Nature's Algorithms: Natural and Social Metaphors in Algorithm Design

JOSEPH CARNAHAN[*] RAHUL SIMHA[†]

Abstract

Combinatorial optimization problems are a class of problems in which, typically, every possible solution must be evaluated to be certain of finding the optimal solution. Since such exhaustive searches are often impractical, there is now a vast body of heuristic algorithms for combinatorial optimization problems. Among these are algorithms based on metaphors borrowed from other areas of science. The idea is that key elements of physical processes can be used abstractly to form the basis of an optimization algorithm. This paper presents a broad overview of several metaphor-based algorithms, including the widely-used genetic and simulated annealing algorithms, as well as some newer algorithms.

[*]Department of Computer Science, College of William and Mary, Williamsburg, VA 23187.

[†]Corresponding author. Department of Computer Science, The George Washington University, Washington, DC 20052. Email: simha@seas.gwu.edu. Supported in part by grant CDA-9712718 from the National Science Foundation.

Introduction

Certain problems in computer science, mathematics and engineering continue to evade any search for an efficient algorithm to solve them. These particularly include combinatorial optimization problems whose key feature is an often extremely large set of potential solutions that must be evaluated exhaustively to be certain of finding the best solution.

Possibly the most famous combinatorial optimization problem is the *Traveling Salesman Problem*: Given a set of points, either nodes on a graph or cities on a map, the objective is to find the shortest possible tour that visits every single point exactly once and then returns to its starting point. Simple heuristics for choosing a tour, such as iteratively picking the closest city to the current one, do not necessarily yield optimal tours. At the same time, an exhaustive search of all of the possible tours to find the best one can be computationally impractical, since there are $(n-1)!/2$ possible tours for any n-city traveling-salesman problem. Thus, while there are only 2520 possible solutions (tours) for a small 8-city problem, there are 653,837,184,000 possible solutions for a 16-city problem. Even if a computer could evaluate a single tour in one-millionth of a second, it would require almost eight straight days to evaluate all of the possible solutions to a 16-city problem. Furthermore, this extensive computation would be of little use in another instance (with different data) of the same problem.

Another example of a combinatorial optimization problem is the *Location-Allocation Problem*: Given a set of facilities, each of which has a capacity to serve a certain number of nodes on a graph, the objective is to place the facilities on the graph so that the average distance between each node and the facility that serves it is minimized. To put the problem in more practical terms, where should the police department build stations (of possibly different sizes) in a new town so that the residents are as close as possible to the stations that serve them? Figure 1 shows an example of a location-allocation problem and its solution, with the square nodes representing facilities serving nodes in their respective regions.

Strictly speaking, any problem that has a large set of discrete solutions and a cost function that rates those solutions relative to one another is a combinatorial optimization problem. In combinatorial optimization problems, the number of solutions grows exponentially as the size of the problem grows, which is what makes them so challenging. For traveling salesman problems, for example, the size of the problem is the number of cities to visit, and the solution set is the set of all possible tours through those cities. The cost function for a traveling salesman problem is the length of the tour. Meanwhile, for location-allocation problems, the size of the problem is affected by both the number of nodes in the graph and the number of facilities. The solution set of a location-allocation problem is the set of possible ways that the facilities can be placed on the graph crossed with the number of possible ways that the nodes can be assigned to facilities. The cost function is the average distance from each node to its corresponding facility.

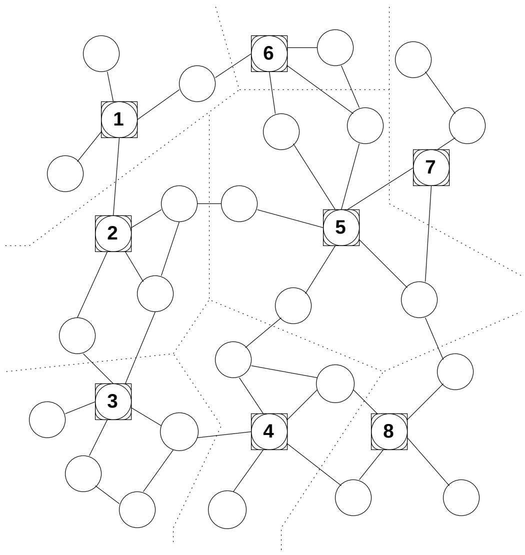

Figure 1: A Solution to an Arbitrary Location-Allocation Problem

With problems like these, the engineer's job is to find algorithms that can reliably produce results that are close to optimal in a reasonable amount of time. In the search for innovative answers to these algorithmic questions, engineers and scientists have sometimes borrowed analogies from biology, physics, and even economics to try to solve these problems. A scientist may notice a process by which a value in nature is optimized, and then that process becomes a metaphor for an algorithm to solve an abstract problem. There is now an entire family of these metaphor-based algorithms, some of which have proved quite useful.

In this paper, we will look at four successful attempts to apply metaphors from other sciences to the development of algorithms for combinatorial optimization problems. In addition, we will look at an example of where such an application was attempted with less successful results. Finally, we will discuss the overall usefulness of metaphors from other sciences in algorithm development.

First Example: Evolution and the Genetic Algorithm

As their name suggests, genetic algorithms attempt to mimic the process of biological evolution in the development of a solution. The algorithm begins by randomly generating a reasonably large set of candidate solutions (the initial "population"). Then, each solution is rated for its "fitness" according to some metric that is appropriate for the problem. For example, an attempt to solve a traveling salesman problem would start by generating a set of possible tours and then calculating each tour's length. The shorter the tour length, the higher the fitness of the solution.

After all of the solutions' fitnesses are calculated, a new set of solutions is produced by randomly picking solutions (with some replication, to keep the population more or less constant) out of the original set with a preference for the solutions with higher fitness. This is the "natural selection" of biology, weeding out the weaker members of the population of solutions. Next, members of the new set of solutions are paired, and parts of their solutions are swapped within each pair, with an occasional "mutation" to change a solution slightly. In theory, this swapping reflects the way that DNA is combined in sexual reproduction. Then, this pattern of evaluation, reproduction, swapping and mutation is repeated until a sufficiently satisfactory (low-cost) solution appears to dominate the rest of the population.

Genetic algorithms have been shown to perform fairly well in a wide variety of problems, with the largest difficulties arising when there is no obvious way to "mate" solutions. Also, since genetic algorithms are a general-purpose problem solver, they can often be outperformed by more specialized algorithms that are designed to take advantage of the characteristics of a particular problem.

Second Example: Metallurgy and Simulated Annealing

Simulated annealing is another general-purpose optimization problem solver, whose metaphor came from metallurgy instead of biology. It is known that when molten metal cools too quickly, many desirable qualities, such as strength or magnetism, are lost or degraded. In contrast, when molten metal is cooled slowly by a process called annealing, these properties are retained or enhanced. This idea of slow cooling has been applied in the abstract to the process of searching through the possible solutions to a combinatorial optimization problem.

The simulated annealing algorithm is most easily understood as an improvement on a very simple heuristic, called "greedy local search", that operates as follows:

1. Start with any proposed solution.
2. Change the solution slightly. (For example, swap two cities in a traveling salesman tour).
3. Check to see if the new solution is better than the original. If not, undo the change.
4. Keep going back to step 2 until it is apparent that any small change will only make the solution worse.

Greedy local search has a major weakness: what if a dramatically better solution can be reached by making two or more consecutive changes? This algorithm finds a result that is only *locally* optimal — that is, it is better than any of the results that are close to it, but it may not be the best result overall. The best result overall, which is what the algorithm is seeking, is called the "globally optimal" result. Figure 2 shows both a locally optimal solution and the globally optimal solution to an arbitrary function. Note that a genetic algorithm without mutation is also likely to get 'stuck' in a local optimum.

Simulated annealing attempts to improve on greedy local search by occasionally taking a risk and accepting a worse solution. Specifically, when a solution that is less optimal

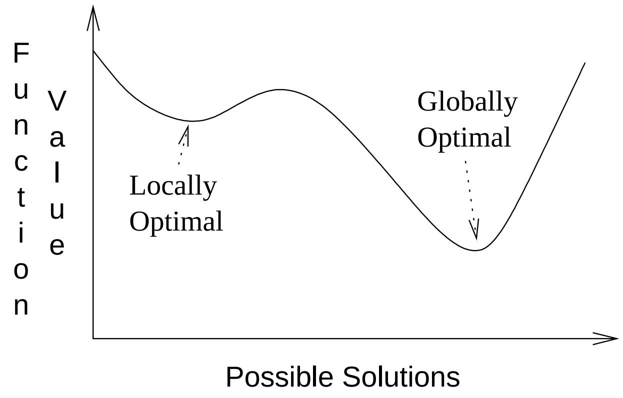

Figure 2: Local and Global Optima

than the current solution is proposed, the algorithm starts by computing ΔC, the difference between the cost of current solution and the cost of the proposed one. Then, the algorithm generates a random number r (the function `Random()` below) in the range $0 < r < 1$, and if r is less than $e^{-\Delta C/T}$, then the proposed change is accepted. Here, T is the metaphoric "temperature", that is iteratively decreased to make this probability of acceptance low at low temperatures. Here is an outline of the algorithm:

```
S    = initial_state; // The current "working" solution
minS = S;             // The best solution so far
T    = initial_temperature;
while (T > final_temperature)
  S' = Generate_New_State(S);
  if (Cost(S') < C(S))
    S = S';
  else if (Random() < exp((Cost(S) - Cost(S')) / T))
    S = S';  // even though Cost(S') > Cost(S)
  else
    // do nothing - stay in same state
  end if;
  if Cost(S) < Cost(minS)
    minS = S;
  end if;
  T = Get_New_Temperature();
end while;
output minS, Cost(minS);
```

At the beginning of the search, the temperature is high and the algorithm is very likely to explore new solutions, even if those new solutions are slightly less desirable than the current one. However, as the search progresses, the temperature is slowly lowered and the algorithm is less and less likely to accept a switch to a locally less optimal solution. Theoretically, it is possible to show that the temperature should be lowered in proportion to $(1/\log(n))$. However, at that rate, the cooling is too slow for the algorithm to terminate within a reasonable amount of time. Realistically, the temperature is usually lowered at a linear or exponential rate. Even with these faster cooling schemes, simulated annealing's compromise between exploring all of the possibilities and homing in on a locally optimal result has proven to be a very effective way to search through the possible solutions of optimization problems.

Third Example: Physics and the Location-Allocation Problem

The previous two examples are general-purpose algorithms that are designed to handle any combinatorial optimization problem. However, the characteristics of a particular problem often lend themselves to a more specialized heuristic. Here too, physical metaphors have played a role in inspiring optimization algorithms.

To solve the location-allocation problem that was discussed in the introduction, an approach has been adapted from the study of moving particles in physics. The facilities that need to be placed on the graph are modeled as particles. The mass of each particle is proportional to the capacity of the facility that it represents, and particles are allowed to move from node to node around the graph. There is a repulsive force between each pair of particles that is inversely proportional to the distance between the particles. This means that to minimize the potential energy of the system, the particles will tend to spread out from one another. Also, to keep the particles from all being pushed to the outer edge of the graph, a repulsive force from an artificial surrounding "boundary" is added.

To fix the particles into a low-energy configuration, a process of aggregation and decomposition is used. Initially, every particle is aggregated into one large particle with the combined mass of all of its member particles. Then, the smallest of all of the particles is removed from the large aggregate particle. Next, the large particle and the small one are moved into whatever positions will give them the lowest potential energy. The smallest particle is fixed into place, and then the process is repeated by removing the next-smallest particle from the large aggregate particle. From smallest to largest, the particles are all eventually attached to various nodes. The configuration of particles that results from this approach turns out to be a reasonably good configuration of facility locations.

Fourth Example: Mathematical Economics and Computer Networks

A familiar pricing mechanism from economics has been used as a metaphor for an algorithm targeted at some optimization problems in computer networks and distributed systems. Although not strictly a combinatorial optimization algorithm, the algorithm can be applied to certain classes of continuous (real-variable) optimization problems, and to combinatorial optimization problems that can be approximated by transformation to a continuous optimization problem.

While the details may vary from problem to problem, the idea is to identify certain parts of the system as "commodities" and other parts of the system as "consumers" who are competing for these commodities. In a real economy, such competition is settled by prices: Each consumer buys as much as they can afford, buying more of whichever commodities they need more than others. The prices, in turn, are influenced by the demands of the consumers, so that commodities which are in high demand become more expensive.

This system of pricing prevents any one consumer from purchasing all of the valuable commodities, while still allowing consumers to buy more or less of a given commodity, depending on how important that commodity is to them. In effect, the consumer is maximizing a utility function, which is a function that weights the importance of each commodity to the consumer. A similar process can be used for a distributed system: Each consumer has a utility function and a fixed amount of money with which to bid. Whether the commodity is processor time, access to a networked printer, or anything else, the bidding process establishes an equilibrium which takes into account the demands of each of the consumers.

For example, consider the situation where two computer terminals wish to share two database files. The files can be stored at either location, but accessing a part of the file which is stored at the other location is much slower than accessing a part of the file which is stored locally. So, an intelligent decision must be made as to how much of each file should be stored locally at each terminal.

Now, suppose that information has been gathered about the probability that an access at each location will require either file. Based on this probability and based on the amount of each file which is already stored locally, a utility function can be defined for each terminal as the probability that any given access at that terminal can be handled locally. Each file is given an arbitrary starting price and each terminal is given a fixed amount of simulated money with which to bid. Maximizing the utility functions within the constraints of the amount of money which they have, each terminal asks for a certain portion of each file. After that, the prices of each commodity are adjusted in proportion to how much of each of them was requested. Based on the new prices, the two terminals recompute their utility functions and make a new bid for a share of the two files. This iterative process (called *tatonnement* in microeconomics) is repeated until the prices have stopped changing and some sort of equilibrium has been achieved.

At any point in the bidding process, if either terminal starts requesting enormous amounts of either file, the price of that file will jump up. With only a fixed amount of money to bid with, the terminal will be forced to ask for less of the now-expensive file. Similarly, if either file is accessed less often than the other one, its price will drop, encouraging both terminals to bid for more of it to increase the likelihood that database queries can be satisfied locally.

Final Example: Particle Physics, Elasticity and the Traveling Salesman Problem

The paper, "Emergent Collective Computational Abilities in Interacting Particle Systems", by Z.Zhang, S.Bai, and G.Li, is another example of an attempt to apply a metaphor from particle physics to solving a combinatorial optimization problem. Specifically, the goal is to solve traveling salesman problems with a simulation of charged particles connected by elastic bands.

The algorithm operates as follows: The cities that the traveling salesman seeks to visit are modeled as N fixed negatively-charged particles. Then, N freely-moving positively-charged particles are placed in a circle around the center of mass of the cities. The radius of this circle is equal to the average distance of the cities from their collective center of mass. The moving particles are connected by a loop of elastic string, which applies a force which is proportional to the distance between the particles. Figure 3 shows an example of the particles in their initial configuration for a ten-city traveling salesman problem.

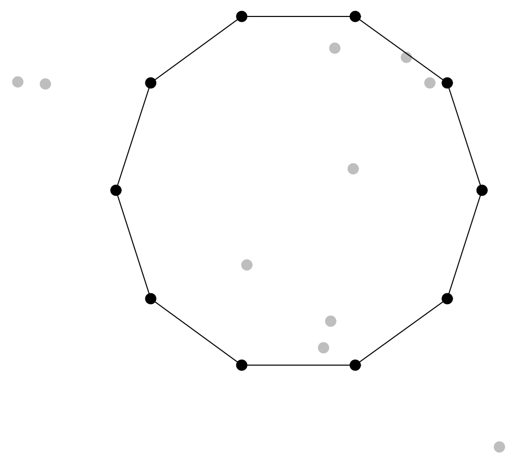

Figure 3: An Initial Layout for the 10-City String-Particle Model
(The gray points are the cities. The black points are the moving particles.)

To find a solution, the particles are set free to move under the influence of their electric charges and the force of the elastic string. Figure 4 shows the particles for the example problem after they have begun moving. Whenever a moving particle collides with a city, it locks into place there and stops. When all of the particles stop moving, the elastic string now traces a tour through all of the cities.

This string-particle metaphor shows some promise as an algorithm, since the force of

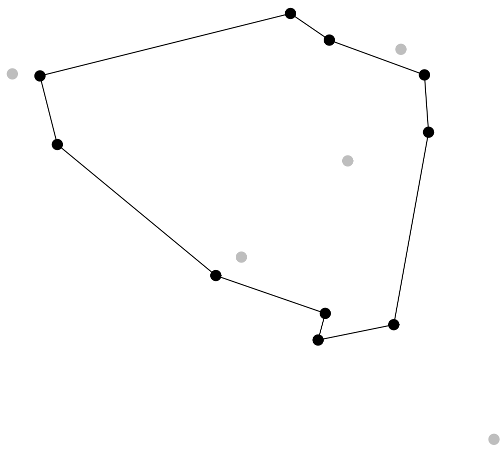

Figure 4: An In-Progress Snapshot of the 10-City String-Particle Model

the elastic bands will try to minimize the distance between adjacent particles. However, some empirical testing has shown that other methods produce better solutions to traveling salesman problems. Figure 5 is a graph which compares the costs of the solutions found by the string-elasticity model as compared to the solutions found by two other metaphor-based algorithms when applied to the same set of traveling salesman problems. Clearly, simulated annealing produces the lowest-cost tours, followed by genetic algorithms. (In fact, genetic algorithms do not perform well for traveling-salesman problems because most schemes for mating two different tours actually reduce the average fitness of the two tours.) Apparently, while the particle physics metaphor works well for solving the location-allocation problem, it does not apply as well to the traveling salesman problem.

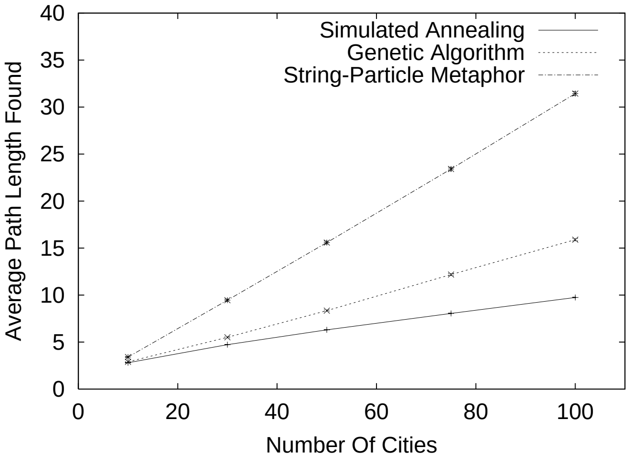

Figure 5: Comparison of Algorithms for Solving Traveling Salesman Problems

Discussion

As we have seen, there are now a variety of algorithms inspired by processes and systems studied in other sciences. These metaphor-based algorithms often succeed by introducing a completely new approach to a familiar problem. This new idea may or may not be better than any previous ones, but the novelty itself has propelled several of these metaphor-based algorithms to popularity. However, the catchy sound of a new metaphor does not necessarily reflect on the algorithm which is based on that metaphor, and it certainly gives no guarantees about the quality of a particular implementation of that metaphor-based algorithm. In particular, genetic algorithms sound very intriguing due to the fact that as human beings, we ourselves are the result of an evolutionary process. Naturally, there is some appeal in the assertion, "The algorithm that created me is a good algorithm."

Personal appeal aside, there has been a fair amount of empirical testing of several metaphor-based algorithms. Particularly for combinatorial optimization problems, where conventional search algorithms face such difficult challenges, metaphor-based algorithms have provided useful alternatives. In addition, metaphor-based algorithms are often fairly simple to implement. It is usually quicker to apply simulated annealing than to custom-design a heuristic for a particular problem even though experience shows that specially designed heuristics can often outperform general purpose heuristics such as annealing. Also, some metaphor-based algorithms lend themselves very well to parallel processing, whereas considerable parallelization effort may be required for a custom heuristic.

Finally, we note that genetic algorithms can be confused with two other areas of research, namely DNA computing and evolutionary algorithms. The former involves storing data in strands of actual DNA and using enzymes to break down and recombine the DNA in order to process the data. The latter is the idea of applying genetic algorithms to the process of algorithm development itself. In this case, the population of solutions is actually replaced by a population of algorithms, which are allowed to compete and recombine to produce new algorithms. Also, the term "emergent behavior" is sometimes used to describe macro-behavior obtained from interacting components, leading to yet another biology-inspired optimization algorithm, the neural network.

Summary

Metaphor-based algorithms have proven their use in optimization, both as general purpose heuristics as well as custom heuristics for particular problems. The general purpose ones, such as simulated annealing and genetic algorithms, have spurred a vast literature of their own. However, while a creative metaphor may make an an algorithm sound attractive, the algorithm itself still needs to provide convincing results.

For Further Reading

- D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Pub. Co., Reading, Mass., 1989.
- E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John-Wiley & Sons, 1989.
- R. Simha, W. Cai and V. Spitkovsky. Simulated N-Body: A New Particle Physics-Based Heuristic for a Euclidean Location-Allocation Problem. *J. Heuristics*, to appear.
- J.F. Kurose and R. Simha. A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems. *IEEE Transactions on Computers*, May 1989.
- D.F. Ferguson, C. Nikolaou, J. Sairamesh and Y. Yemini. Economic Models for Allocating Resources in Computer Systems, in Market based Control of Distributed Systems, Ed. Scott Clearwater, World Scientific Press, 1996.
- X. Yao (Ed.). Evolutionary Computation: Theory and Applications. *World Scientific Pub.*, 1999.
- L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021, 1994.
- S.S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice-Hall, 1998.