Search Algorithms

Jennifer Mosher

To search is “to look at or examine [...] carefully in order to find something concealed.” An algorithm is “a set of rules for solving a problem in a finite number of steps.” Thus, a search algorithm is to look at or examine a problem, with a set of rules for solving said problem, in a finite number of steps. So, what kind of problems could a search algorithm solve? For starters, a sudoku-lover could solve a puzzle in minutes with a search algorithm. They are also used for finding information in large databases such as the fingerprint and image databases we sometimes see in crime shows when police are matching the fingerprints of perpetrators. They can be used for solving things like the popular traveling salesman problem. The myriads of applications for these search algorithms are still expanding, as anyone can see by looking for information on search algorithms.

The instances above are basic examples of the applications of search algorithms, but there are four main cases where search algorithms can be used: explicitly stored databases, virtual search spaces, sub-structures of a given structure, and finally the quantum computers of the future.

Explicitly stored databases include bank account information, electronic documents, and more. The search algorithms used to search in these are the linear search, binary search trees, heaps, hash tables and others. Virtual search spaces use search algorithms such as brute-force searches, and heuristics (heuristics is a term used for trying to find the shortest path). There are also local searches, which include meta-heuristics like simulated annealing and the tabu search. The sub-structures of a given structure, has the combinatorial search method again with lesser-known types of algorithms as well. Finally, there is the Grover’s algorithm for quantum computers. Some of each of these categories and their applications will be discussed.

Categories and Subject Descriptors: D.2.7 [Software Engineering]: Distribution and Maintenance—documentation; H.4.0 [Information Systems Applications]: General; I.7.2 [Text Processing]: Document Preparation—languages; photocomposition

General Terms: Documentation, Languages

Additional Key Words and Phrases: Document preparation, publications, typesetting

A Brief Outline of a Few Search Algorithms

Jasmine, an adorable five year-old, has a problem. She has vanilla and chocolate ice cream in her freezer. Jasmine’s mother told her she can only have two scoops of ice cream. Alas, there is only enough of each flavor to have once scoop of chocolate and one scoop of vanilla. Her first scoop

is vanilla. So what should Jasmine’s second scoop of ice cream be? If you said she should have chocolate ice cream as her second scoop, you are right. You have also just implemented a search algorithm. We had a problem: picking a combination of two scoops of ice cream for Jasmine. We had constraints: she is only allowed two scoops of ice cream, and only one of each flavor. Then, based on our first move (the scoop of vanilla), we inferred that the second scoop must be chocolate ice cream. This is exactly how a search algorithm, called the combinatorial search works. Search algorithms are used in many of the calculations that effect our every day lives, like the example above. Without even realizing it, many people in the world know how to compute some of the simpler search algorithms, including a five year-old named Jasmine. They are more essential in our everyday lives than most would believe. However, search algorithms can be applied to many different problems beyond just deciding on ice cream scoops. Basically, there are three different categories that search algorithms are used for: virtual search spaces, explicitly stored databases, and quantum computers. Under each of these there are different kinds of search algorithms as well, such as: heuristic, metaheuristic, local searches, combinatorial searches.

The definition of a heuristic is a method “involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods” [Webster, 2010]. When exact methods fail, heuristics and metaheuristics are used to solve discrete optimization problems. These methods are approximate and normally provide good solutions to complex optimization problems in a relatively short amount of time [Heuristics, 2010]. However, because they are approximate methods, the solutions they provide are usually not the optimal solutions.

In 1986 Fred Glover (then the Director of Technology Development for Management Robotics, Inc.) proposed the Tabu Search. The tabu search falls under the meta-heuristic category and is a sub-category of local search. Most local search methods have setbacks when they approach the
local optima. In order to overcome the issues with local optima, Glover created this new approach [Gendreau, 2002]. The tabu search does the following, in simple terms: when it encounters a local optimum it “pursues local search [...] by allowing non-improving moves” [Gendreau, 2002]. The Tabu Search uses memories, called Tabu Lists, to record the recent history of the search. This prevents the search from going back to previously visited solutions. When a move occurs, the search declares moves that reverse the effects of recent moves tabu (from the word taboo, and for this case means to disallow) [Gendreau, 2002]. In other words, when the search declares tabu it says, “you can’t do that move, because it will undo another move you have done.” All of the tabus are stored in a short-term memory of the search, or the tabu list mentioned earlier. This tabu list usually only has “a fixed and fairly limited quantity of information recorded” at once so tabus generally do not last for too long into the search [Gendreau, 2002].

Sometimes tabus can be too powerful, by stopping steps that would not have caused cycling and even making the search unable to go on. This is why Glover also has aspiration criteria that allows you to disable a tabu. In theory, a tabu search would go on forever. This is why there are three most commonly used termination criteria. It is stopped: “after a fixed number of iterations, after some number of iterations without an improvement in the objective function value, or when an object reaches a pre-specified threshold value” [Gendreau, 2002]. Michel Gendreau gives a template for a simple tabu search:
Interestingly, the idea of Tabu Lists can be linked to Artificial Intelligence concepts [Gendreau, 2002]. The tabu search as a whole is also perceived as affiliated with the AI field and thus has gained prominence as a framework for solving difficult problems [Glover, 1990].

Another heuristic associated with the AI field is simulated annealing. “Simulated annealing is a programming method that attempts to simulate the physical process of annealing” where annealing is the process of heating and then cooling something repeatedly for softening and making material less brittle [Heaton, 2008]. Therefore simulated annealing is to take a solution, heat it, and then cool it to produce a more optimal solution [Heaton, 2008]. Simulated annealing is a very practical
algorithm to use when the number of objects becomes unmanageably large. Sadly, this technique is unlikely to find an optimum solution, only very good solutions [Carr, 2010]. Nonetheless, there are several common problems for which simulated annealing is useful. One very common use is for the traveling salesman problem, where you have a number of cities and must calculate the best route (shortest mileage) through these cities. Most algorithms can easily find the local minimum, but cannot get beyond that to the global minimum [Carr, 2010]. Simulated annealing uses two tricks to get past this. First, it allows for “bad trades” using the metropolis algorithm. Though it gains more mileage by picking these steps, it allows the solver to examine other paths that may lead to less mileage further on in the algorithm. The second trick is to “lower the temperature.” Lowering the temperature limits the number of “bad trades.” Eventually, after the temperature is lowered many times, the algorithm only accepts “good trades,” thus it is capable of getting closer to the global minimum.

There are other uses for simulated annealing, as well. For example, you can solve a sudoku puzzle with simulated annealing. Just take a look at the blog for X’ian [Smith, 2010].
“A combinatorial problem is modeled as a set of variables, representing the objects the problem deals with, and a set of constraints, representing the relationships among the objects” [Tuck, 2009]. This is called a Constraint Satisfaction Problem (CSP). Constraint programming applies constraints, thus giving the problem a solution procedure. During this procedure, the program tries to find variables that solve the constraints that is set upon the problem. Constraint Programming, which constraint propagation falls under, is used for common things like “scheduling
and planning, vehicle routing, configuration, networks (such as power or pipeline networks), and bioinformatics” [Tack, 2009]. As a matter of fact, “intervals are among the most general constraints propagated; for example, given $y - 2x$ and $1 < x < 2$, one concludes $2 < y < 4$” [Ward, 1989]. As you will see, this is considered a constraint propagation because $2 < y < 4$ is an inference based on two constraints, $y - 2x$ and $1 < x < 2$.

To be more specific, constraint propagation is when you use constraint programming with a propagation-based constraint solver [Tack, 2009]. Constraint propagation is a heuristic search algorithm. It is used for solving combinatorial, or optimization, problems. Put simply, constraint propagation is when a program “infers that certain values cannot be part of certain variable domains because they violate some constraint” where a propagator is the “entity that performs constraint propagation” [Tack, 2009]. Looking back at the interval example in the prior paragraph, it does in fact make an inference that $y$ cannot be less than or equal to 2, or greater than or equal to 4 because it would violate the constraints $y - 2x$ and $1 < x < 2$. From here, we can see that two things are needed in order to successfully solve a problem using constraint propagation: “first, a model that makes the structure of the problem explicit, stating it in terms of high-level constraints such as all-different; and second, a solver that provides efficient implementations of a sufficient number of these high-level constraints as propagators.” [Tack, 2009]

Combinatorial search is another search algorithm that can solve many of the problems that might be considered CSPs or, as mentioned before, Constraint Satisfaction Problems. A combinatorial search consists of finding combinations of a “discrete set of items” that satisfy certain constraints [Hogg, 1996]. A sub-class of the combinatorial search is combinatorial optimization where, instead of just finding combinations that satisfy certain constraints, it finds the maximum or minimum for some parameter [Lee, 2004]. A drawback of any combinatorial search is that the solution time
can grow exponentially with the problem size [Hogg, 1996]. Also, unlike constraint propagation, the tabu search, and simulated annealing, the combinatorial search is usually used for finding substructures of a given structure. The combinatorial search is also used differently than the algorithms for both explicitly stored databases and quantum computers listed below.

Explicitly stored databases are used by most major organizations today. They include bank account information, electronic documents, personnel data, manufacturing databases, and financial databases. If a business uses some book keeping program, they are considered explicitly stored databases.

One of the most commonly used algorithms for explicitly stored databases is the linear search. One of the ways to explicitly store information is by using an array. In terms of arrays, a linear search compares a key element sequentially with each element in the array [Liang, 2009]. It does not stop the process until the key matches an element in an array, or the array ends without a match. When the key matches an element, the linear search returns the index of the element that matches the key. Specifically for java programming, when the search does not find a match it returns \(-1\) because there is not a \(-1\) index in an array for java [Liang, 2009].

For example, say you have an array \{1, 2, 3, 4, 5\}. The elements for this array are 1, 2, 3, 4, and 5. The index of the element 1 is 0. The index of element 2 is 1. The index of element 3 is 2, and so on. Suppose we want to search for the number 3, the key (defined above) will then be equal to 3. Thus the linear search will compare 3 to 1, 3 to 2, and then 3 to 3. The search has found an element matching the key, so it will return the index of the element. In this case it will return 2.

Now suppose we want to search for the number 7, the key will now be equal to 7. The linear search will compare 7 to 1, 7 to 2, 7 to 3, 7 to 4, and 7 to 5. The search has gone through the array with no matches, therefore it will return a \(-1\).
According to Y. Daniel Liang, a linear search will, on average, “compare half of the elements in an array before finding the key, if it exists” [Liang, 2009]. Also, if an array is large the completion time of a linear search will also be large; therefore it is inefficient for larger arrays [Liang, 2009].

A binary search is considered one of the more efficient algorithms for explicitly stored databases, and is also very similar to the linear search. This method is used for values, and the values must be ordered correctly in an array. The same terminology is used in both binary and linear search. First the binary search compares the key with the middle element. If it is equal, the search ends and gives the index of the middle element. If the key is less than the middle element, the binary search continues its search only in the first half of the array. Essentially, the binary search becomes a linear search at this step only for the first half of the array. If the key is larger than the middle element, the binary search continues to search only for the second half of the array. Again this part is exactly like a linear search but only with the second half of the array. However, if a match is not found, the search does not return \(-1\). It returns \((insertion\text{point} + 1)\), the insertion point being the point where we would insert the key if we placed it in the array [Liang, 2009].

Going back to the example that was used for linear search, the array \(\{1, 2, 3, 4, 5\}\) is already in order so we can use the binary search method. Using 3 as the key again, we first compare 3 to the middle element 3. It is a match, so binary search would return the index 2. Next, supposed the key is now 7. First, the algorithm would compare 7 to the middle element 3. It is not a match, so the algorithm would check if 7 is greater than 3 or if 7 is less than 3. Because 7 is greater than 3, it would then compare 7 with only the second half of the array. This process is done exactly like the linear search. First 7 is compared to 4. They do not match, so then 7 is compared to 5. Again, the two values do not match. The search algorithm has reached the end of the array, so for this case the binary search would return \(-6\) because \((insertion\text{point} + 1) = -(5 + 1) = -6\).
For a worst-case scenario, binary search would work through half of the array, which is the average for a linear search [Liang, 2009]. This is the reason that for most cases, especially for larger arrays, the binary search is more efficient than linear. However, the downfall of the binary search is exactly what makes it more efficient than the linear search; the array must already be ordered.

The binary search tree is essentially the binary search modified to work for information other than values. Binary search trees are a popular means of retrieving information by its name, or tag, etc, because they are great for organizing large files. This is because “they are efficient with both random and sequential access of records, and for modification of a file” [Nievergelt, 1974]. Binary search trees are one of the more simple search algorithms. To find if a given name is in a tree (where the information is stored), you compare it to the name at the root [Knuth, 1971]. Then there are four situations that arise:

One, there is no root because the tree is empty. The given name is not in the tree, and the search terminates unsuccessfully. Two, the given name matches the name at the root: the search terminates successfully. Three, the given name is less than the name at the root: the search continues by examining the left subtree of the root in the same way. Or four, the given name is greater than the name at the root: the search continues by examining the right subtree of the root in the same way. [Knuth, 1971]

You can see the similarity of the binary search method and binary search trees looking at these four situations. They are the exact same situations that arise during a binary search. However, the main difference between the binary search algorithm and binary search trees is that the first is used with arrays and similarly stored information, and the second is used with nodes.

The last case where search algorithms can be used is the quantum computers of the future. The big difference between today’s computer and a quantum computer is that the computers right
now “exist in one of two states: a 0 or a 1,” but a quantum computer isn’t limited to two states [Howstuffworks, 2000]. Most research having to do with quantum computers is theoretical. This includes Lov Grover’s theories, some of which are discussed in the following paragraph.

A normal search algorithm would take $O(N)$ steps to solve because there are $N$ items to be examined. “However, quantum mechanical systems can simultaneously be in multiple Schrodinger cat states and carry out multiple tasks at the same time. [Grover’s algorithm presents an] $O(\sqrt{N})$ step algorithm for the search problem” [Grover, 1996]. Invented in 1996 by Lov Grover, Grover’s algorithm is an algorithm that is “significantly faster than any classical algorithm can be” [Grover, 1996]. He poses the following problem in his paper: “there is an unsorted database containing $N$ items out of which just one item satisfies a given condition - that one item has to be retrieved. [...] There does not exist any sorting on the database that would aid its selection” [Grover, 1996]. The last statement means that, other than Grover’s algorithm, the quickest and only way to solve this problem would be to use the linear search method. Grover’s algorithm is essentially a replacement for the linear search method. From Matthew Hayward, we can see an outline of how Grover’s algorithm works.
Grover’s algorithm is as follows:

1. Prepare a quantum register to be normalized and uniquely in the first state. Then place the register in an equal superposition of all states \( \left( \frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}} \right) \) by applying the Walsh-Hadamard operator \( \hat{W} \). This means simply the state vector will be in an equal superposition of each state.

2. Repeat \( O(\sqrt{N}) \) times the following two steps (the precise number of iterations is important, and discussed below):
   
   (1) Let the system be in any state \( S \). If \( C(S) = 1 \), rotate the phase by \( \pi \) radians, else leave system unaltered. It is worth noting that this operation has no classical analog. We do not observe the state of the quantum register, doing so would collapse the superposition. The selective phase rotation gate would be a quantum mechanical operator which would rotate only the amplitude proportional to the marked state within the superposition.

   (2) Apply the inversion about average operator \( \hat{A} \), whose matrix representation is: \( A_{ij} = 2/N \) if \( i \neq j \) and \( A_{ii} = -1 + 2/N \) to the quantum register.

3. Measure the quantum register. The measurement will yield the \( n \) bit label of the marked state \( C(S_M) = 1 \) with probability at least \( 1/2 \).

[Hayward, 2008].

Some of the uses of Grover’s algorithm are as follows: estimating the mean and median of a set of numbers, solving the collision problem, “to solve NP-complete problems by performing exhaustive searches over the set of possible solutions” [Quantiki, 2006].

Though the tabu search, simulated annealing, constraint propagation, combinatorial search, the linear search, the binary search, binary trees, and Grover’s algorithm are all different algorithms, they share the same purpose: to search for an answer. Search algorithms, though seemingly unimportant to those who have never been forced to use one, are an integral part of the new technology age of the world. Without search algorithms we would not be able to simply type in a phrase at Google.com and get quick information. Without search algorithms we would not be able to use spell check on our documents. We would not be able to find a video on youtube. We would be incapable of doing many of the things we take for granted this day and age. Search algorithms are
applied for explicitly stored databases, virtual search spaces, quantum computers, as well as our everyday lives. The next time you search for something, whether it be for a television show or for your keys, I hope you keep in mind that there are millions of other people searching for something else and many times with the assistance of a search algorithm.

REFERENCES


