights are not random but based on semantic factors, the culling effect of increasing the weight threshold will be more pronounced.

In the final case in this set, where the link threshold is set to 1.0, no links will be followed. This case is exactly equivalent to sending a scout to parse all of the documents looking for the specified topic. No *a priori* information at all is used in the information retrieval process. The time required for the parsing is far greater than any case that uses the link data.

8.3.4 Effect of numbers of scouts

A key element of this research is to investigate the ability of a number of scouts to cooperate in a search, thereby improving efficiency. The next set of experiments show the results of that cooperation. The resulting times and numbers of messages are shown in Table 6. The numbers of messages sent and received are for the Scoutmaster only and the number of documents visited is the total for all the scouts.

Table 6: Scoutmaster, Effect of number of scouts, BaseA, topic=14, weight=0.6

| # of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|-------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 8:16 | 496 | 8765 | 129 | 128 | 1.00 |
| 5 | 1:44 | 104 | 8769 | 137 | 132 | 4.77 |
| 10 | 1:03 | 63 | 8774 | 147 | 137 | 7.87 |
| 15 | 0:57 | 57 | 8774 | 152 | 137 | 8.70 |
| 20 | 0:46 | 46 | 8786 | 169 | 149 | 10.78 |
| 25 | 0:54 | 54 | 8795 | 158 | 183 | 9.19 |

The results demonstrate that increasing the number of scouts reduces the time required to find the desired information, that is, the search is more efficient. The minimum time to perform the search, 46 seconds, occurs when the number of scouts is 20. The num-

ber of documents visited increases by only 21 out of the total of 3000 from the single scout case. Thus, the efficiency of the search is improved substantially with the use of cooperating scouts, and the high recall and high precision are maintained.

The speedup is essentially linear up to 5 scouts but is less than linear for greater numbers. For more than 10 scouts the speedup improvement tails off even more. For more than 20 scouts the speedup actually decreases. The result is shown graphically in Figure 8.

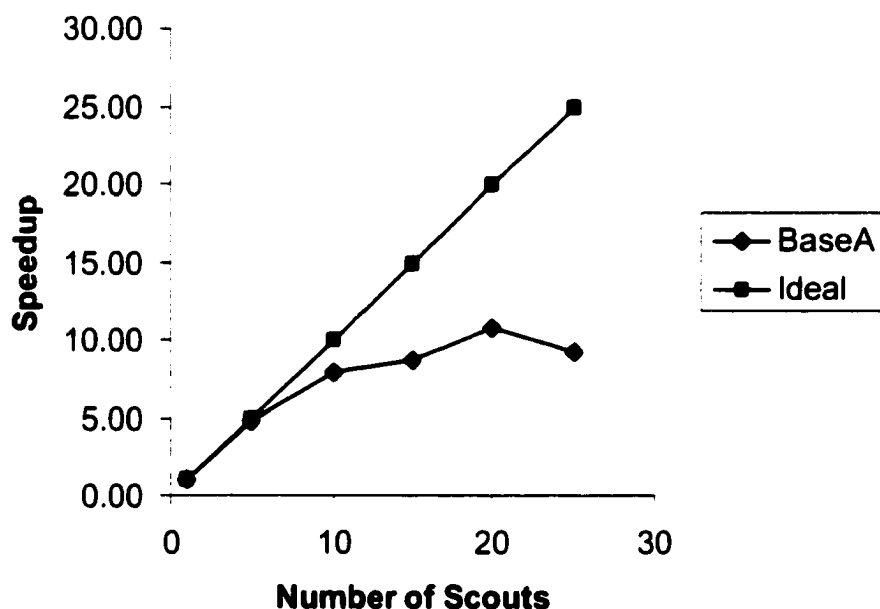


Figure 8. Scoutmaster, Effect of number of scouts on efficiency, BaseA, topic=14, w=0.6

One possible contributor to the less than linear improvement at higher numbers of scouts is that the elapsed time becomes small enough such that no further reduction can occur. Although each scout must find a starting point and thus more documents must be

parsed, very little message traffic occurs during the parsing phase of the process and, on average, each scout will take about the same time to find its starting location, so the document parsing does not contribute to the sublinear improvement. After the presentation of the results of more experiments, the question of sublinear speedup will be revisited later in this chapter.

The number of messages in table 6 are for the Scoutmaster only. Most of the messages are sent from the scouts to the Scoutmaster and the number of messages received by the Scoutmaster is essentially constant across the range of scout number. The messages sent by the Scoutmaster increases slightly with the number of scouts, principally because each scout is sent some new documents to parse to find the starting point. The distribution of the messages among the scouts was not uniform. For example, the number of messages sent by each scout in the case where there were 10 scouts is shown in Figure 9.

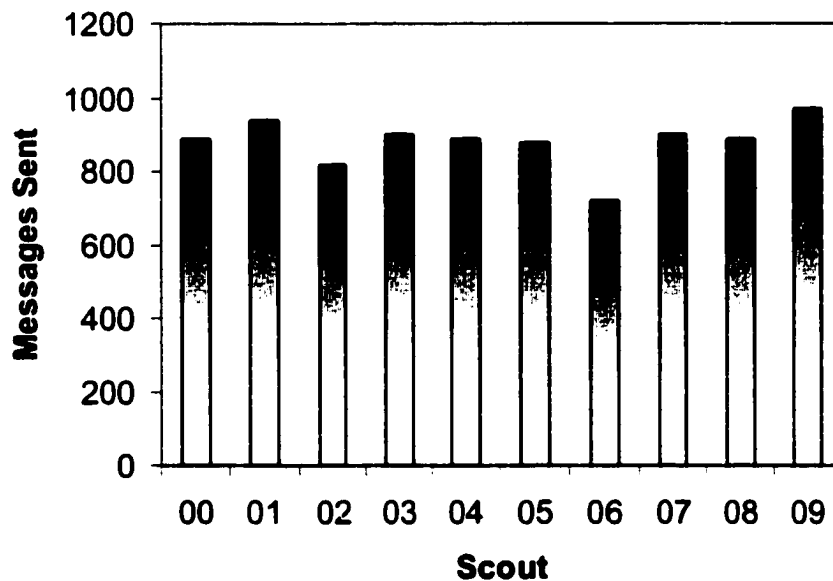


Figure 9. Scoutmaster, Messages sent by each scout, number of scouts=10, Base A, topic 14, weight=0.6.

The highest number of messages sent by any scout is 35% higher than the lowest number sent by any scout. This variation was typical of all of the Scoutmaster experiments, including the ones to follow. The variation was expected because the number of links per paragraph in the documents was random. The number of documents actually visited, however, was divided equally among the scouts, varying by no more than two in any Scoutmaster experiment.

8.3.5 Effect of Document Base

The next experiments investigated the performance of the scouts on the document set BaseB with fewer but larger documents. Because the number of links in BaseB is substantially larger than in BaseA, the number of messages sent from the scouts to the Scoutmaster increased accordingly. As a result, the message buffer in the JATLite router filled to the maximum and the router shut down and the entire process terminated. To alleviate the number of messages being sent, the experiments were run again using a link weight threshold of 0.95. As we saw before in Table 5, this level of link weighting results in fewer messages being sent but all paragraphs with the specified topic are found. The results are in Table 7.

This time the improvement in speedup for larger numbers of scouts is much greater. The speedup increase is linear up to about 10 scouts and falls off only slightly after that. When there are 25 scouts the speedup is 80% of the ideal linear value.

Table 7: Scoutmaster, Effect of number of scouts, BaseB, topic = 14, weighting = 0.95

| # of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|-------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 14:43 | 883 | 2692 | 216 | 215 | 1.00 |
| 5 | 2:46 | 166 | 2695 | 223 | 218 | 5.32 |
| 10 | 1:21 | 81 | 2675 | 225 | 216 | 10.91 |
| 15 | 0:58 | 58 | 2700 | 238 | 223 | 15.22 |
| 20 | 0:46 | 46 | 2705 | 248 | 228 | 19.20 |
| 25 | 0:39 | 39 | 2711 | 259 | 234 | 22.64 |

Because of the different weighting thresholds used we cannot directly compare the BaseA and Base B results. For a more consistent comparison between the results using the two document bases, the BaseA cases were run again using the same value, 0.95, for link weighting. The following table contains the results.

Table 8: Scoutmaster, Effect of number of scouts, BaseA, topic = 14, weighting = 0.95

| # of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|-------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 7:50 | 470 | 994 | 129 | 119 | 1.00 |
| 5 | 1:33 | 93 | 998 | 137 | 132 | 5.05 |
| 10 | 0:49 | 49 | 1004 | 148 | 138 | 9.59 |
| 15 | 0:35 | 35 | 1006 | 155 | 140 | 13.43 |
| 20 | 0:32 | 32 | 1032 | 186 | 166 | 14.69 |
| 25 | 0:28 | 28 | 1034 | 196 | 171 | 16.79 |

The improvement in speedup is better than the case when weighting was 0.6,

though the improvement is not as dramatic as the BaseB result. By the time 20 scouts are used the speedup is only 61% of the linear ideal.

Before tackling the problem of the dropoff in performance at larger numbers of scouts, let us look at one more case that will be useful in the comparisons of search efficiency. As noted previously, for a the case where a scout does not use the link information but instead parses the documents in the search for the specified topic, the elapsed time required is very large compared to that of a scout using the links. The next experiment investigated the use of multiple scouts doing the parsing and the results are presented in Table 9. For this experiment, no link data is being used, thus the scout type, Scoutmaster, Broadcaster, or Melee, is not a factor.

Table 9: Document parsing, no links, BaseA, topic =14

| No. of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|---------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 2:55:56 | 10556 | 3121 | 3001 | 3000 | 1.0 |
| 5 | 33:51 | 2031 | 3121 | 3005 | 3000 | 5.20 |
| 10 | 16:52 | 1012 | 3121 | 3010 | 3000 | 10.43 |
| 15 | 11:15 | 675 | 3121 | 3015 | 3000 | 15.64 |
| 20 | 8:36 | 476 | 3121 | 3020 | 3000 | 22.18 |
| 25 | 6:44 | 404 | 3121 | 3025 | 3000 | 26.12 |

The elapsed times for these cases are far larger than comparable runs that use links, but the improvement in speedup for larger numbers of scouts is greater, even slightly superlinear. The scouts do not send large batches of link data to the Scoutmaster; there is only one pair of messages exchanged for each document visited. Many messages are sent but over a larger span of time. Also, because all 3000 documents were visited, tis case

exhibits high recall, but very low precision.

We can now look at all of the multiple scout results together in Figure 10. Except for the first set of results using document base A, all of the cases show substantial improvement in speedup as the number of scouts is increased to 25. All of the Scoutmaster cases start to show some degradation from the ideal when the number of scouts exceeds 10 with the original BaseA case actually showing a decrease in speedup beyond 20 scouts.

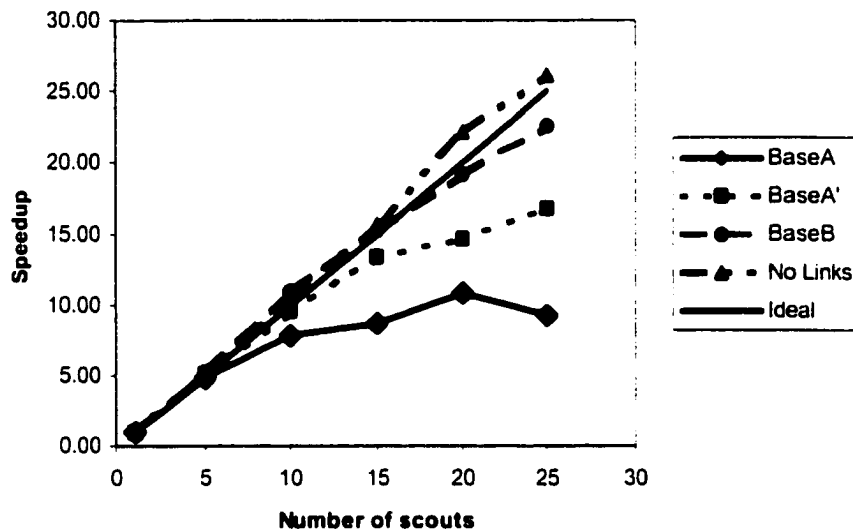


Figure 10. Scoutmaster, Effect of number of scouts on speedup, all cases.

8.3.6 Degradation of speedup improvement

The results from all of the multiple scout cases indicate that for instances where large numbers of messages are sent in a short time the speedup improvement experienced falls short of linear. For each of these cases the message rate was calculated by dividing

the total number of messages sent by the elapsed time. The result is illustrated in Figure 11 below.

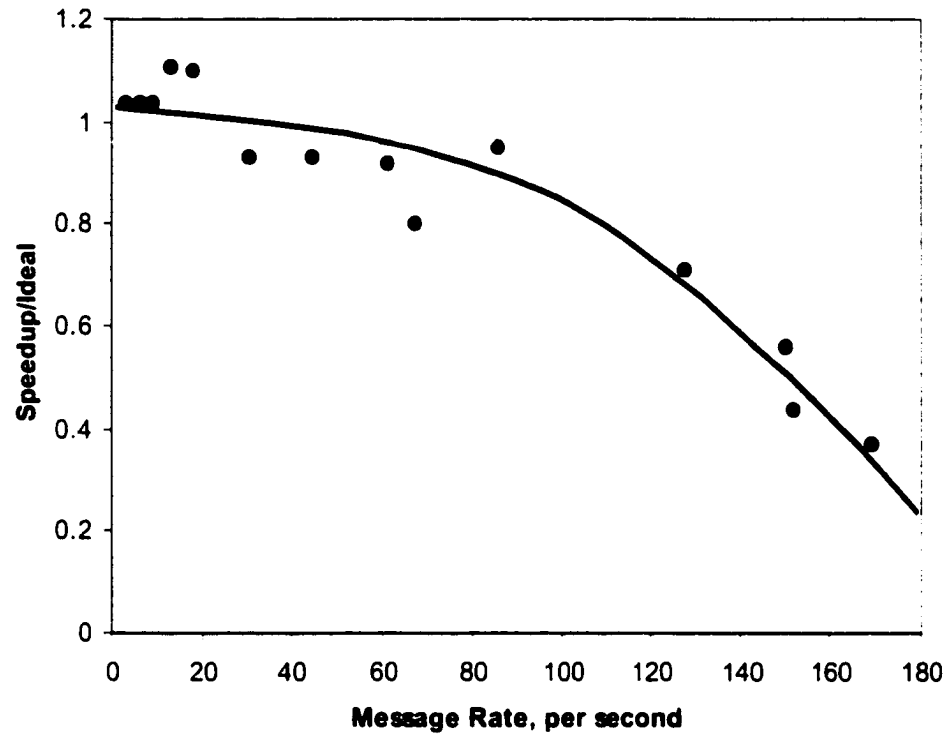


Figure 11. Scoutmaster, Degradation of speedup improvement with message rate, all cases.

There is a sharp drop in speedup improvement once the message rate goes above about 85 messages per second. However, the message rate only goes higher than 85 when the elapsed time is small, say less than a minute. For such low values of elapsed time, one should not expect to reap much benefit from adding more scouts.

8.4 Broadcaster performance results

Each of the Broadcaster type scouts maintains a list of links to follow. The list is

updated by links that the scout finds itself and links that other scouts find as relayed by the Broadcaster. Links are removed from the list when the scout accesses the link and when other scouts access links and relay that information through the Broadcaster. In contrast to the Scoutmaster type, where the scouts must ask for the next link to traverse and then wait for the response, the Broadcaster scouts simply go to the next link on their respective lists. The messages they receive are not in response to a query, but arrive asynchronously as updates to the lists they maintain. The Broadcaster scouts do not have to wait for anything.

The object in this section is to compare the performance of the Broadcaster scouts with those of the Scoutmaster type in terms of elapsed time, speedup improvements with numbers of scouts, and message traffic.

8.4.1 Effect of numbers of scouts

The results from operating numbers of scouts on BaseA are presented in Table 10. The numbers of messages sent and received are for the Broadcaster only, but this is meaningful because all messages go through the Broadcaster. In the case of a single scout the Broadcaster receives link information from the scout but has no other scouts to relay the information to. Therefore, the number of messages sent by the Broadcaster is small. Because the scout does not know that it is the only one, it still sends the link information, hence the large number of messages received by the Broadcaster in the single scout case.

Table 10: Broadcaster, Effect of number of scouts, BaseA, topic = 14, weight = 0.6

| # of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|-------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 1:16 | 76 | 4003 | 238 | 131 | 1.00 |
| 2 | 0:55 | 55 | 720 | 367 | 132 | 1.38 |
| 5 | 0:42 | 42 | 1761 | 871 | 138 | 1.81 |
| 7 | 0:30 | 30 | 2479 | 1235 | 142 | 2.53 |
| 10 | 0:29 | 29 | 3405 | 1698 | 148 | 2.62 |

The most obvious difference with the earlier results is that the elapsed time for low numbers of the Broadcaster scouts is only a small fraction of that for the Scoutmaster type. The times are so low that increasing the number of scouts has a much smaller effect. Another effect of the small elapsed time is that the message rate becomes high, about 158 messages per second through the Broadcaster when the number of scouts is 10. As a result, the JATLite router failed when the number of scouts exceeded 10. Also, as discussed with the Scoutmaster in section 8.3.6 above, the high message rate is responsible for the small increase in speedup as the number of scouts is increased.

8.4.2 Analysis of message traffic, BaseA

As opposed to the Scoutmaster, the number of messages passed during a Broadcaster search varies significantly with the number of scouts. This result is illustrated in Figure 12 below where the sum of the messages sent by the Broadcaster and by the scouts is plotted versus number of scouts.

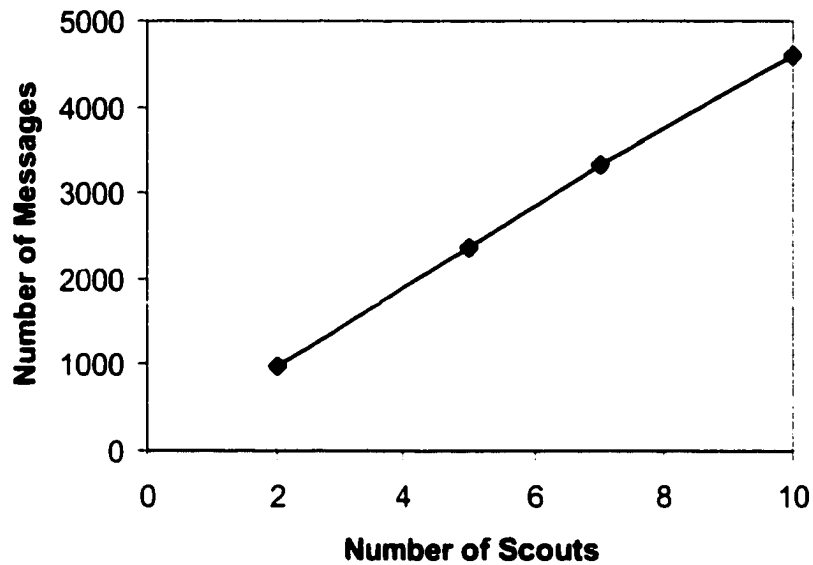


Figure 12. Broadcaster, Effect of number of scouts on message traffic, BaseA, topic=14, weight=0.6.

The number of messages increases linearly with the number of scouts. When compared to the Scoutmaster results in Table 6, the total number of messages for the Broadcaster is substantially less. The Broadcaster is slightly more efficient than the Scoutmaster in that it uses slightly less time and fewer messages. The number of documents visited is about the same.

In the case of the Scoutmaster type, the messages sent from scouts dominated the message traffic while messages sent by the Scoutmaster were relatively few. With the Broadcaster type, the number of messages going from the Broadcaster to the scouts is large. It is useful then to look at the numbers of messages going to and from each scout. These values are shown in Table 11.

The table is divided into sections, one for each value of scout number. The heading

for each section contains the sums of the numbers of messages in the columns below and a message rate calculated by dividing the total messages sent by the elapsed time from Table 10. The new documents column shows how many documents had to be parsed in the beginning of the process.

The Broadcaster always receives about twice as many messages as it sends because most of the messages it receives consist of link data that is passed on only if it is not duplicated. Since only unvisited links are transmitted, the number of messages received by each scout is approximately the same as the number of documents that contain paragraphs with the desired topic.

Table 11: Broadcaster messages, BaseA, topic = 14, weight = 0.6

| Scout | Messages Received | Messages Sent | New Documents |
|-----------------|-------------------|---------------|----------------|
| 1 scout | 4015 | 4014 | 52.81 msgs/sec |
| Broadcaster | 4003 | 12 | 12 |
| scout00 | 12 | 4002 | 12 |
| 2 scouts | 967 | 1096 | 19.93 msgs/sec |
| Broadcaster | 720 | 367 | 13 |
| scout00 | 125 | 365 | 7 |
| scout01 | 122 | 364 | 6 |
| 5 scouts | 2363 | 2632 | 62.67 msgs/sec |
| Broadcaster | 1761 | 871 | 19 |
| scout00 | 119 | 346 | 4 |
| scout01 | 119 | 346 | 4 |
| scout02 | 122 | 356 | 4 |
| scout03 | 120 | 357 | 3 |
| scout04 | 122 | 356 | 4 |

Table 11: Broadcaster messages, BaseA, topic = 14, weight = 0.6

| Scout | Messages Received | Messages Sent | New Documents |
|------------------|-------------------|---------------|----------------|
| 7 scouts | 3322 | 3713 | 123.8 msgs/sec |
| Broadcaster | 2479 | 1235 | 23 |
| scout00 | 120 | 352 | 3 |
| scout01 | 120 | 352 | 3 |
| scout02 | 121 | 353 | 4 |
| scout03 | 121 | 353 | 4 |
| scout04 | 121 | 353 | 4 |
| scout05 | 120 | 352 | 3 |
| scout06 | 120 | 363 | 2 |
| 10 scouts | 4594 | 5102 | 175.9 msgs/sec |
| Broadcaster | 3405 | 1698 | 29 |
| scout00 | 119 | 340 | 3 |
| scout01 | 118 | 344 | 2 |
| scout02 | 119 | 340 | 3 |
| scout03 | 119 | 340 | 3 |
| scout04 | 119 | 340 | 3 |
| scout05 | 119 | 340 | 3 |
| scout06 | 119 | 340 | 3 |
| scout07 | 119 | 340 | 3 |
| scout08 | 119 | 340 | 3 |
| scout09 | 119 | 340 | 3 |

The total number of the messages received displayed at the top of each section is always less than the number of messages sent. When the search halts after all of the links have been visited, some messages that were sent are still in the JATLite router queue at

termination and thus are never passed on to the intended receivers. The difference increases with the number of scouts. The variation in numbers of messages among the scouts is small, giving an indication of the robustness of the implementation.

8.4.3 Effect of Document Base

The results for BaseB, as shown in Table 12, are qualitatively the same as for BaseA. This time the elapsed times are greater than for BaseA, but they are much smaller than their Scoutmaster counterparts. As in the Scoutmaster cases, the time to search BaseB is about twice as long as for BaseA. The number of messages is about 50% greater than for the BaseA tests. The improvement in speedup efficiency is poor compared to the BaseA case. The best time to complete the search is two minutes, far larger than was achieved in any of the other Broadcaster or Scoutmaster tests.

Table 12: Broadcaster, Effect of number of scouts, BaseB, topic = 14, weight = 0.6

| # of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|-------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 2:54 | 174 | 13351 | 406 | 218 | 1.00 |
| 2 | 2:31 | 151 | 1272 | 629 | 218 | 1.15 |
| 5 | 2:04 | 124 | 3131 | 1557 | 234 | 1.40 |
| 7 | 2:00 | 120 | 4543 | 2263 | 240 | 1.45 |

The distribution of the message traffic among the scouts is very similar to that for BaseA, the only difference being that the numbers of messages were larger. The number of messages increased linearly with the number of scouts.

8.5 Melee performance results

The operation of the Melee scouts is the same as the Broadcaster scouts except that messages containing link information are transmitted to other scouts directly rather than relayed by the Broadcaster. When there is a single scout, the Broadcaster and Melee results are identical.

8.5.1 Effect of numbers of scouts

The results for the Melee searching BaseA are shown in Table 13. To be consistent, the numbers of messages sent and received are for the central controller, although because the Melee controller is not involved in the actual search, the message traffic is very low. Comparing these results to those of the Broadcaster in Table 10, the Melee is more efficient, i. e., uses less time, when the number of scouts is five or less. The time is greater when the number of scouts is 7 or more.

Table 13: Melee, Effect of number of scouts, BaseA, topic = 14, weight = 0.6

| # of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|-------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 1:16 | 76 | 4003 | 12 | 131 | 1.00 |
| 2 | 0:39 | 39 | 120 | 12 | 230 | 1.95 |
| 5 | 0:35 | 35 | 121 | 19 | 237 | 2.17 |
| 7 | 0:40 | 40 | 121 | 23 | 238 | 1.90 |
| 10 | 0:64 | 64 | 122 | 29 | 244 | 1.19 |

8.5.2 Analysis of message traffic, BaseA

Because almost all of the message traffic is among the scouts, we will take a closer look at the message distribution. These results are shown in Table 14. Several striking

results are immediately apparent. First, numbers of messages sent is much larger than for the Broadcaster case.

Table 14: Melee, Message activity, BaseA, topic = 14, weight = 0.6

| Scout | Messages Received | Messages Sent | New Documents |
|-----------------|-------------------|---------------|----------------|
| 1 scout | 4138 | 4244 | 55.84 msgs/sec |
| Melee | 4003 | 238 | 12 |
| scout00 | 135 | 4006 | 12 |
| 2 scouts | 364 | 741 | 19.00 msgs/sec |
| Melee | 120 | 12 | 12 |
| scout00 | 122 | 366 | 6 |
| scout01 | 122 | 363 | 6 |
| 5 scouts | 717 | 5314 | 151.8 msgs/sec |
| Melee | 121 | 19 | 19 |
| scout00 | 119 | 1078 | 3 |
| scout01 | 119 | 1053 | 4 |
| scout02 | 119 | 1053 | 4 |
| scout03 | 120 | 1058 | 4 |
| scout04 | 119 | 1053 | 4 |
| 7 scouts | 953 | 10852 | 271.3 msgs/sec |
| Melee | 121 | 23 | 23 |
| scout00 | 119 | 1547 | 3 |
| scout01 | 119 | 1547 | 3 |
| scout02 | 118 | 1547 | 2 |
| scout03 | 119 | 1547 | 3 |
| scout04 | 119 | 1547 | 3 |
| scout05 | 119 | 1547 | 3 |
| scout06 | 119 | 1547 | 3 |

Table 14: Melee, Message activity, BaseA, topic = 14, weight = 0.6

| Scout | Messages Received | Messages Sent | New Documents |
|------------------|-------------------|---------------|----------------|
| 10 scouts | 1192 | 20585 | 321.6 msgs/sec |
| Melee | 122 | 29 | 29 |
| scout00 | 119 | 2285 | 3 |
| scout01 | 119 | 2285 | 3 |
| scout02 | 119 | 2285 | 3 |
| scout03 | 119 | 2285 | 3 |
| scout04 | 119 | 2285 | 3 |
| scout05 | 119 | 2285 | 3 |
| scout06 | 119 | 2285 | 3 |
| scout07 | 119 | 2285 | 3 |
| scout08 | 118 | 2276 | 2 |
| scout09 | 119 | 2285 | 3 |

When there are 10 scouts, the number of messages sent is about 7 times the number for the corresponding Broadcaster test. This large increase in message traffic is the cause of the degradation in speedup after 5 scouts. As shown in Figure 13, the increase is superlinear with increasing number of scouts.

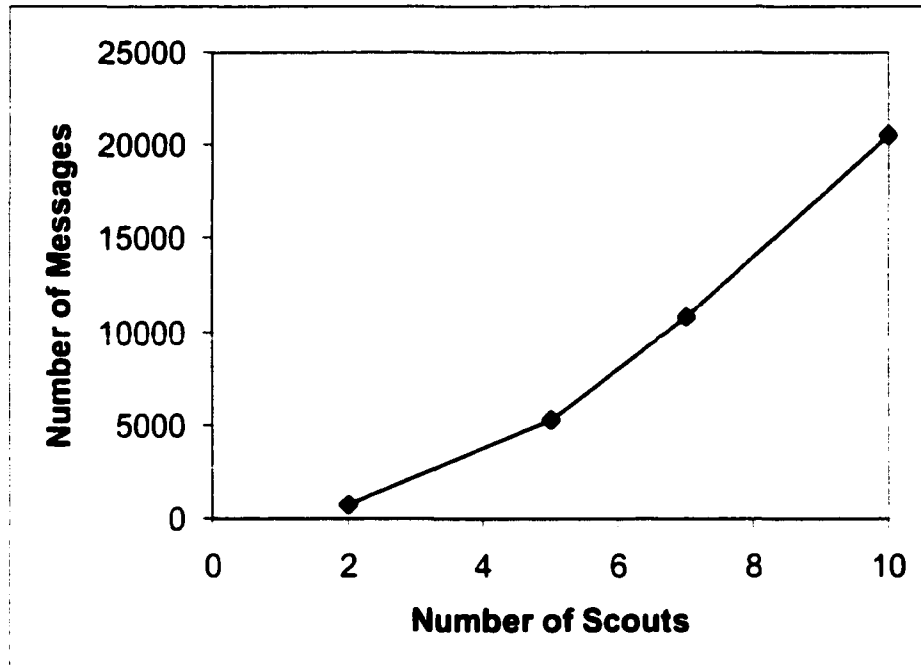


Figure 13. Melee, Effect of number of scouts on message traffic, BaseA, topic=14, weight=0.6.

A second result from Table 14 is the large disparity in the number of messages sent and the number received. The elapsed times are so short and the message rate is so high, that the search is over long before all the messages can be delivered. Also, there is no filter at the central controller as there was with the Broadcaster. The third result is the remarkable uniformity of the numbers of messages sent by the scouts. The largest variance, about 2%, was seen when there were 5 scouts.

8.5.3 Effect of Document Base

When the Melee search was run on BaseB, the results as compared to BaseA were essentially the same as the Broadcaster cases. These results are shown in Table 15. Again evident was the remarkable growth and uniformity of the number of messages sent by the

scouts. Also, the elapsed times were over twice as high as the BaseA results.

Table 15: Melee, Effect of number of scouts, BaseB, topic = 14, weight = 0.6

| # of Scouts | Elapsed Time | Elapsed Time, sec | # of Recv Messages | # of Sent Messages | # of Docs Visited | Speedup |
|-------------|--------------|-------------------|--------------------|--------------------|-------------------|---------|
| 1 | 2:54 | 174 | 13351 | 406 | 218 | 1.00 |
| 2 | 1:37 | 97 | 220 | 11 | 229 | 1.79 |
| 5 | 1:31 | 91 | 218 | 14 | 232 | 1.91 |
| 7 | 1:37 | 97 | 218 | 19 | 237 | 1.79 |

8.6 Summary Comparison

Now that the results have been presented for Scoutmaster, Broadcaster, and Melee, the salient features of the scout types can be compared side by side in Table 16.

Table 16: All scout types, Comparison of results, topic=14, weight=0.6;

| Scout Type | Doc Base | Single Scout | | Best Case | | Speedup |
|-------------|----------|--------------|-------------|--------------|-------------|---------|
| | | Elapsed Time | No. of Msgs | Elapsed Time | No. of Msgs | |
| Scoutmaster | A | 496 | 8765 | 54 | 8795 | 9.19 |
| | B | 932* | 23737* | 66* | 23029* | 14.12* |
| Broadcaster | A | 76 | 4244 | 29 | 3064 | 2.62 |
| | B | 174 | 13778 | 120 | 6804 | 1.45 |
| Melee | A | 76 | 4244 | 35 | 5314 | 2.17 |
| | B | 174 | 13778 | 91 | 9463 | 1.91 |

note - values marked by an * are calculated rather than measured

The results for searching BaseB with the Scoutmaster are not actual values obtained in an experiment. As discussed in section 8.3.5, the experiments for this case failed because the message rate was too high, filling the JATLite message buffer. The values in the table were calculated by determining the ratio between the values in Tables 7 and 8 and multiplying the values in Table 6 by those ratios. This assumption is very optimistic because if the high numbers of messages that were calculated had been actually realized, the performance would have been substantially degraded due to the message congestion at high traffic rates described in Figure 11.

All of the scout types are very efficient; in their best cases they were able to complete a search of BaseA in less than a minute. The searches of BaseB took from about 20% longer for the Scoutmaster to about 4 times as long for the Broadcaster. Using only a single scout, the Scoutmaster had an elapsed time about 7 times longer than the Broadcaster and Melee. However, the Scoutmaster benefitted most by using multiple scouts. The best case for the Scoutmaster took less than twice as long as the best Broadcaster case.

The links and the weighting and type of the links were used to focus the search on the parts of the document sets that were germane to the query. In all cases, all of the paragraphs with the specified topic were found and the actual number of documents visited was only a few more than the number of documents that contained the desired information. Also, the documents that were pointed to by the semantic links did not have to be parsed. The link information directed the scouts to the appropriate paragraphs within the documents.

When the communications traffic was high, above 85 messages per second, there was a profound effect on the results. Because the Scoutmaster searched at a slower speed

than the other types, the effect of communication congestion was delayed until there was a large number of scouts. The Broadcaster and Melee types could not use more than 10 scouts or the search process would fail. In all cases, it appears as if the shortest elapsed time achievable for a search was about 30 seconds. If the JATLite router could be improved or another method of passing messages with reduced latency could be found, the search time could be reduced still further.

In summary, using a coordinated search through a semantically-linked hypertext document is very efficient and provides high precision and high recall. There are two sources of this efficiency. First, the distribution of labor among the scouts generally speeds the search by a factor equal to the number of scouts. Secondly, the addition of meta-information such as link types and weights reduces the area of the document base that is searched by the scouts.

CHAPTER 9

CONTRIBUTION AND FUTURE RESEARCH

9.1 Contribution

A new concept for information storage and retrieval was presented that has the potential for greatly improving recall and precision of information retrieval while substantially reducing search time. The principal contribution is the demonstration of the feasibility of using numbers of cooperating agents to traverse the links in a set of documents connected by semantic hypertext to quickly retrieve information. Experimentation showed that the potential for rapid, very accurate, retrieval of information is indeed realizable. The performance was achieved by using software agents, called scouts, working simultaneously in a coordinated manner. The distributed scouts were easy to construct in Java and required only simple communications protocols.

The scouts will work on any set of documents connected by hypertext links enriched with the semantic characterizations, type and weight. Because the hypertext is a slight variation of standard HTML, the documents may be widely distributed geographically, as long as they are network accessible. To reap the benefits of high recall, high precision, and efficient search provided by the cooperating scouts, researchers, the author included, will be motivated to discover efficient methods for constructing semantic hypertext links in such document bases. Because the scouts were written in Java and use a freely available Java agent framework, they will operate on practically any set of networked computers and could be used to search the constructed hypertext.

9.2 Future Research

Further research in this area will focus on the short term where there is incremental improvement in the current scout system and on the long term where more advanced methods must be developed to carry the concept into actual use.

In the near term, there are several potential areas of improvement. A more compact representation of the links in a document may speed the traversal of the links. One way to achieve this may be to keep the links in a header in the document or in a separate document that is referenced by the original one. The links would be indexed to exact locations within the original document. Indicators could still be placed at those locations to enable browsing. Because automatic methods for generating links in actual documents may not be developed for some time, the simulated document base could be made more representative of actual documents. The results of statistical analyses of real document sets could be used instead of the normal distributions that were assumed in this study. To better simulate actual user queries, the ability to accept Boolean combinations of topics or link types will be added.

The use of JATLite proved effective in the development of the scouts. The built-in nameserver and KQML handling capability proved to be real boon. The system, however, suffered from poor performance when the rate of message passing was high. Newer agent frameworks that have been developed since the study began will be explored. Specific capabilities that would be useful are direct communications rather than funneling all messages through a router and the ability to use computers with SMP architectures.

In the long term, research will address the two main aspects to the real world application of information retrieval through semantic hypertext. The first is the generation of

the hypertext links. The second, efficient, accurate, and cooperative navigation of those links, was the subject of the current study. Both areas offer many opportunities for future research.

9.2.1 Automatic Generation of Semantically-linked Hypertext

While the current research has demonstrated the efficiency of parallel coordinated search, the quality of the resultant information depends entirely on the quality of the semantic links. The construction of these links has to be automatic. While manual methods would certainly generate superior link semantics, the people creating the links (including authors) may not be able to create links to documents of others. In any case, whether generating links within a single document or linking a local document to a large number of documents authored elsewhere the effort would be prohibitively laborious. Thus the capability to automatically generate hypertext links is important and has attracted a number of researchers. An issue of Information Processing and Management journal was dedicated to that very topic.[34] All of the methods discussed in the special issue use IR techniques.[35]

All attempts at automatic generation of hypertext links, including this study, have depended on conventional IR means of determining similarity among texts. These methods, some of which were surveyed in Chapter 4, have been developed for the purpose of indexing documents to be searched in an automatic IR system. While much research is being done in this area, a candid observer would conclude that improvements in text indexing have reached a plateau. Conventional similarity detection with conventional IR indexing methods is not close to solving the problem of automatically developing these links.

Some exciting developments are taking place in another paradigm, that of evolutionary computing. Genetic algorithms have already been applied to IR. One technique [36], treats keywords as genes, sets of document keywords as individuals, and document collections as populations. The concept is to select a group of documents that best matches a user query. Other genetic algorithm applications have been used to fine tune the term weightings used in the vector space model of conventional IR.

Another method to be considered is neural networks, in which the software is trained via trial and error to solve a specific problem. Neural networks have been used extensively in pattern matching. Perhaps the technique can be extended to semantic matching.

Both evolutionary computing and neural networks are inherently massively parallel and are being applied to a number of problems that have been resistant to deterministic, equational methods. The determination of relevance of distributed chunks of knowledge would be an exciting challenge for the application of evolutionary methods.

Techniques for specifying the relevance of distributed knowledge that go beyond simple link types and weights should be developed. The goal here would be to give the scouts as much meta-information as possible to enhance their information gathering ability. One potential research path is the development of a standard language for expressing the semantic relationships. A promising candidate, or at least a framework for developing one, is the Extensible Markup Language (XML) [39] currently being applied in WWW-based business applications. With XML, the user can develop new tags that define the meaning of information that appears in a WWW document.

9.2.2 Scout Cooperation

The characteristics of the information gathering scouts are inextricably tied to the methods of linking. There are some areas of research, however, that can be pursued independently. Fortunately, communication among objects distributed over a computer network is currently receiving great attention. Standards, such as CORBA and DOM, are being developed and applications that employ the standards are being implemented. The Java language has facilities, such as Remote Method Invocation and Enterprise JavaBeans for developing distributed applications. Computational frameworks for intelligent agents are being developed as well.

Additional work needs to be done in developing languages for scout communication. Although KQML performed admirably in this research, it is relatively new and untried. Additional implementations of KQML should be developed, such as one that can handle XML, so that lessons can be learned and improvements can be made or entirely new languages can be proposed.

When many scouts are navigating a knowledge base and finding information that is relevant to the specified query, the user could be overwhelmed by the response. Methods for eliminating redundant information and fusing knowledge that comes from several sources and in several formats must be developed.

Finally, as the amount of information reaches even more staggering proportions, it may be necessary to deploy millions of scouts for some applications. While it has been demonstrated that for relatively few scouts, the performance improves linearly with the number of scouts, this relationship does not hold when there are many of them. As shown in chapter 8, high network traffic rates cause communications bottlenecks that degrade the

performance of large numbers of scouts. This degradation was particularly acute in the Broadcaster and Melee scouts. These two methods were unscalable to more than a small number of scouts because of their very high message rates. To take advantage of the potential speed increase due to scalability, research into algorithms for super massively parallel systems should be explored.

CHAPTER X

BIBLIOGRAPHY

1. van Rijsbergen, C. J., Information Retrieval, 2nd Edition, London, Butterworths, 1979.
2. Feldman, S.: "The Internet Search-Off," in *Searcher*, Vol. 6, No. 2, February 1988, - <http://www.infoday.com/searcher/feb98/story1.htm>
3. Smeaton, A., "Overview of Information Retrieval", in *Information Retrieval and Hypertext*, pp. 3-25, edited by Agosti, M., and Smeaton, A., Kluwer Academic Publishers, London, 1996.
4. Jacobs, P. S. and Rau, L. F.; *SCISOR: Extracting Information from Online News*, Communications of the ACM, Vol. 33, No. 11, pp. 88-97.
5. Mauldin, M. L.; Conceptual Information Retrieval: A Case Study in Adaptive Partial Parsing, Kluwer Academic Publishers, London, 1991.
6. Bush, Vannevar; *As We May Think*, Atlantic Monthly, pp. 101-108, July 1945.
7. Englebart, D. C.; "A Research Center for Augmentating Human Intellect", in *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference*, San Francisco, CA, pp. 395-410, December 1968.
8. Digital Libraries Initiative, <http://www.dli2.nsf.gov/>.
9. WebAnts - <http://206.101.96.68:8001/webants/> (No longer active)
10. Center for Intelligent Information Retrieval at University of Massachusetts - <http://ciir.cs.umass.edu/>
11. Woelk, D. and Tomlinson, C.: "Intelligent Search Management vis Semantic Agents," WWW '94 Conference.
12. Agosti, M. "Hypertext and Information Retrieval," in *Information Processing and Management*, Vol. 29, no. 3, pp. 283-285, 1993.
13. Lucarella, D. and Zanzi, A.: "Information Retrieval from Hypertext: An Approach using Plausible inference," in *Information Processing and Management*, Vol. 29, No. 3, pp. 299-312, 1993.
14. Salton, G., "Developments in Automatic Text Retrieval", in *Science*, Vol. 253, pp. 974-979, August 1991

15. Salton, G. and Allan, J.; "Selective Text Utilization and Text Traversal," Department of Computer Science, Cornell University. TR 93-1366, July 1993.
16. Allan, J. Automatic Hypertext Construction, Ph.D dissertation, Cornell university, January 1995.
17. Green, S. J.; Automatically Generating Hypertext by Computing Semantic Similarity, Ph.D dissertation, University of Toronto, October 1997.
18. Salton, G., ed.: The SMART Retrieval System---Experiments in Automatic Document Processing, Prentice Hall, Inc, Englewood Cliffs, N. J., 1971.
19. Latent Semantic Indexing - [ww.cs.utk.edu/~berry/lsi++/node5.html](http://www.cs.utk.edu/~berry/lsi++/node5.html)
20. Dumais, S. T.. "LSI meets TrEC a Status Report"; In: D. Harman (Ed.), *The First Text REtrieval Conference (TREC1)*, National Institute of Standards and Technology Special Publication 500-207 , pp. 137-152.
21. NASA Thesaurus - <http://www.sti.nasa.gov/thesfrm1.htm>
22. Miller, G. A. Beckwith, R., Fellbaum, C., Gross, D. and Miller, K.: "Introduction to WordNet: An Online Lexical Database". August 1993.
23. Managing Gigabytes - <http://www.mds.rmit.edu.au/mg/>
24. TREC collection - <http://trec.nist.gov/>
25. CACM collection - <http://www.acm.org/pubs/contents/journals/cacm/>
26. Langley Technical Report Server - <http://techreports.larc.nasa.gov/ltrs/ltrs.html>
27. HTML 4.0 spec - <http://www.w3.org/TR/REC-html40/>
28. Finin, T., McKay, D., Fritzson, R., and McEntire, R.: "KQML: An Information and Knowledge Exchange Protocol", in Kazuhiro Fuchi and Toshio Yokoi (Ed.), *Knowledge Building and Knowledge Sharing* , Ohmsha and IOS Press, 1994
29. Fini, et. al.; "Specifications of the KQML Agent Communication Language," The DARPA Knowledge Sharing Initiative External Interfaces Working Group June 1993.
30. SodaBot - <http://www.ai.mit.edu/people/sodabot/sodabot.html>
31. IBM Aglets - <http://www.trl.ibm.co.jp/aglets/>
32. Java Agent Template - <http://www-cdr.stanford.edu/ABE/JavaAgent.html>.

33. JATLite - http://java.Stanford.EDU/java_agent/html/
34. Agosti, M, and Allan, J.; "Introduction to the Special Issue on Methods and Tools for the Automatic Construction of Hypertext," in *Information Processing and Management*, Vol. 33, No. 2, pp.129-131, 1997.
35. Agosti, M, Crestani, F., and Melucci, M.; "On the Use of Information Retrieval Techniques for the Automatic Construction of Hypertext," in *Information Processing and Management*, Vol. 33, No. 2, pp.133-134, 1997.
36. Chen, H.; "Machine Learning for Information Retrieval: Neural Networks, Symbolic Learning, and Genetic Algorithms," in *Journal of the American Society for Information Science*, Vol 46, No. 3, April 1995.
- 37 XML (Extensible Markup Language) - <http://www.w3.org/XML/>

VITA

John Joseph Rehder

Born in Wilmington, North Carolina, September 21, 1947. Graduated from William Fleming High School in Roanoke, Virginia in June 1965. B.S. Aerospace Engineering, Georgia Institute of Technology, 1970. M.S. Flight Sciences, George Washington University, 1975. M.S. Computer Science, Old Dominion University, 1986.

The author is currently employed in the Computational AeroSciences Team at NASA Langley Research Center in Hampton, Virginia