C. Project Description

1 Results from Prior NSF Support

The first PI was supported by NSF grant IRI-9357772, "Softbots as Testbeds for AI" over the award period June 16, 1993 to June 15, 1998; this award is an NSF Young Investigator Award in the amount of $312,500 (including matching funds).

Graduating Ph.Ds. supported by the grant include: Richard Segal (IBM Watson Research Center, 1996); Neal Lesh (Mitsubishi Electric Research Labs, 1997); Keith Golden (NASA AMES Research Center, 1997, co-advised with Dan Weld). The grant also funded Masters dissertations for Jonathan Shakes 1996; and Terrance Goan 1994.

"In order to do good research you must have a secret weapon."
–Herb Simon

Our "secret" weapon has been the *softbots* (*soft*ware ro*bots*) project. A softbot is an "intelligent agent" that interacts with a software environment, such as Unix or the World Wide Web, by issuing commands and interpreting the environment's feedback.

We began to experimentally investigate softbots in 1991, beginning with the Unix softbot [24], moving to the Internet softbot [21], and its specialized successors: MetaCrawler [55], Ahoy [57], Occam [36], and Razor [25]. While there are many differences between the individual softbots, several central principles emerged:

- **Goal oriented**. A user should be able to state *what* he or she wants accomplished. The softbot should determine *how* and *where* to achieve that goal, then perform the actions with minimal supervision.
- **Integrated:** Users dislike being forced to remember the details of particular databases or the wide and growing variety of Internet services and utilities in order to use them effectively. Instead, a softbot should provide a single, uniform interface to all such services.
- **Expressive**. If common goals are impossible to specify, then a goal-oriented softbot is of limited use. Users should be able to request tasks such as "Get all of Tate's technical reports that aren't already stored locally." The softbot should handle this goal even though the underlying services (*e.g.*, `ftp`) does not handle this combination of universal quantification and negation.
- **Cooperative:** Instead of issuing an error message in response to incorrect or incomplete specifications, a softbot should collaborate with the user in order to build a reasonable request.
- **Customizable:** Softbots should adapt both to the environment and to their users. A combination of learning from experience and guidance from direct requests from the user should guide this process.

From a methodological perspective, our softbot research combines a dedication to fielding robust, implemented systems on the Internet with a desire to discover and precisely analyze novel algorithms and representations. Below, we discuss our experience with these twin aspects.

1.1 Softbots Deployed on the Web

In order to demonstrate the potential of the softbot paradigm, to validate our ideas, and to gain valuable experience with real users, we have deployed the following softbots on the Web:

- MetaCrawler (1995) — a softbot that queries multiple general-purpose search engines in parallel, collates the results, and presents its user with a single ranked results list [55]. In order to accomplish its goal, MetaCrawler must be adept at several tasks. First, it needs to take user queries and format

them appropriately for each search service. Next, it must be able to correctly parse the results and aggregate the results from each service. These steps are accomplished using a set of programs called *wrappers* that manage communication between each underlying engine and the MetaCrawler; the MetaCrawler maintains a wrapper for each engine it queries. A key element of our research program is scaling up from nine wrappers (in MetaCrawler) to thousands in the proposed Reference Librarian for the Web.

As MetaCrawler became incresingly popular, the cost of maintaining it increased. As a result, the University of Washington licensed MetaCrawler to a local company, Go2Net, Inc. At Go2Net, MetaCrawler has become the leading meta-search service on the Web at `www.metacrawler.com`. However, we have retained full rights to a research version, accessible at `huskysearch.cs.washington.edu`.

- Ahoy! The Homepage Finder (1996) — a softbot that finds people's homepages with high accuracy [57]. "Less is more" is the relevant motto. While search engines attempt to generate hundreds of responses to a given query, Ahoy! attempts to return a small number of high precision references. In our experiments, Ahoy! proved to have both better precision *and* recall than standard search engines. For example, on 74% of the queries from our test sample, Ahoy! found the target homepage and ranked it as the top reference. In contrast, Alta Vista was only able to find 58% of the target pages and ranked only 23 % as its top reference.

  To improve with experience, Ahoy! relies on an unsupervised learning algorithm that attempts to discover the conventions underlying home page placement at different institutions. For example, Ahoy! learns that home pages at the University of Washington's Computer Science Dept. typically have the form `http://www.cs.washington.edu/homes/<lastname>`. In our experiments, 9% of the home pages located by Ahoy! were found using learned knowledge of this sort.

  Finally, we have shown how Ahoy!'s *dynamic reference sifting* architecture is applicable to a broad range of information needs from locating recipes to finding train schedules. The paper on Ahoy! and *dynamic reference sifting* was recognized as the Runnerup for the Best Paper Award in WWW6. See `www.cs.washington.edu/research/ahoy`.

- Shopbot (1997) — a parallel-access, comparison-shopping softbot [16, 52]. Shopbot used a combination of heuristic search, pattern matching, and inductive learning techniques to semi-automatically learn resource models of online stores (including their wrappers). Shopbot was the precursor of Jango, which now has over 300 wrappers and is available as the Excite Product Finder at `jango.excite.com`.

1.2 Research Highlights

The softbots project has proven to be a successful research vehicle; our research contributions include novel action representation languages [23, 28], tractable algorithms for planning with incomplete information [30, 29, 36], provably fast closed-world reasoning mechanisms [18, 25], learning algorithms for automatically generating wrappers [53, 52, 16, 35, 34], mechanisms for automatically recognizing user goals by observing their behavior [38, 39], a theoretical study of softbot safety [58], and numerous experimental studies validating the different algorithms and associated softbots.

We briefly sketch the most important of these contributions below:

- **Action Representation Languages:**

  - Classical planners presuppose complete and correct information about the world. These assumptions are clearly untenable for a softbot. In response, we have designed UWL, a representation for information goals and sensing actions that facilitates planning with incomplete information. We provided UWL with clean semantics and specified a provably correct planning algorithm based on the language [23]. While logics of knowledge such as those developed by Morgenstern and Moore are more expressive than UWL, UWL is sufficiently expressive to encode software commands, and is simple enough to incorporate in a planner.

- To build effective planning systems, it is crucial to find the right level of representation: too impoverished, and important actions or goals are impossible to express; too expressive, and planning becomes intractable. Under the classical framework, Pednault [49] provided a happy compromise between the impoverished STRIPS representation, and the expensive situation calculus.

  Motivated by ADL [49] (which is in the middle ground between impoversished STRIPS and the intractable situation calculus) we defined a novel action language, SADL, that represents the middle ground for reprsenting sensing actions. Our SADL language, which combines of UWL [23] and ADL, is based on two insights: 1) Knowledge goals are inherently temporal; the failure to support this weakened UWL. 2) Knowledge preconditions are unnecessary for an important class of domains (those obeying a knowledge-free Markov property); SADL focuses on this class. SADL is expressive enough to encode the rich domain theory of the Internet Softbot, including hundreds of UNIX and Internet operators.

- **Tractable Algorithms for Planning with Incomplete Information:**

  - Although recent work has sketched a number of algorithms for planning with incomplete information (*e.g.*, [1, 46, 33, 51, 23, 20, 26]), substantial problems remain before these planners can be applied to real-world domains. Since the presence of incomplete information invalidates the Closed World Assumption, an agent cannot deduce that a fact is false based on its absence from the agent's world model. This leads to two challenges: satisfying universally quantified goals and avoiding redundant sensing.
  - In [30] we report on the fully-implemented XII planner[1] which addresses these challenges. We allow incomplete information in the initial conditions, and uncertainty in the effects,[2] but assume the information that *is* known is correct, and that there are no exogenous events. XII's planning algorithm is based on UCPOP [50], but XII interleaves planning and execution and, unlike UCPOP, does not make the closed world assumption.
  - In [36] we describe OCCAM, a query planning algorithm that determines the best way to integrate data from different sources. As input, OCCAM takes a library of site descriptions and a user query. As output, OCCAM automatically generates one or more plans that encode alternative ways to gather the requested information.

    OCCAM has several important features: (1) it integrates both legacy systems and full relational databases with an efficient, domain-independent, query-planning algorithm, (2) it reasons about the capabilities of different information sources, (3) it handles partial goal satisfaction *i.e.*, gathers as much data as possible when it can't gather exactly all that the user requested, (4) it is both sound and complete, (5) it is efficient. We present empirical results demonstrating OCCAM's performance on a variety of information gathering tasks.

- **Closed-world Reasoning Mechanisms:**

  - Closed-world inference — an essential component of many planning algorithms — is the process of determining that a logical sentence is false based on its absence from a knowledge base, or the inability to derive it. We developed a novel method for closed-world inference and update over the first-order theories of action used by planning algorithms. We showed the method to be sound and efficient, but incomplete. In our experiments, closed-world inference consistently averaged about 2 milliseconds while updates averaged approximately 1.2 milliseconds. Furthermore, we demonstrate that incompleteness is nonproblematic in practice, since our mechanism makes over 99% of the desired inferences. Our work was reported in [22, 18].
  - We designed and implemented the XII [30] and PUCCINI [29, 27] planners. These algorithms interleave planning and execution to control the Internet Softbot [21]. Both planners

[1] XII stands for "eXecution and Incomplete Information."

[2] All effects of operators must be specified, but XII supports a three-valued logic which allows us to specify a limited form of uncertainty in the effects.

use the SADL action language described above. In addition, the planners use local closed world (LCW) inference to handle universally quantified goals and to avoid the problem of redundant sensing. Our technical innovations include the LCW machinery (effects, goals, and novel techniques for resolving threats to LCW links) and the LCW-based pruning techniques which solve the problem of redundant information gathering. Experiments show that these techniques cut the number of actions executed by the Softbot by a factor of one hundred, and resulted in a corresponding speedup to the planner.

- In [25] we describe RAZOR, a planning-based information-gathering agent that assists users by automatically determining which Internet information sites are relevant to their query, accessing those sites in parallel, and integrating the results. RAZOR uses a disjunctive graph-based plan representation. It then uses a novel and powerful form of local completeness reasoning in order to transform those plans into contingent plans of high quality. These contingent plans can be efficiently executed, obtaining more answers at less cost than the original plans.

- **Machine Learning Algorithms for Automatically Generating Wrappers:**

  - Due to the increasing number and diversity of information sources on the Internet, the softbot is forced to contend with incomplete information about available resources. In response we have developed ILA, a program able to autonomously learn models of information resources available on the Internet. ILA learns the meaning of external information by explaining it in terms of internal categories. In our experiments, ILA starts with knowledge of local faculty members, and is able to learn models of the Internet service WHOIS and of the personnel directories available at Berkeley, Brown, CalTech, Cornell, Rice, Rutgers, and UCI, averaging fewer than 40 queries per information source. ILA's learning task and its learning method are novel, demonstrating the potential of the softbot for inspiring new directions in machine learning [53].
  - The World-Wide-Web is less agent-friendly than we might hope. Most information on the Web is presented in loosely structured natural language text with no agent-readable semantics. HTML annotations structure the *display* of Web pages, but provide virtually no insight into their *content*. Thus, the designers of intelligent Web agents need to address the following questions: (1) To what extent can an agent understand information published at Web sites? (2) Is the agent's understanding sufficient to provide genuinely useful assistance to users? (3) Is site-specific hand-coding necessary, or can the agent automatically extract information from unfamiliar Web sites? (4) What aspects of the Web facilitate this competence?

    We investigated these issues with a case study using Shopbot, a fully-implemented, domain-independent comparison-shopping agent. Given the home pages of several online stores, Shopbot autonomously learns how to shop at those vendors. After learning, it is able to speedily visit over a dozen software and CD vendors, extract product information, and summarize the results for the user. Preliminary studies show that Shopbot enables users to both find superior prices and substantially reduce Web shopping time [16].

    Remarkably, Shopbot achieves this performance without sophisticated natural language processing, and requires only minimal knowledge about different product domains. Instead, Shopbot relies on a combination of heuristic search, pattern matching, and inductive learning techniques.
  - In [35] we invented *wrapper induction* a new technique for automatically constructing wrappers for information extraction. Our work makes five contributions. First, we have formalized the wrapper construction problem in terms of inductive learning. Second, we have defined the HLRT bias, which is efficiently learnable in this framework. Third, we used the definitions of PAC learning theory to provide bounds on the sample complexity — how many interactions with a source are necessary before one can induce a reliable wrapper. Fourth, we implemented a java enhancement to Microsoft's Internet Explorer that allows a human to quickly label search result pages for learning. Fifth, we have shown how to use imperfect, heuristic recognizers to perform completely autonomous wrapper induction in some cases.

- **Softbot Safety:** As powerful software agents come closer to deployment, the issue of agent safety becomes more pressing. Recently, we broke ground in this area, suggesting computationally tractable techniques for ensuring softbot safety and posing a number of open problems [58].
- **Softbot goal recognition:** The ability to identify a user's goal by observing his or her actions is handy for a softbot. We have devised a sound and polynomial-time goal recognizer and tested it on human subjects executing Unix commands. We have found the goal recognizer to be both fast and effective in practice. Our approach adapts techniques from inductive concept learning to the field of plan recognition [38, 39].

2 Proposed Research

By all accounts, the Web is humanity's largest and fastest growing repository of digital information. Many collections of information are Internet-accessible, and most will provide a searchable Web interface. While some collections have a broad array of materials, trends show an explosion in the number of *specialized collections* with narrow but very deep content. Thus a principle challenge facing users will be the selection of Web information sources capable of answering their query. In a physical library, users rely on a reference librarian to help point them at the correct resource, but while human librarians are becoming increasingly sophisticated in their use of the Web, they are only part of the solution. We need more powerful automatic reference tools to help people efficiently retrieve high quality information from the Web.

Typically, reference librarians are not specialists in the topic of inquiry (e.g., computational fluid dynamics) but they *are* expert at *identifying relevant resources* (e.g., The International Journal of Fluid Dynamics) and at *appropriate strategies for obtaining the necessary information.* The central objective of this proposal is to create software agents that posses *reference intelligence* — a limited understanding of complex technical topics, but a very sophisticated understanding of how and where to find high-quality information on the World Wide Web.

Existing techniques have not achieved reference intelligence. The number of specialized collections is growing so quickly than manually created directories (e.g. Yahoo) cannot keep up. It is estimated that there are over a million specialized information sources today. For example, there are over 1000 weather forecasting and data repositories and 600 online book stores. Many high-quality sites are extremely specialized: searchable indices to the complete works of Shakespeare, nutritional information for foods, historical hydrological information for rivers in Western Washington, etc. Existing techniques for indexing the Web (e.g. spiders such as Alta Vista) also fail for our purposes. For example, a recent article in the journal Science [37] claims that only 20% or so of the pages on the Web are indexed by today's search engines. This deficit is exacerbated by the fact that many libraries store information in a database system and only materialize them in response to a query through a CGI bin script. These non-materialized Web pages form an increasingly important "Hidden Web" that can never be found by a spider.

In contrast, our system will outperform previous approaches by tracking specialized information sources, automatically directing a user query to the best subset of these digital collections, and displaying a summary of the relevant information received. In order to build this Reference Librarian agent, we will confront the following research questions:

- **Automatic Modeling of Specialized Information Sources:** We will investigate novel techniques for automatically generating models of Web information sources including discovering their location, identifying their topic, learning how to query them and parse their responses. Our approach will be based on the machine-learning algorithms we introduced in [53, 16, 35].
- **Intelligent Routing of User Queries:** A key element of the Librarian's intelligence is to identify the set of information sources likely to be *relevant* to the current query. We plan to fully automate this routing process, over thousands of information sources, by introducing a Bayesian Network [48] that will infer the query's topic based on background knowledge drawn from WordNet [44], from large-scale technical dictionaries, and from the results of previous queries.

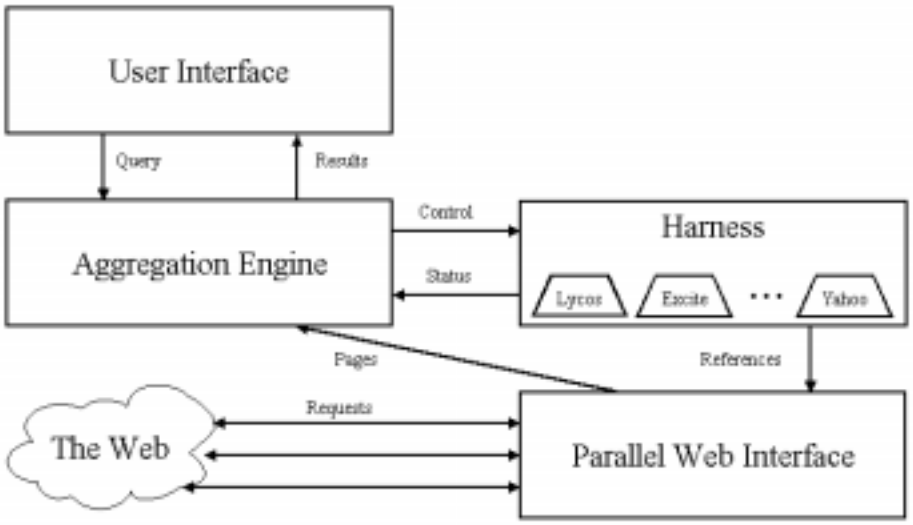

Figure 1: The basic Web Librarian Architecture. In subsequent sections we discuss how to extend the architecture with intelligent query routing, automatic modeling of information sources, and algorithms for approximating optimal query scheduling.

- **Scheduling Parallel Access to Information Sources:** Accessing all potentially relevant sources in sequence is prohibitively slow, but accessing all sources in parallel is prohibitively expensive (in terms of load on the sources and the Internet). Thus, we plan to introduce fast algorithms for query scheduling based on our theoretical analysis in [19].

Below, we briefly sketch the Web Librarian's architecture and then consider each of these challenges in turn; we conclude with a discussion of related work and with milestones for our project.

2.1 The Web Librarian Architecture

A simple, modular architecture, based on the MetaCrawler system [55, 56], should form an excellent foundation for our proposed Web Reference Librarian. We plan to extend the architecture with intelligent query routing, automatic modeling of information sources, and algorithms for approximating optimal query scheduling as discussed in subsequent sections. Below, we outline the Librarian's basic architecture (see Figure 1).

Given a user query, the Librarian formats it appropriately for each search engine it plans to query, and then queries the engines in parallel. Next, it parses the results as they come back from each engine, collates them into a single, coherent list factoring in the ranking information provided by each engine, eliminating duplicated information, etc.

The Librarian also supports a powerful language for expressing queries including the ability to specify phrases, specify weights on different words in the query, and more. Many engines do not support such features, but the Librarian can automatically implement these and other features not available from any engine by downloading the pages referred to by each engine it queries and analyzing them directly (e.g., searching the body of the page for the presence of a desired phrase). The time required to download all pages in parallel is reasonable, usually adding no more than a couple of minutes to the search time. In addition, the Librarian is able to process results that have arrived while waiting for others. Therefore, minimal computation time required after all the references have been retrieved. A desirable side effect of this Librarian capability is that the Librarian is able to verify that pages returned by its underlying engines actually exist and are available for perusal by a person.

The Librarian is designed in a modular fashion, as depicted in Figure 1. The main components are the User Interface, the Aggregation Engine, the Parallel Web Interface, and the Harness. The User

Interface is simply the layer that translates user queries and options into the appropriate parameters. These are then sent to the Aggregation Engine, which is responsible for obtaining the initial references from each service, post-processing the references, eliminating duplicates, and collating and outputting the results back to the User Interface for proper display. The Parallel Web Interface is responsible for downloading HTML pages from the Web, including sending queries and obtaining results from each information resource.

The Harness is the crux of the design; it's where the service-specific information is kept. The Harness receives control information detailing which references to obtain; it then formats the request properly and sends the reference to the Parallel Web Interface, which then returns a page to the Aggregation Engine. It also sends some status information back to the Aggregation Engine that is used to measure progress. The Harness is implemented as a collection of wrappers, where each wrappers represents a particular service. The harness is designed so that wrappers can be added, modified, and removed dynamically without impacting the rest of the Librarian.

The Librarian's architecture provides several advantages. It provides a layer of abstraction above traditional search services, which allows for greater adaptability, scalability, and portability as the Web grows and changes. Information resources are extremely volatile in today's Internet. New resources are being launched continually, while others languish. For this reason we have chosen a modular architecture; we are able to plug in, modify, and remove wrappers easily. This enables us to combine many different resources without difficulty. It is adaptable to new and modified services. Because of its minimal resource requirements, it is very portable, able to exist as a group server on enterprise-scale servers or even as an individual user's application on a PC or Macintosh.

2.2 Automatic Modeling of Specialized Information Sources

Our MetaCrawler and Jango systems have demonstrated the power of simultaneous searching specialized information sources in parallel, but the efficacy of this process is determined by the *number* of *appropriate* sources which can be brought to bear on an individual query — the more distinct and relevant sources searched, the better the results. At present, Jango and Search Broker represent the state-of-the-art: each with a library of approximately 300 wrappers from which to choose. Despite a significant investment in wrapper-writing tools, it is both tedious and expensive to *manually* create and maintain a large library wrappers.

This last point is worth emphasizing. In order for softbots to automatically use thousands of information sources, they need thousands of wrappers. In addition, information sources often change their presentation format, so the existing wrappers need to be updated regularly. Handling the huge variety of information sources on the Internet demands that wrappers be created with little or no human involvement. It also requires automated monitoring code that notices when an wrapper no longer works and prevents softbots from using it until it is updated. We believe that the best way to construct a system with an order of magnitude increased number of wrappers is to create them completely automatically. Figure 2 diagrams the architecture of the offline automatic modeling process that will support the Librarian's query processing. Building the modeling system means confronting the following research problems:

1. Finding potential information sources
2. Categorizing the topical expertise of each source
3. Determining how to query the information source
4. Determining how to parse the information source
5. Gathering statistics on source speed, reliability and prominence

Of these challenges, number 2 is hardest and most critical to the overall performance of our automated librarian because the accuracy of our topical expertise model determines the quality of query routing (section 2.3). Nevertheless, all problems are important so we address them each in turn below.

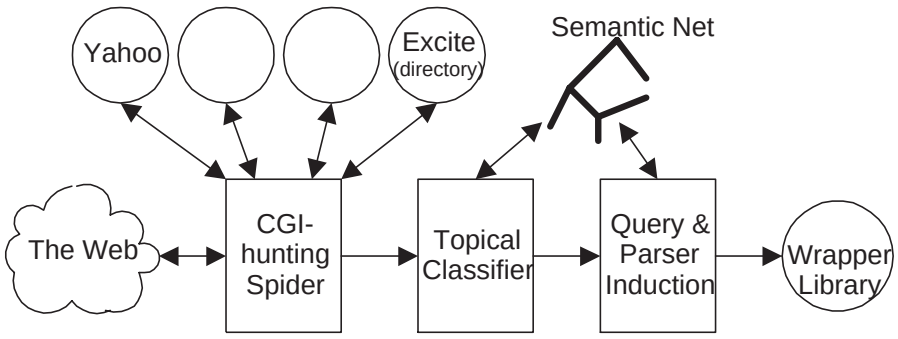

Figure 2: Offline Wrapper Learning System.

2.2.1 Locating Potential Information Sources

We expect it will be relatively easy to construct an initial database of information sources. The first step is to take the union of existing compendia such as `search.com` list of 400 search engines and the sources handled by Search Broker. These are high-quality information sources selected by a strong editorial process. We plan to cull these compendia for specialized scientific resources relevant to our Reference Librarian.

A more sophisticated approach will be to run a CGI-hunting spider over the sites listed at Yahoo (and in similar directories hosted by competitive portals). The spider would perform a page-limited, domain-bounded,[3] breadth-first search of the pages accessible from the page linked by Yahoo. At each point during this traversal, the spider would look for HTML get and post commands which signify CGI scripts implementing searchability.

Our preliminary research suggests that only a small minority of extant CGI scripts are useful for our purposes; the remainder implement user-registration, password entry, marketing-data collection, or email feedback forms. Therefore, we will use a set of heuristic filters to eliminate these pages from consideration. However, even if only 5% of Yahoo sites contain CGI scripts and only 10% of those denote a searchable information source, we shall build a database of many thousand potential information sources. We could find even more by using Alta Vista or Hotbot to search for pages containing HTML get and post commands, but we expect that this would yield too many sites. By taking the intersection of searchable sites with those classified by Yahoo, we gain the benefit of Yahoo's investment in editorial selectivity.

2.2.2 Categorizing Topical Expertise

Given the URL of a page corresponding to a specialized search engine, our system needs to determine the topic areas for which this information source can provide useful information. Our approach is based on the existence of a comprehensive semantic network of topic areas (described in section 2.3.2). Thus the process of topical categorization reduces to the problem of mapping the information source into one or more nodes in the semantic network.

Initially, we will try two techniques for creating this mapping, refining our approach once we understand their strengths and limitations:

- Create a text corpus by combining the text on the search form itself with text on surrounding pages (follow hyperlinks to depth d) at the same site (e.g. same domain name). We can also use the text on page that point *to* the search form. Next, we use term frequency / inverse document frequency (TF/IDF) and possibly latent semantic indexing to calculate the match strength between topic nodes in the semantic network and the text corresponding to the information source.

- Recall from section 2.2.1 that we will have likely located the information source by means of a spider traversing a directory such as that of Yahoo. Thus, the location of the source in the Yahoo

[3] I.e. the search would not follow links which pointed to pages hosted by a server with a different domain.

taxonomy provides ideal information for categorizing the source — the remaining problem is to map Yahoo's categories to the Librarian's internal semantic network. Our first effort will use the TF/IDF approach unless it proves inadequate. If an information source is indexed in multiple directories, then we can either union the individual descriptions for a larger corpus, or take the average of the separate mappings.

2.2.3 Learning to Query & Parse

Querying a page with a single searchable form is straightforward — we simply need to include code for handling both get and post methods. Handling pages with multiple forms is much more difficult because the Librarian would need to know which parts of the query to map into the different forms. For example, Amazon.com has separate search forms for Author name, title, and subject. Since the only foolproof way to choose the correct form requires deep semantic knowledge of the query string (e.g. "Basic Field Hydrology" is more likely a title than an author name), we plan to sidestep this problem in our initial implementation. Our preliminary survey of web information sources suggests that almost all sites (including Amazon) with sophisticated, multi-form input also provide a "quick search" feature which is comprised of a single form. The CGI-hunting spider will be directed to look for pages with single forms.

Automatically constructing a parser for page results is more challenging. We are heartened by a preliminary survey of resources which indicates that 70% of the information sources listed at `search.com` are in the HLRT [35] (or a closely related [34]) wrapper class. This means that polynomial-time inductive learning is feasible with a small set of labeled examples. However, we think we can vastly improve upon the results of [34] and automatically derive a parser for as much as 90% of information sources using the following approach.

First, we note that at a certain level of abstraction all information sources are the same — the user issues a query and the source returns a list of n responses (each is typically a URL with a bit of descriptive text). This is important for two reasons: 1) each information source shares the same relational schema (a URL, text snippet pair), and 2) the parser required to extract this information is both simple and *fault tolerant*. Specifically, it is easy to extract URLs because of their well defined format and extraction of a text snippet is robust because the snippet will be understandable to the user even if a word is missing on one end or an extra word is accidentally included on the other. Contrast this with the wrappers needed by `jango.excite.com` to parse specifications of desktop personal computers — the Jango wrapper must extract nine fields: manufacturer, model, cpu speed, memory size, hard disk size, CD speed, modem speed, and price — and an offset error would be catastrophic.

We expect to try several approaches to learning parsers, but our first implementation will work as follows. The learner will first issue a series of "known failure" queries to the source using nonsense strings such as "frobinium57" to derive a recognizer for the source's "Your query returns no matches" error response. next, the learner will issues a sequence of heuristically composed queries using words found on the site's page and words suggested by Wordnet given our candidate catagorical match. Any resulting pages which differ from the failure response will form the training corpus for the parser. The learner will then generate an abstract HTML parse tree for the page and look for regularities as described in [16]. In a nutshell, the learner will exploit the fact that a successful query will typically generate enough hits to cause a regular, repeating pattern. Once this pattern is recognized, a simple parser is easy to construct.

2.2.4 Source Speed, Reliability and Prominence

In addition to creating a wrapper for each resource and mapping its topic area into the Librarian's semantic network, our system will build and maintain statistics about the site's speed (mean time to open an HTTP connection, and average data transport rate) and reliability (percentage of connections which were refused or timed out). Estimating the accuracy of an information source is more difficult, but we plan to approximate this metric by measuring prominence — what percentage of authoritative directories maintain links to this information source? To some extent this approach simply pushes the quality determination back one level — which sites are "authoritative?" But there are two answers

to this challenge. First, certain sites *are* known to have high quality based on their editorial process: Yahoo, `search.com`, the Internet Public Library (`www.ipl.org/ref/RR`) as well as the directory (i.e. "channel") sections of Excite, Lycos, NetCenter and other portals. A second, more interesting answer is that there *are* hyperlink patterns that indicate that a particular site is authoritative as argued by Kleinberg in [32]. Third, as discussed in section 2.3.4, we plan to use the behavior of users interacting with the Librarian (monitor their anonymous clicks as they choose references suggested by different sources) to provide feedback to update the source's score. Our implementation plan is to start by using a set of a dozen known authorities, with update, and then explore Kleinberg's ideas as time permits.

2.3 Intelligent Query Routing

Query routing is the problem of computing a mapping from a researcher's query to the set of relevant specialized resources. Below, we propose to achieve accurate query routing via a novel combination of semantics networks [6, 60], Bayesian networks [48], and large scale technical dictionaries.

Clearly, the exceedingly simple routing strategy implemented by MetaCrawler ("ask everyone") does not scale to a large number of specialized search engines. First, it will impose an undue load by sending a large number of irrelevant queries to specialized resources that are not geared for a high volume of queries. Second, it will generate a lot of irrelevant responses. For example, a query on Jupiter can provide high quality information from an astronomy site, and irrelevant references from a mystery book search service. Finally, an increasing number of sites charge for information (e.g., the New York Times archives), so a profligate query strategy can be expensive.

Softbots such as Savvysearch (`guaraldi.cs.colostate.edu:2000/form`) have taken the first step towards sophisticated query routing by grouping some twenty search engines into labeled categories (e.g., software searches versus entertainment searches) and enabling the user to specify which category she wishes to search. However, Savvysearch does not have the intelligence for accurate automatic query routing. Thus, as the number of specialized engines and topics increases, the user is effectively left to decide what is the set of potentially relevant engines to query. This decision is particularly difficult when queries correspond to more than one of the pre-defined categories (e.g., consider a query that is relevant to both "children" and "medicine").

We expect that our Librarian will have access to a large number (in the thousands) of specialized search engines and will encounter an even larger number of distinct queries (in the millions). Furthermore, we assume that the approximate scope of each specialized engine is known; we discuss how this information can be learned and updated in Section 2.2. Given these assumptions, query routing proceeds in four phases: term labeling, topic computation, routing, update. A depiction of our query routing architecture appears in Figure 3.

2.3.1 Term Labeling

Queries are likely to include highly technical jargon drawn from the field of interest. Technical vocabulary often provides excellent clues as to the query's topic and thus the appropriate set of search engines. Thus, the first step in query routing is to attempt to label any piece of jargon (word or phrase) with its "origin." Thus, "precordial capnograph" belongs to the field of pulmonary medicine whereas "Schmidt-Cassegrain" refers to telescopes. To facilitate powerful term labeling we plan to collect a large number of on-line technical dictionaries and databases. Powerful indexing methods enable us to search through literally millions of terms at lightening speeds. Finally, each term that is identified during the term labeling process is replaced with its topic label. In the cases where terms are associated with multiple fields, a term can be replaced with a set of labels. A central research question is how to handle the case where labels from multiple distinct topics match a query producing ambiguity. As described below, we view query words and their labels as evidence for various topics being present in the query. We propose to use a Bayesian network [48] to combine the evidence and carry out the probabilistic reasoning necessary to reach a conclusion.

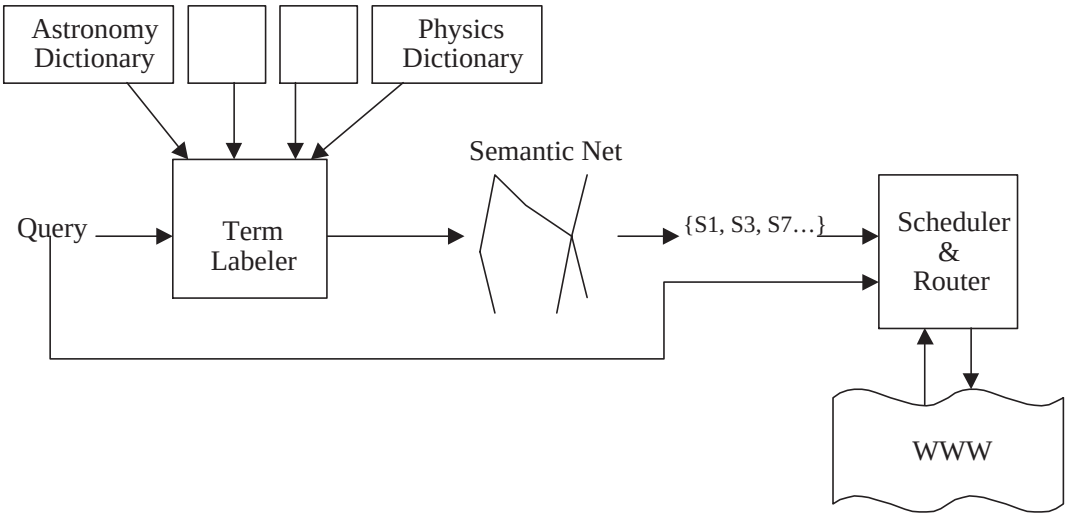

Figure 3: Intelligent Query Routing Architecture.

2.3.2 Topic computation

The labeled query is then used to index into a semantic network (e.g., WordNet [44]) that connects different words and terms to topics and subtopics. Each specialized search engine will be linked to the topics (or subtopics) that it is likely to be relevant to. We refer to topics (or subtopics) directly linked to a search engine as searchable. Thus, we have boiled down the problem of query routing to the problem of finding the searchable topics closest to the labeled query in the network. See Figure 4 for a fragment of a sample semantic network. To compute proximity in the network one option would be to rely on spreading activation techniques [9], which are both fast and parallelizable.

A novel approach, which we plan to investigate and compare with spreading activation, is viewing the semantic net as a Bayesian network. In this view, the links between nodes in the network are interpreted as dependence relations and each node is associated with a conditional probability table that quantifies the effect that its parents have on it (the parents of a node are all tose nodes have arrows pointing to it). In Figure 4, for example, if we encounter the query consisting of the key words "Shoemaker-Levy" and "Jupiter" then we would set the probabilities of the corresponding term nodes to 1, and the probabilities of other primitive query terms to 0. Now, identifying the most likely topic(s) for the query and the appropriate search engines to query becomes probabilistic inference in a Bayesian network.

This approach is based on a number of simplifying assumptions, whose efficacy we plan to test empirically. Most important, as the fragment in Figure 4 suggest we are making the assumption that the interpretation of the different query words is independent. Thus, whether Jupiter refers to a deity or to the planet does not depend on other words in the query, which is clearly a simplification. However, this assumption closely related to the standard simplification used in the highly successful naive Bayes classifier [45, Chapter 6].

Although probabilistic inference in Bayesian networks is NP-hard in the worst case [12], a number of methods have been developed that make inference feasible in practice. Specifically, we can rely on stochastic simulation of the network to approximate the necessary posterior probabilities [47].

Prior probabilities will be assigned to network links based on the following principles. First, the evidential weight of a term for a topic will be a function of the term's ambiguity. For example, in the network fragment shown in Figure 4, the evidential weight of term "Jupiter" (which is ambiguous between religion and astronomy) for the topic Astronomy is exactly one half the evidential weight of the unambiguous term "Shoemaker-Levy". Such prior probabilities can be assigned automatically based on the fan-out from each term in the network and the network topology. Second, the priors will be weak, enabling experience to rapidly update the probabilities and improve network performance. If necessary, we may elicit some additional priors from domain experts.

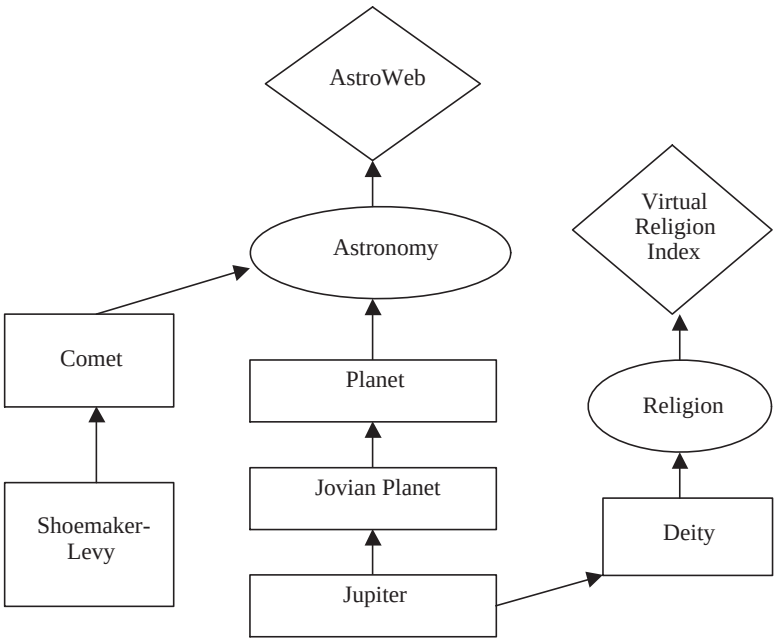

Figure 4: A Fragment of the Semantic Network that would enable us to route queries containing the keywords "Jupiter" and "Shoemaker-Levy" to Astronomy-specific resources such as AstroWeb. Rectangles represent terms, ovals represent searchable topics, and diamonds represent Web information resources. Arrows represent dependence relations.

2.3.3 Routing

Once the potentially relevant set of search engines has been identified, the next step is to decide which engines to query and when. We may choose to delay queries to *some* of the potentially relevant engines in order to reduce engine (and network) load, to avoid incurring a per-query charge, etc. In Section 2.4 we sketch our model for making query scheduling decisions. Finally, we come to the fairly straight forward process of translating the query to the appropriate format for each individual search engine about to be queried. This translation requires a representation of each engine that indicates what search features it supports (e.g., phrases). Often, specialized search engines support a more limited set of search features than say Alta Vista. However, the post-processing techniques developed in MetaCrawler can be used to compensate for missing features. For example, even though the AstroWeb search engine does not support phrase searching, the Librarian can download the pages it returns and prune away any pages that do not contain the desired phrases.

2.3.4 Update

Given the large number of engines that the Librarian may route to and the complexity of the envisioned semantic net, it is essential to continually update and improve the mapping with experience. First, we plan to record the references that Librarian users accessed for each topic (maintaining the full anonymity of the Librarian users, of course). This information enables us to track how often a resource that we predicted to be relevant to a particular topic was in fact relevant. Second, we plan to utilize (and extend!) methods from the burgeoning field of learning Bayes nets [31] to update the links and the probabilities in the network.

2.3.5 Discussion

A key issue that merits further discussion is network initialization. We recognize this is a massive undertaking. We plan to begin with the hypernym hierarchies encoded in the WordNet semantic network

(publicly available from Prof. Miller's group at Princeton University) and extend it with nodes corresponding to high-quality search engines and searchable topics (see Figure 4). By traversing the hypernym hierarchies it is possible to quickly move from query words to topics. For example, Jupiter is a kind of celestial object, which suggests the topic astronomy, etc. Finally, to keep the project manageable, we plan to focus initially on a subset of scientific topics (and queries) to demonstrate the feasibility of the Reference Librarian approach.

The algorithms we are investigating for both term labeling and topic computation are memory and compute-cycle intensive. Thus, we plan to create a client-server architecture in which the query router runs on a powerful server machine. The server would also maintain a library of all the current wrappers. This distributed architecture has several advantages:

- Fast operation of the query router requires a very large database; on a large centralized server such a database can totally reside in the computer's main memory.
- Without the query router the Librarian code has a small footprint and can be easily and quickly downloaded to client machines.
- The contents of the query router database must be updated regularly, so centralized control ensures high quality and up-to-date information.
- Wrappers are being created and updated regularly; this arrangement allows them to be centrally tracked and updated.

Once query routing has identified a relevant set of resources, the next question is what is the appropriate sequence for querying the resources, which we refer to as query scheduling.

2.4 Query Scheduling

Currently, much of the information on the Web is available free of charge, and as a result parallel search tools such as MetaCrawler can respond to requests by querying numerous information sources simultaneously to maximize the information provided and minimize delay. However, information providers are likely to start charging for their services in the near future; indeed, the New York Times already charges to retrieve articles from its archives. Billing protocols to support an "information marketplace" have been announced by large players such as Visa and Microsoft and by several research groups (e.g., IBM).

Once billing mechanisms are in place, Internet users will have to balance speedy access to information against the cost of obtaining that information. Clearly, the speediest information gathering plan would be to query every potential information source simultaneously, but that plan may well be prohibitively expensive. The most frugal alternative—querying the information sources sequentially—may prove to be prohibitively slow. This observation suggests the following information access problem: given a collection of N information sources, each of which has a known time delay, dollar cost and probability of providing the needed information, find an optimal schedule for querying the information sources. The success probability is computed by the Bayesian network described above and the dollar cost and expected time delay are provided as part of each information resource's model.

In [19] we developed several optimization models for the information access problem that vary according to the objective function. In all cases there are N information sources. Each information source is described by three numbers: its execution time, its dollar cost, and its success probability. A source is said to succeed if it provides the answer to the query. The event that a given source succeeds is assumed to be independent of the success or failure of the other sources. We considered situations where information is needed before a certain deadline, where information has to be found with a certain budget, and where there is a linear tradeoff between execution time and dollar cost. We proved that several plausible variants of the information access problem are NP-hard, but we were also able to come up with several algorithms that generate schedules that are provably close to optimal (see [19]) for the details.

Our earlier work on query scheduling suffers from a number of limitations that we propose to address in this project. First, the work presupposes that the different information sources are mutually independent. In practice, strong dependencies are possible. For example, we may find that if resource A returns empty-handed, then resource B is very likely to fail too. We propose to empirically explore the

dependencies between information resources on the Web and extend our information access algorithms to leverage these dependencies for improved performance. Second, our earlier work presupposes that accurate information about each resource is available. We plan to study automatic method for obtaining and updating cost, execution time, and probability of success models for each resource. Finally, we need to investigate the different methods of charging for information on the Web and ensure that the Librarian has an appropriate response to each.

2.5 Milestones

This proposal has twin objectives. First, we propose to deliver and deploy an Automatic Reference Librarian on the Web as we have done with previous research projects (see Section 1.1). Second, we plan to report on the lessons learned regarding foundational research issues, and specifically on the algorithms and representations discovered and analyzed in the premier conferences and journals. Below, we enumerate three key milestones for the project.

1. **Year one:** A massive database of wrappers for specialized Web information resources automatically compiled and a subset manually categorized; preliminary query routing semantic network built based on WordNet augmented with topic hierarchy and wrapper database.

2. **Year Two:** Intelligent query routing based on probabilistic reasoning over the semantic net is operational, debugged, and evaluated; algorithms for automatically updating and maintaining the wrapper database are implemented; automatic source categorization complete and under evaluation.

3. **Year Three:** Librarian deployed on the Web and in popular use; query routing semantic net automatically updated to reflect usage patterns and to track changes in the quality of information sources; user studies of the Librarian completed.

2.6 Related Work

Work related to our project appears in a number of distinct areas and communities including work on heterogeneous database integration, work on mediators and autonomous software agents, and AI work on machine learning and planning. We briefly consider each area below.

By and large, work on "virtual libraries" is *not* closely related to our project because it focuses on putting together large but static collections of information to merit the label *library* instead of creating an agent or a *librarian* as we are trying to do. For example, The Stanford Virtual Library (`vlib.stanford.edu/Overview.html`) is merely a directory of links to Web pages "run by a loose confederation of volunteers". While useful, it suffers from the scaling problems associated with all manually compiled directories. Mitchell et. al's [14] WWKB project attempts to turn the Web directly into a symbolic knowledge base that is computer-understandable, which would allow for sophisticated automatic reasoning and information retrieval. This is an ambitious, visionary project. However, its focus in the foreseeable future is on the development of concept learning techniques that attempt to extract symbolic information from collections of pages and links in service of its long-term goal.

Our proposal is related to research on heterogeneous database integration, e.g. TSIMMIS [8], Garlic [7], Information Manifold [40], SIMS [3], and our previous Occam [36] and Razor [25] projects. However, these systems focussed on a smaller number (e.g. dozens to a hundred) of well-structured information sources (e.g. those whose information could be conceptualized in relational or object-oriented schemata). In contrast, our Librarian will index thousands of less-structured resources, e.g. those returning English-text responses to queries rather than just tabular data. WHIRL [10] is similar to the heterogeneous database systems in the sense that it requires a relational schema for each information source, but like our librarian WHIRL uses information retrieval similarity metrics to connect different sources.

Previous work has exhibited a variety of approaches towards the problem of automatic query routing. Some, like Search Broker or SavvySearch, place that burden on the user of the system (or ignore query routing altogether as in commercial systems such as MetaCrawler, Jango, and Junglee). Garlic [7] and

TSIMMIS [8] use hard-coded query routing. For every predicate that could appear in a query, they have a fixed procedure for extracting tuples in that relation. Occam [36] and the Information Manifold [40] demonstrate powerful query routing capability by generating multi-step plans that involves invoking and composing the appropriate information sources [17]. Again, this approach relies on being able to represent each information source in terms of relational schemata, which is impossible when the source is a directory or a document index and the query consists of a phrase or a set of keywords. Yet in such cases our probabilistic approach is appropriate.

Work in the field of autonomous software agents has, by and large, focused on a very different set of research issues than the ones raised by the Automatic Librarian we propose. Much work has gone into agents that automatically filter information streams (e.g., e-mail or newsgroup postings) by learning their user's taste [42, 43, 41]. Other groups have focused on multi-agent coordination [15], particularly under economic models [59]. Much more closely related is the fielded agent known as the Search Broker (`sb.cs.arizona.edu/sb`), which grew out of the Harvest project [5]. Search Broker relies on hand-written wrappers for some 400 information sources for topics ranging from animals to travel. Search Broker associates a topic with each information source. However, instead of automatic query routing, Search Broker asks its user to specify *the* topic of each query, the Broker then queries a single relevant source. While Search Broker provides some validation of our basic approach, it is clear that it would be enhanced by the automatic modeling of information source and automatic query routing we propose. Finally, Search Broker has no notion of query scheduling.

Our approach towards automatic modeling of information sources (section 2.2) is based on the ideas we reported in [53, 16, 35]. Some of these can be viewed as a special form of grammar induction suggesting techniques from [2, 54]. Recently, other researchers have investigated wrapper learning. ARIADNE [4] is a semi-automatic system for constructing wrappers from hierarchical web sites; it benefits from domain- and HTML-specific heuristics which we may adopt. A related idea, explored in [11] is the notion of extracting information from a HTML parse tree of the source page, rather than the page itself; we expect to adopt this technique if our evaluation is favorable. Natural language processing has been used for similar information-extraction tasks (see [13] for a recent summary), but we don't foresee requiring the power that these systems evoke; nor do we wish to pay the performance cost that NL-based techniques demand.

In summary, our proposed project differs from previous and related work in three ways. First, we propose an unprecedented degree of automation from the fully automatic creation of wrappers to automatic routing of queries. Second, automation enables us to build a Librarian that is one to two orders of magnitude more comprehensive than existing systems such as Jango or Search Broker. Finally, we are committed to both investigating the fundamental research issues *and* deploying a working prototype on the Web (see Section 1 for our track record on both counts).

2.7 Summary

We propose to take the modular MetaCrawler softbot architecture and extend it to support the creation of an Automatic Reference Librarian for the World Wide Web. In the limited space of this proposal, we have sketched the overall architecture of the Librarian, referring to our technical papers to flesh out many of the details omitted for brevity. The most important issues for future work are 1) scale up — increasing the number of adaptors the Librarian can access by two orders of magnitude from fewer than 10 in the MetaCrawler to more than 1,000 within three years. 2) query routing — the novel combination of Bayes nets and semantic nets to perform evidential reasoning from query terms to topics to specialized search engines. 3) automation — the use of machine learning techniques to automatically generate (and maintain) several thousand information-resource models and the corresponding semantic net.

We plan to make the Librarian source code freely available to other research groups and to make its wrapper database and query routing semantic net highly extensible and customizable so that group wanting to extend or tune the Librarian for specific topic areas will be able to do so with minimal overhead. Over time, we see the potential for a network of more specialized Reference Librarians emerging where each Librarian has the option of routing a query to its colleagues.