Web Hunting: Design of a Simple Intelligent Web Search Agent

by G. Michael Youngblood

Introduction

The World Wide Web has become a vast resource of information. The problem is that finding the information that an individual desires is often quite difficult, because of the complexity in organization and the quantity of information stored. Information is stored in uniquely named files located within uniquely named directories on a Website identified by a series of four numbers or a corresponding symbolic name. The number of directories and files is limited only by storage capacity and content availability. Finding the right page, the one containing the information that is sought in one of those uniquely named files, via its Uniform Resource Locator (URL; protocol + host + file -- e.g., http://acm.uta.edu/youngblood/index.html) would be nearly impossible if an individual had to guess where the information was located. Fortunately, search engines such as Alta Vista (www.altavista.com), Yahoo (www.yahoo.com), HotBot (www.hotbot.com), and many others have made navigating the Web easier. These services provide users access to large databases containing keyword cross references to URLs. The heart of a search engine is the search agent, which collects the information to be placed in the database.

The goal of this article is to introduce the reader to the basic elements of an intelligent agent, and then apply those elements to a Web search agent to provide the framework for the construction of a simple intelligent Web search agent. An overview of typical artificial intelligence search algorithms will be presented and performance metrics will be discussed. This article presents a collection of ideas and pointers to resources that will hopefully provide some insight and basis for further inquiry into the subject matter. Many of the ideas presented are quite intuitive and there are numerous sources available on the Internet that present complete search agent systems, design considerations, and other valuable resources [11,12,14]. The cited sources in this article are by no means comprehensive -- readers are encouraged to follow the Internet links to discover more information.

Elements of an Intelligent Agent

What is an agent? ``An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."[17, p. 31] In the case of the Web search agent, the environment is the World Wide Web for searching and the computer terminal for interaction with the user. The agent's percepts are the words of an HTML (Hypertext Markup Language) document acquired from using software sensors that connect through the worldwide network (i.e., Internet) utilizing HTTP (Hypertext Transfer Protocol). The agent's actions are to determine if its goal of seeking a Website

containing a specified target (e.g., key word or phrase), has been met and if not, find other locations to visit in which to further the search to attain the goal. It acts on the environment using output methods to update the user on the status of the search or the end result(s), which is hopefully the attainment of the goal.

What makes the agent intelligent is a matter of defining intelligence, which has been a large debate in the field of artificial intelligence. For the sake of avoiding the argument, let us say that an intelligent Web search agent is one that makes a rational decision when given a choice. In other words, given a goal, it will make decisions to follow the course of actions that would lead it to that goal in a timely manner. After all, the ``intelligent'' thing to do when confronted with a problem is to work towards a solution and not away from it.

Search Algorithms

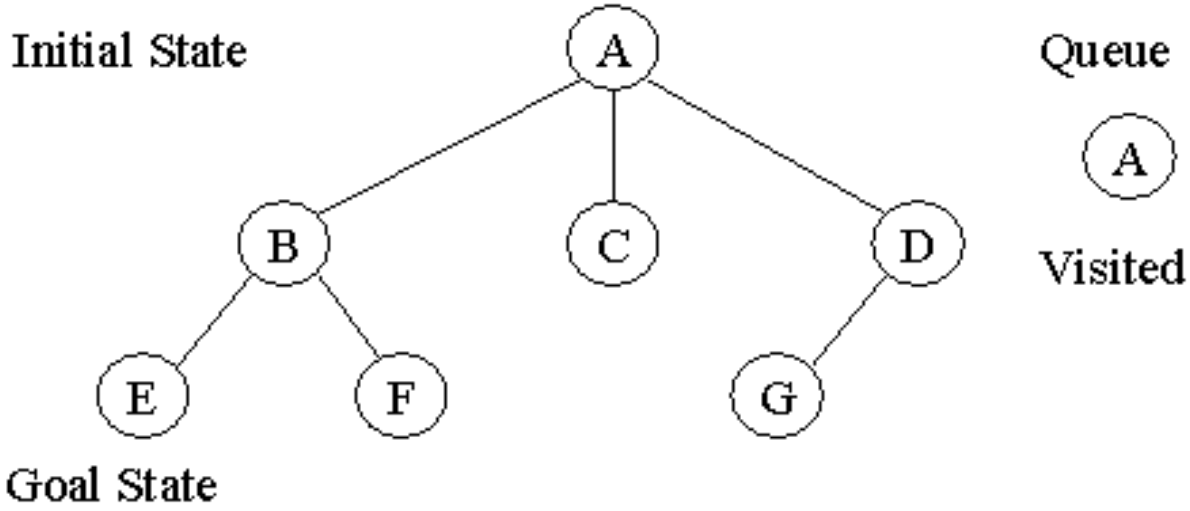
In the study of artificial intelligence, problems can often be reduced to a search. An agent can usually generate all of the possible outcomes of an event, but then it will need to search through those outcomes to find the desired goal and execute the path (sequence of steps) starting at the initial, or current state, to get to the desired goal state. In the case of the intelligent Web search agent it will need to utilize a search to navigate through the World Wide Web in an effort to reach its goal. There are many search algorithms and many journal articles and books written about them. This article will present an overview of a few search algorithms.

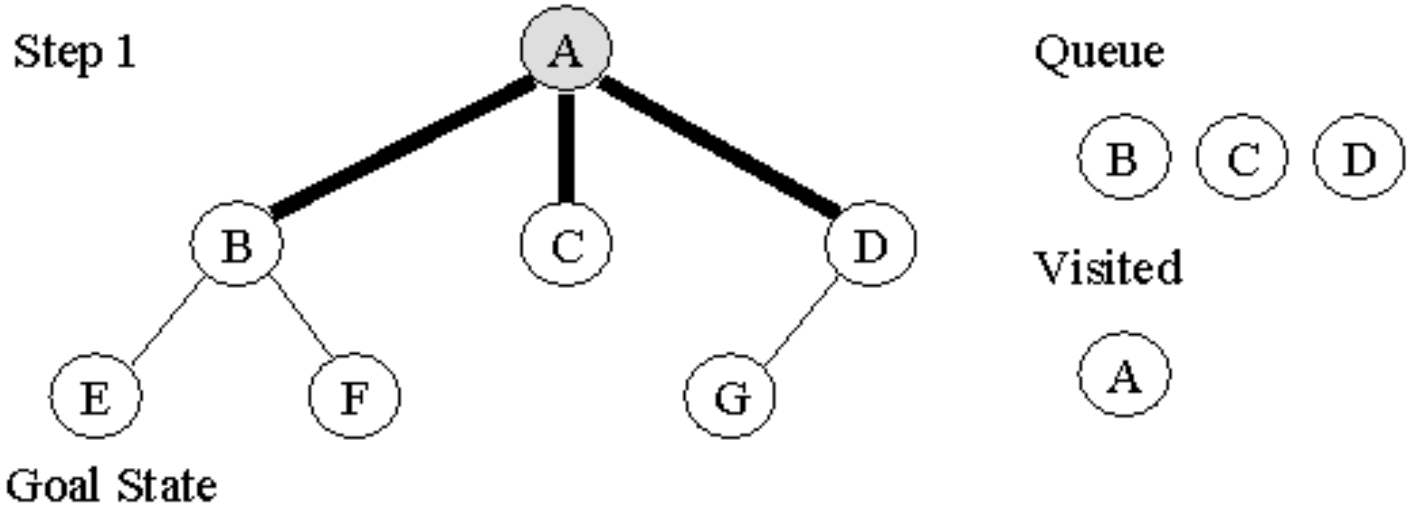
There are two basic classes of search algorithms: uninformed and informed. **Uninformed**, or blind, searches are those that have ``no information about the number of steps or the path cost from the current state to the goal.''[17, p.73] These searches include: depth-first, breadth-first, uniform-cost, depth-limiting, and iterative deepening search. **Informed**, or heuristic, searches are those that have information about the goal; this information is usually either an estimated path cost to it or estimated number of steps away from it. This information is known as the heuristic. It allows informed searches to perform better than the blind searches and makes them behave in an almost ``rational'' manner. These searches include: best-first, hill-climbing, beam, A*, and IDA* (iterative deepening A*) searches [17].

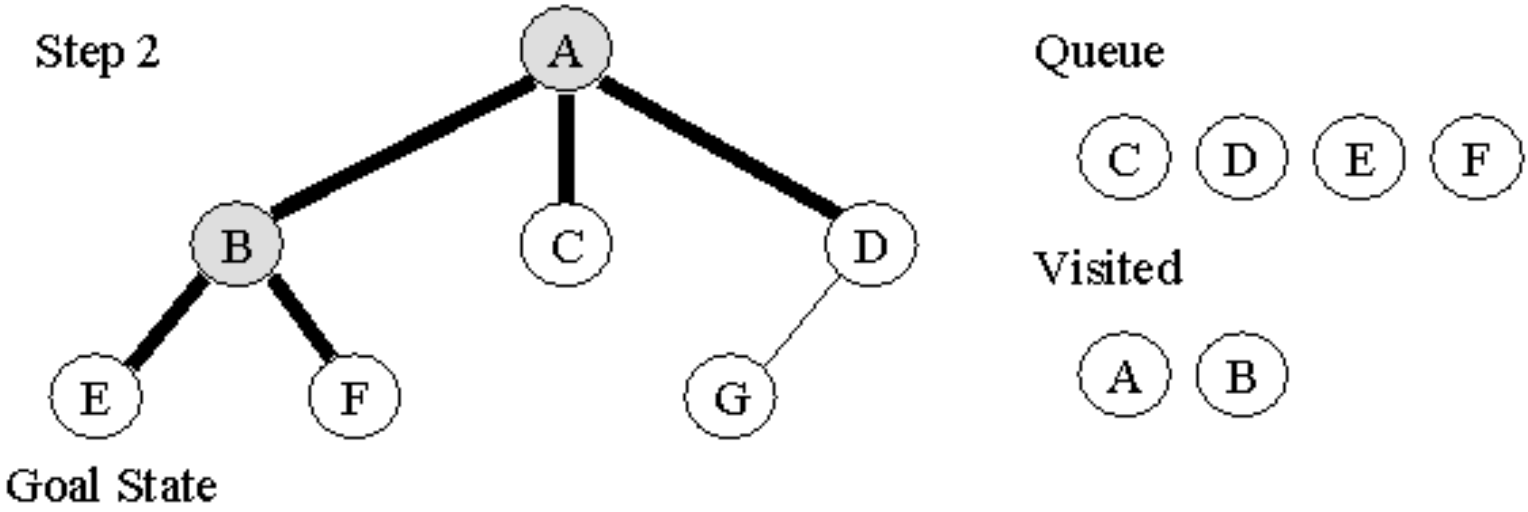
To provide some example search algorithms, the strategies of four searches will be discussed: breadth-first, uniform-cost, best-first, and A*. There are many others and readers are encouraged to find more information on advanced searches. A good search overview is available at the UGAI Website [8].

Breadth-first search is one of the two most common search algorithms every computer science student learns (the other being depth-first search). The approach of breadth-first search is to start with a queue containing a list of nodes to visit. A node is a state or a value; in programming it is usually represented as a structure containing particular information about the environment or domain of the problem. The algorithm starts by placing the initial state of the problem into the head (beginning) of the queue. The search then proceeds by visiting the first node and adding all nodes connected to that node to the queue. When viewed as a tree graph, it would move from left to right from the current node (usually represented

as a circle on a graph) along links (represented as connecting lines inbetween the node circles) to connected nodes, adding the connected nodes to the queue. As the head node of the queue is visited it is removed. The search then moves to the next node on the queue and continues until the goal is reached. This search will always find a solution if it exists [17]. Figure 1 provides a graphical example.







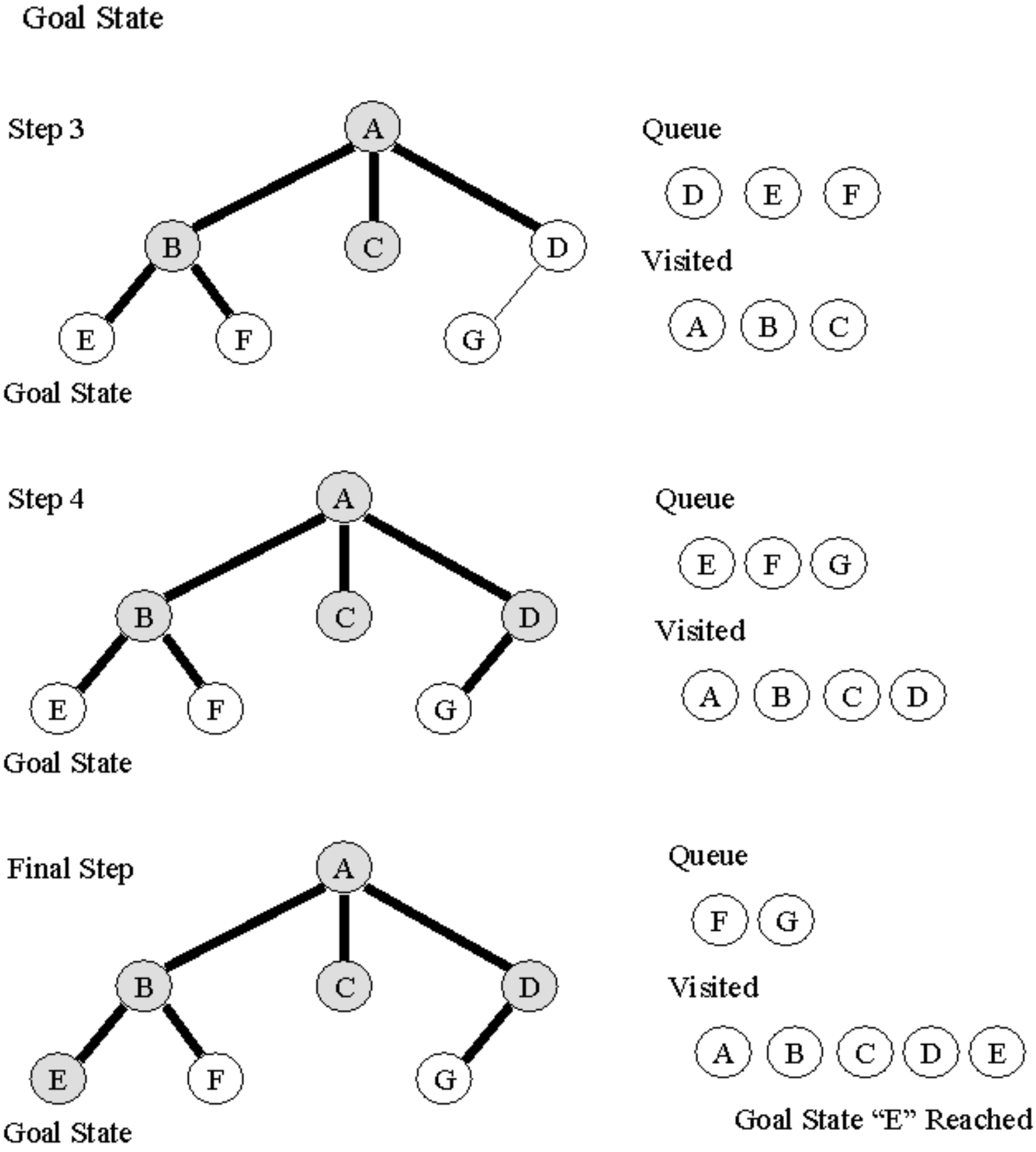


Figure 1 Breadth-first Search Example

Uniform-cost search is similar to breadth-first search, but after it adds a set of nodes to the queue it sorts

all of the nodes on the queue by cost (distance value from root node to current node) before taking the next head node off of the queue. It will always find an optimal solution if one exists [17].

For heuristic (``informed'') searches, best-first search is similar to uniform-cost search except that it uses a heuristic by which to sort. It is usually faster, but is not guaranteed to find a solution. An A* search is an improvement on best-first search. It works in the same manner as best-first search (and uniform-cost search), but instead of sorting based on just a heuristic it adds the cost and the heuristic together and sorts on their sum. Given that the heuristic is never an overestimate, it will always find an optimal solution [17]. By virtue of being an informed search it should always yield better results than the blind searches.

Searching is a valuable tool in many applications. It will be a key component in building an intelligent Web search agent.

Building a Web Hunter

The concept of what is needed to build an intelligent Web search agent should be clarified. From this point on the agent will be referred to as the Web Hunter. More advanced versions of this type of agent are known as robots, spiders, or web crawlers [13]. Since the framework will be simple at this point, and not a full robot, we will call it a Web Hunter. This is a good description of what the desired agent should do. A complete Web robot should also include mechanisms for multiple and combinational keyword searches, exclusion handling, and the ability to self-seed when it exhausts a search space. These issues are discussed but are not included in the framework. Given a target, the Web Hunter should proceed to look for it taking as many paths as are necessary. This agent will be keyword based. The method advocated is to start from a ``seed'' location (user provided) and find all other locations linked in a tree fashion to the root (seed location) that contain the target. Blind URL searches based on random IP (Internet Protocol; i.e., 129.107.2.249) numbers would be a long and tedious search, but could be an exhaustive method for a complete Web hunt. There are a few parameters that should be defined for the Web Hunter. It needs to know the target (i.e., key word or phrase), where to start, how many iterations of the target to find (the more URLs the better), how long to look (time constraint), and by what method should it choose paths (search methods). These issues will need to be addressed by the software.

Implementation will require some knowledge of general programming, working with sockets, the Hypertext Transfer Protocol (HTTP), Hypertext Markup Language (HTML), sorting, and searches. There are many languages with Web based utilities, advanced APIs (application programming interfaces), and superior text parsing capabilities that can be used to write a Web Hunter. Perl is a good language choice that can be used with the libwww-perl5 library available at the Comprehensive Perl Archive Network (www.cpan.org) to build the Web Hunter [13]. Java (java.sun.com) and Python (www.python.org) are also excellent languages for writing Web based agents [20]. There exist some agent shells for writing agents as well. Information on agent shells can be viewed at the AgentBuilder Website (www.agentbuilder.com/AgentTools). This article does not explore agent shells, but will instead focus on design and implementation items for creating the Web Hunter. It is anticipated that many readers will start in ANSI C in a UNIX environment since most beginning computer science students are familiar with C. For

C programmers, a good location to find useful code for using sockets is the *Programming UNIX Sockets in C - FAQ* [15]. The best source of information about Internet protocols on the World Wide Web is the World Wide Web Consortium Website (www.w3.org), which contains specifications for HTTP [4]. Also helpful are the Request For Comments (RFCs); both are usually created by the working groups of the Internet Engineering Task Force (IETF - www.ietf.org) [1,4,5]. RFC 1945 covers HTTP 1.0, currently in wide use, and RFC 2068 covers HTTP 1.1, which is growing in popularity[1,5]. For information on HTML there is a plethora of books on the subject, but most of the information pertinent to the simple Web Hunter is discussed in this article. The use of a sorting routine will be necessary for many of the search techniques. Most basic algorithm or programming books will contain sorting routines [2]. Using a more advanced, efficient sorting algorithm (e.g., shell or quick sort) will help improve the performance of the Web Hunter [6,18]. Simple search techniques can be found in introductory algorithm books and even in some programming books, but the best location for finding information about advanced search techniques is in an artificial intelligence text [2,17].

The example Web Hunter design consists of four main phases: initialization, perception, action, and effect. In the initialization phase the Web Hunter should set up all variables, structures, and arrays. It should also get the base information it will need to conduct the ``hunt'' -- the target, the goal (the number of Websites containing the target, found in a timely manner), a place to start (a good hunter starts at a good entry location into the forest), and the method of searching, if more than one is available. The perception phase is centered on using the knowledge provided to contact a site and retrieve the information from that location. It should identify if the target is present and should identify paths to other URL locations (i.e., hyperlinks within the page). The action phase takes all of the information that the system knows and determines if the goal has been met (the target has been found and the hunt is over). If the hunt is still active it must make the decision on where to go next. This is the intelligence of the agent, and the method of search dictates how ``smart'' the Web Hunter will be. The effect phase is to list the location(s) of the target and to provide status and feedback to the user. Implemented together and connected to the Internet the phases form the Web Hunter.

During initialization, a data structure to contain information on locations visited should be created. Either a linked list or array will work. The nodes of the structure should contain information about the URL, the directory and specific page location, a distance value (used in cost searches), and one or more heuristic value(s) (used in informed searches). Using HTTP to connect to web sites, we need to specify the location of the web page to retrieve (e.g., /youngblood/crossroads/webhunter.html). This is the importance to having URL and page location fields within a node. The method(s) of search utilized will necessitate the use of additional variables within a node. An initial node should be created containing a ``seed'' value provided to the Web Hunter. The seed value could be a random IP address in dotted IP notation (e.g., 12.34.56.78), but then it is unknown where the search is starting and the likelihood of a match is low. I recommend starting with a URL of a search engine page (i.e., HotBot and others mentioned previously) or a known URL that contains links on the subject. Once that tree is exhausted the search would be over, unless another seed value was inserted into the queue. Next, the goal should be clearly defined for each hunt. A target has to be established. Again, for simplicity we start with keyword searches. Since the user will want to look at more than just one valid page, the number of different pages (Websites) that contain the target should be specified as the goal. Since the user wants information quickly the goal should also be

to achieve the goal number of Websites in the shortest amount of time. If the Web Hunter is flexible enough to allow different approaches (searches) the method to use should be established. Given the characteristics of searches these could be classified as behaviors of the Web Hunter. The software could be set up to prompt the user to invoke an aggressive, moderate, or casual hunter. Combination and permutations of searches could also be employed. The Web Hunter could start out casual and eventually transition to become aggressive depending on the results of the hunt. After initialization and approach method establishment, the hunt can begin!

First, the Web Hunter should enter the perception phase by opening a socket on the HTTP port (80) to the specified URL site. Once a connection is made, the HTTP command ``GET'' used with the directory information (use ``/'' for the default site page) will invoke the server to issue the requested page [1]. The received page should be parsed for hyperlinks to be added in nodes to the queue and to determine if the target is present. The key HTML phrases to look for hyperlinks or other URL locations are ``HREF='' and ``SRC=''. The link will be contained in quotes after the key. The link should contain the string ``http'' somewhere in the text. If other types of data search are desired, the Web Hunter could also parse for other key file elements. In this manner it could be made to search for graphics by looking for ``JPG'' or ``GIF'' references, or any easily identifiable file type. Care should be taken to not request types of information that the Web Hunter implementation would be incapable of handling (i.e., do not ask for GIFs if you cannot properly receive them) [11]. A node should be built based on the link and added to the queue. Websites can contain a variable number of hyperlinks and referenced HTML pages ranging from none to hundreds or possibly thousands. It may be helpful to set an upper bound on the number of lines it will receive. Text parsing can be an involved programming task, especially in C; familiarity with the string library is a must if using straight ANSI C, but obtaining an advanced parsing library or utilizing the features of a better text parsing language could reduce the task [6]. In order to avoid some pitfalls, remember that the World Wide Web directory location and page are case sensitive. Once the page is received and classified, the next step is the action phase.

Since the Web Hunter has now captured a page and extracted the important information, it is time for some ``thought'' and some action. If the goal has been reached (comparing the number of Websites currently found to the number of goal Websites) then the hunt is over and it was successful. If the goal has not been met, a decision of what step to take next has to be made. The heart of the decision is the choice of which search algorithm to utilize. The nodes captured from the current document will be added to the queue. The method of search dictates the manner and order in which they are entered, as was previously discussed. The decision of which node to visit next is made by the search algorithm. The more informed the search method, the better the chance of reaching the goal.

Blind searches will not usually care about additional information, but may require cost information. Cost for Websites can be defined by their distance from the Web Hunter. The ``traceroute'' and ``ping'' UNIX system commands can be used as methods to measure distance (cost) on a network [7]. Ping, which measures the round-trip-time for a packet sent on the network, is recommended for speed. The distance, or access speed, could be used as a heuristic since each node can be viewed as part of the goal and part of the goal is to minimize the overall time of the search. As a different heuristic for A* search, a weighting factor based on country could be used. Since almost all Websites in the United States are in the ``COM'',

``NET", ``EDU", ``ORG", ``GOV", ``MIL", or ``US" top-level domain, a U.S.-based web hunter may assign Websites that are not in these domains a higher cost value as a penalty for being out of country and possibly in a different language. As a side note, ping failures (e.g., www.microsoft.com does not return pings) should be assigned an estimated cost value which should be higher than the out of country penalty since the site may not be operational. This is admittedly a fairly weak heuristic since many companies and educational institutions outside of the US have ``COM" and ``EDU" addresses. Due to the nature of how IP addresses and naming conventions have been handled, there is not an accurate way to determine locations of sites on the Web. There is some reprieve from this chaos thanks to Classless InterDomain Routing (CIDR), the current remaining Class C network IP numbers can be traced to one of four zones on Earth [19]. In light of the current number of IP addresses already taken, this is only minimally helpful. For information on CIDR see RFC 1519 [19]. IP allocations can be seen at the ``IP Network Index" Website maintained by John Crossley which contains links to more recent information [3]. There are many other possibilities for heuristics and cost -- implementers should experiment to make their Web Hunter better. It is important to remember that heuristics should not overestimate in order for many of the search techniques to work properly (ensure an optimal solution cost).

After the search method inserts and/or deletes nodes it may sort them based on cost and/or a heuristic. The choice of sorting algorithm is not that important, but one that performs rapidly and correctly will (obviously) improve the performance of the Web Hunter and reduce run time (part of the goal). If the search space has been exhausted (no more nodes in the queue) the Web Hunter can either get a new seed value and start from there or send status and terminate.

The effect phase will provide status to the user. If a match is found, the hunt is interrupted, or the hunt is complete it should provide output to the user. The effector from the action based on the perception in the environment is to provide feedback to the user.

The Web Hunter moves from the initialize phase to a loop consisting of the perception, action, and effect phases until the goal is achieved or cannot be achieved. Figure 2 illustrates the basic action sequence of the Web Hunter intelligent Web search agent.

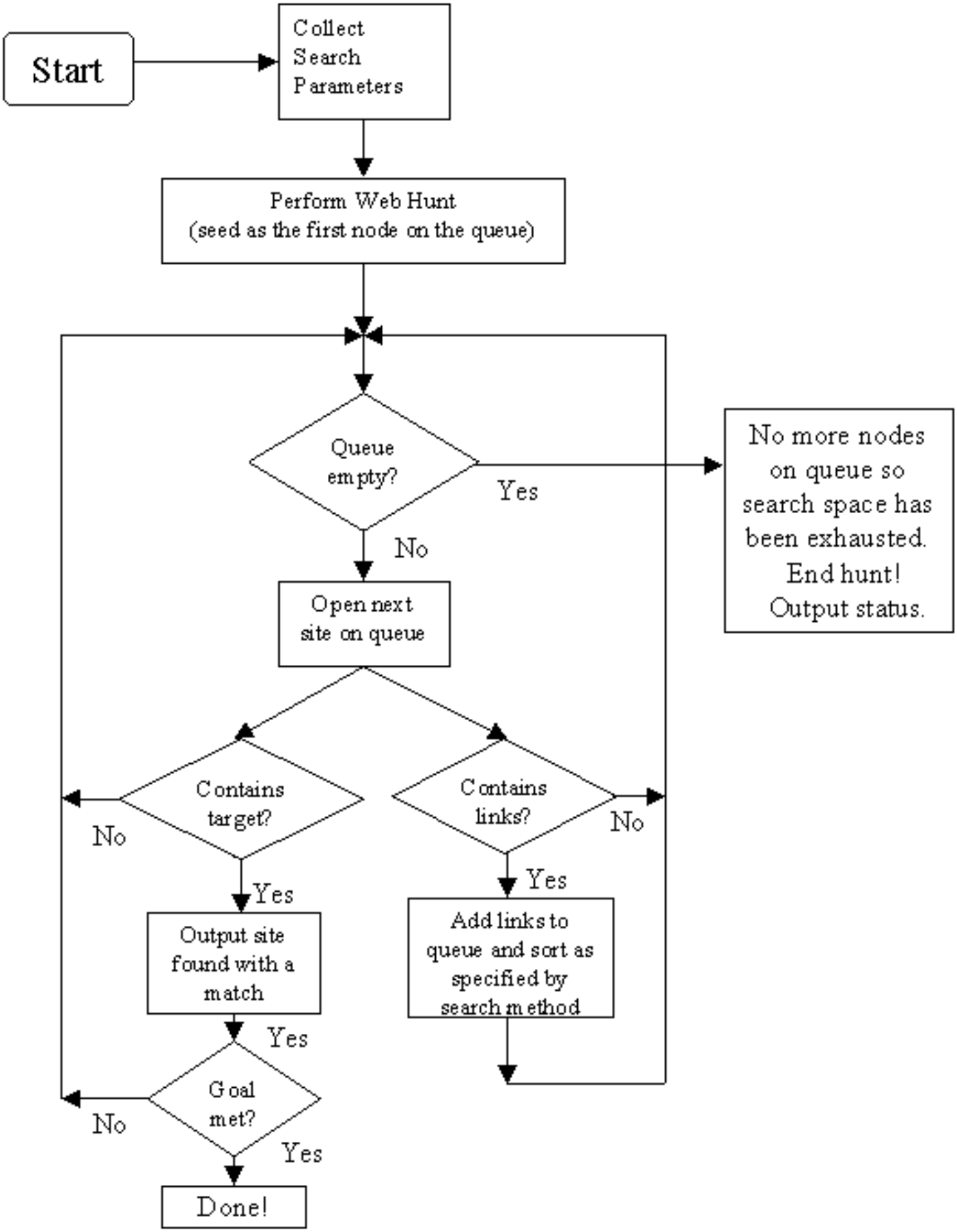

Start
Collect Search Parameters
Perform Web Hunt (seed as the first node on the queue)
Queue empty?
Yes
No more nodes on queue so search space has been exhausted. End hunt! Output status.
No
Open next site on queue
Contains target?
No
Yes
Output site found with a match
Yes
Goal met?
No
Yes
Done!
Contains links?
No
Yes
Add links to queue and sort as specified by search method

Figure 2 Web Hunter Basic Flow

Performance Metrics

An implemented Web Hunter will not perform like a commercial search engine. Search engines use database lookups from a stored database created by similar Web search agents (i.e., industrial spiders). Despite being a low-level utility, good performance will make the Web Hunter a more useful tool. Testing the speed of searches on a given target is a good metric. Wrap the perception-action-effect loop in a timing function, and test the speed of searches by utilizing different search techniques and sorting routines. Goal achievement is also important. Some seed values are better than others. Utilizing more seed Websites to execute sequentially after the previous has been exhausted may improve goal attainment. The program could even be modified to do seed lookups itself, utilizing search engines or random/sequential IP number generation. Use goal achievement as the metric for search ability. The use of time and goal achievement are examples of some simple metrics for performance; others, such as the validity of the hits may prove useful as well.

Web Hunter Applications

After creation of a Web Hunter, the agent could be used in many applications. An entire search engine system could be created using a dictionary word generator and a sequential IP number generator to perform an exhaustive search on a target. The results could be stored in a database for later retrieval. Provided with an HTML interface and its own Website, it could be another Web search engine. The Web Hunter could also be modified to be a Document Hunter or Record Hunter to go through files, instead of Websites, looking for targets. It could be used to hunt through any data in any environment it can access. The Web Hunter and its core, search, could be a useful tool for many applications.

Responsible Hunting

So, you followed the framework, did some research, and spent a night writing a Web Hunter. Now, you are ready to hunt the Web, right? Wrong! There are a few things which have not been mentioned that an implementer needs to know.

Speed was mentioned as being an important attribute, but ``rapid-fire'' searching puts a heavy burden on servers [11]. Limit how fast the Web Hunter makes repeated queries to the same server by introducing timed waits. Overall, speed can be maintained by visiting different servers while waiting to query a previously contacted server. Delays should definitely be longer than a second, as long a wait as possible without sacrificing too much time should be used. Along these lines, be sure to check newly obtained sites against those already visited; otherwise, you could cause a loop and not have a very productive hunt.

When testing the Web Hunter during implementation do it locally [11]. Do not unleash a Web Hunter out on the Web until it has some stability. When it is stable and ready to venture out into the Internet there are a few things the Web Hunter must do. Web Hunters and Hunter operators should identify themselves by

using the HTTP ``User-agent'' and ``From'' fields in the same manner that Web browsers do [4,11]. If the Web Hunter will be doing a lot of activity from a machine or on a specific server, the user should notify that system's System Administrator as a courtesy. If the Web Hunter will be a frequent hunter on the Web, the user should register the agent for inclusion in the ``Web Robots Database.''[12] Reading and following the advice given by Martijn Koster in ``Guidelines for Robot Writers'' is a must for any Web Hunter implementer [11].

Another item to be aware of is the presence of a /robots.txt file on a Website. Web Hunters should check for its existence. The robots.txt file contains information directed at Web agents. It establishes an acceptable use policy and/or explicit bans for Web agents [10]. HTML pages may include meta tags such as <META NAME=``ROBOTS'' CONTENT=``NOFOLLOW''> which specify to Web agents to not parse that page [13]. For more information on items that any good Web Hunter should adhere to refer to Martijn Koster's ``A Standard for Robot Exclusion.''[10]

The Web is a big place and fertile hunting grounds, but as in nature, take a conservationist approach. Use resources wisely, be courteous, and be safe.

Related Web Work

There are numerous intelligent Web robots constantly hunting, or crawling, around the World Wide Web. The Web Robots Database currently lists 185+ known robots [12]. The Database gives contact and link information for each robot registered. It is an excellent starting point to start investigating other Web robots.

Notably, Dr. Oren Etzioni of the University of Washington (www.cs.washington.edu/homes/etzioni) is currently performing research utilizing AI and informal retrieval. The Intelligent WebWare Project has produced a couple of intelligent Web agents and is exploring methods for making the Web easier to navigate.

Future Hunters

There are numerous possibilities for improvements in current Web agent technology. The complex nature of the dynamic World Wide Web makes productive planning and searching difficult tasks. Further research in advanced search methods and planning strategies could be employed in Web robots. There are also gaps in the human-computer interface. There is some disparity between the way people query for information and the way computers store information. Future Web Hunters may benefit from natural language processing research. If people could interact in a way that was more natural to them and the agent could understand their request better, then a better search could be performed. Searching can move from a syntactic basis to a conceptual or semantic basis allowing the agent to search for ideas and not just words. If the agent understands concepts then machine learning could be utilized to understand user behavior and could refine searches in a custom manner, tailored for the specific user based on their

profile. Improvements in planning and search as well as integration of natural language processing and machine learning could improve the intelligence of Web agents. The Natural Language Processing FAQ and the Artificial Intelligence FAQ are good sources to discover about research in these fields [9,16].

Conclusion

The use of software agents and search are becoming important tools in the information age and especially in information environments such as the World Wide Web. Creating a Web search agent may seem like a daunting task, but most of the difficulty in beginning a project is not knowing where to start. RFCs, FAQs, standards, and other resources that are available are important tools that all programmers should be familiar with. The creation of an intelligent Web search agent is not a difficult task and the framework provided is a good basic basis for creating a Web Hunter. But before the hunt begins, programmers should ensure they adhere to netiquette and hunt safely.

References

1

Berners-Lee, T., R. Fielding, and H. Frystyk. ``Hypertext Transfer Protocol -- HTTP/1.0." *RFC:1945*. http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1945.txt. May 1996.

2

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* McGraw-Hill, New York, N.Y., 1997.

3

Crossley, John. ``IP Network Index." http://ipindex.dragonstar.net. 22 Dec. 1997.

4

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. ``Hypertext Transfer Protocol -- HTTP/1.1." *INTERNET-DRAFT*. http://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-rev-06.txt. 18 Nov. 1998.

5

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. ``Hypertext Transfer Protocol -- HTTP/1.1." *RFC: 2068*. http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2068.txt. Jan. 1997.

6

Foster, L.S. *C by Discovery*. Scott/Jones, El Granada, Calif., 1994.

7

Hekman, Jessica Perry. *LINUX in a Nutshell.* O'Reilly, Sebastopol, Calif., 1997.

8

Ingargiola, Giorgio P. ``Search." http://yoda.cis.temple.edu:8080/UGAIWWW/lectures97/search. 1 Dec. 1998.

9

Kantrowitz, Mark. ``Artificial Intelligence FAQ." http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/ai_general/top.html. 10 Aug. 1997.

10

Koster, Martijn. ``A Standard for Robot Exclusion." http://info.webcrawler.com/mak/projects/robots/norobots.html. 30 Nov. 1998.

11

Koster, Martijn. ``Guidelines for Robot Writers." http://info.webcrawler.com/mak/projects/robots/guidelines.html. 30 Nov. 1998.

12

Koster, Martijn. ``The Web Robots Database." http://info.webcrawler.com/mak/projects/robots/active.html. 30 Nov. 1998.

13

Koster, Martijn. ``The Web Robots FAQ." http://info.webcrawler.com/mak/projects/robots/faq.html. 30 Nov. 1998.

14

Koster, Martijn. ``The Web Robots Pages." http://info.webcrawler.com/mak/projects/robots/robots.html. 30 Nov. 1998.

15

Metcalfe, Vic, Andrew Gierth, et al. ``Programming UNIX Sockets in C - Frequently Asked Questions." http://www.ibrado.com/sock-faq/html/unix-socket-faq.html. 21 May 98.

16

Radev, Dragomir R. ``Natural Language Processing FAQ." http://www.cs.columbia.edu/~acl/nlpfaq.txt. 23 Dec. 1996.

17

Russell, Stuart, and Peter Norvig. *Artificial Intelligence A Modern Approach.* Prentice Hall, Upper Saddle River, N.J., 1995.

18

Schildt, Herbert. *C: The Complete Reference.* McGraw-Hill, Berkeley, Calif., 1987.

19

Tanenbaum, Andrew S. *Computer Networks.* 3rd Ed. Prentice Hall, Upper Saddle River, N.J., 1996.

20

Watters, Aaron, Guido van Rossum, and James C. Ahlstrom. *Internet Programming with Python.* M&T Books, New York, N.Y., 1996.

G. Michael Youngblood is a senior in the CSE Department at the University of Texas at Arlington graduating in May 1999. He is the ACM chapter chairman at UTA. He holds an ASAST in Nuclear Engineering Technologies from Thomas Edison State College and served eight years in the United States Navy Submarine Force. He is currently working on research in exploratory map building robots. He would like to thank his wife, Julie, for her patience and understanding while he has pursued his passion for computing.