from Proceedings of the third conference on Artificial Intelligence Applications, 231-235, IEEE, 1987.

SCHEDULE OPTIMIZATION WITH PROBABILISTIC SEARCH

Lawrence Davis and Frank Ritter
Bolt Beranek and Newman Inc.
10 Moulton, Cambridge, MA 02238
DDavis@bbn.com Ritter@bbn.com

Abstract

The authors describe an application of genetic algorithms and simulated annealing to a class section scheduling problem. Included are a comparison of the two types of optimizing technologies, a discussion of the application domain, a description of how annealing can be used to optimize in the domain with annealing, and a description of new techniques used to optimize annealing algorithms or other numeric processes with a genetic algorithm. The authors are able to conclude that the procedures they have used for this application will generalize to similar domains.

Introduction

This paper describes the creation and optimization of a probabilistic search routine to carry out scheduling of students in class sections. The search routine uses simulated annealing. The routine itself was optimized by the use of a genetic algorithm. The resultant system performs the scheduling task better and more quickly than the humans who currently perform it.

We will first discuss the problem and compare the algorithms that we propose to use. Following this discussion, we describe our solution to the problem using simulated annealing* (SA), and our improvements to the algorithm. Next, we cover our optimization of the annealing parameters with the use of a genetic algorithm. Finally, we summarize our results and characterize other areas where these algorithms may be used.

Discussion of the Problem

The problem is a scheduling problem with several interacting constraints. Every semester it is currently solved without computer assistance at Harvard University. Mr. Jeff Bradley, a preceptor in the Expository Writing Department, provided us with the data and the problem. In the data he provided us, 118 students were to be placed in 120 classroom seats, divided into 8 classes of 15 seats. Each student chose 4 sections, and ranked them in order of preference, 1-4, lowest being most favored. The objective of the placement was to give all the students their lowest possible choice. The teachers of each section would also like to see a roughly 50-50 male/female ratio, which they believe results in livelier discussion and participation. Obviously not everyone will get his or her first choice, and statistically the distribution in each class will not be 50-50. Currently the teacher sorting the student into classes has several rules of thumb for what is preferable. For example, he would rather see two people in their second choice than one in their first choice and one in their third. Also, fourth choices are to be avoided if at all possible. There are additional more complicated heuristics for balancing gender across sections.

A function that can estimate the unhappiness (energy) of the students and teachers given a distribution was constructed. This was done by having Mr. Bradley rank pairings and triplets of students and designing the function to conform to these example constraints. The function (equation 1) was also designed with efficiency in mind: it is linear, and it has a simple differential which can be computed from local information. This is useful because the comparison of local operations such as flips for a SA is now very economical. The professor's solution after an hour of work had an unhappiness value of 242.

Discussion of Technologies

The problem is one well-fitted for probabilistic search techniques -- the search space is large, and the globally best answer is not required. We considered two such techniques: genetic algorithms and simulated annealing. In this section, we compare them and describe which features influenced our decision of how to use them best.

The technologies of genetic algorithms and simulated annealing share many features. Both are used to search for near-optimal solutions in large and complex search spaces. Both have been shown to offer significant improvement over conventional algorithms in some domains. Both are stochastic in nature. Both are computer metaphors for optimization processes observed in nature. Both are objects of intense interest at present to individuals in AI and other research communities.

The two techniques also seem to differ along many orthogonal dimensions. Genetic algorithms store the state of their search in the genetic composition of a population of individuals, while simulated annealing saves a single individual. The search space could be thought of as the possible states of individuals multiplied by their chance of being visited. As individuals are chosen

*Kirkpatrick[1] is an excellent introduction, and Metropolis[2] is the seminal paper across time, the search space is explored. Seen in this context, simulated annealing greatly resembles a genetic algorithm with a one-member population and a single genetic operator that replaces the current individual while guaranteeing certain statistical properties of the collection of such individuals.

```
F(sort) =                                        (1)
      4
      S   (Count(i) : Cost(i))
     i=1
          +
      8
      S   (Ratio-cost(Ratio(c)))
     c=1

   Where:
      i represents a student preference
        from 1 (first choice)
        to 4 (last choice)
      Count(i) is the number students
        in their i'th choice.
      Cost(i) is how unhappy students
        are in an i'th choice section.

      c is the number of classes,
        in our example 8.
      Ratio(c) is the male or (female)
        excess in the c'th class.
      Ratio-cost(x) is how unhappy the
        teacher is with that excess.

      Ratio-cost(i) = abs(i)

      Cost(1)   = 1
      Cost(2)   = 3
      Cost(3)   = 6
      Cost(4)   = 18
```

Researchers of the two techniques have represented the individuals in different ways, and the operators applied to those individuals have been different as well.

The algorithms have tended to be applied at different levels of parallelism. SAs are often made parallel on a very fine grained level, with each part of the solution an item to be operated on, flipping in and out of the solution independently of the other parts. GAs generally have each population member (a whole solution) evolve on its own, producing a parallelism that is coarser-grained.

The communities of researchers developing and applying the two technologies for search are nearly disjoint, and the domains in which the two techniques have been applied have been disjoint as well. (This situation will probably change as more representations and operators are added to the body of genetic algorithm knowledge and as SA also grows.)

Genetic algorithms incur more overhead than SA. GAs maintain multi-member populations and store numerical evaluations for each member. In the process of reproduction, computation is required to determine, on the basis of the evaluations, which parents reproduce, how often, and in combination with which other parents. All of this permits GAs to exploit the use of crossover operators between parents.

GAs are capable of using crossover operators to accelerate the initial stages of their search process, before the population is relatively homogeneous. Crossover operators allow GAs to combine parts of two (or more) successful parents, with a reasonable chance of yielding an offspring that is better than either parent because it shares some beneficial features of both. The matrix representation techniques commonly used by simulated annealing are not amenable to crossover operators of this type, while the bit string representations used by many GA researchers are.

The two technologies have not been applied to the same domains. There are good reasons for this, the most important being that the two technologies do not appear to perform well on the same types of problems. SA has been used when one can represent one's solutions so that perturbations of them are rapid and the effects of such perturbations are local. With such representations, a simulated annealer is able to test a large number of individuals in a short amount of time in a statistically ordered manner. (A common representation of this type is a matrix, with perturbations limited to pairwise exchanges of matrix members. Semiconductor wiring problems and travelling salesman problems are two examples of the many types of problems that have been approached by SA researchers in this way.) Genetic algorithms are applied to problems where the overhead of population maintenance is paid for by rapid convergence through crossover, or where the cost of evaluating individuals is so high that more random search procedures are undesirable.

Selection of the Appropriate Technology

Given these considerations, it is at present an engineering choice which to use for practical applications. For the scheduling problem described here, the utility function may be quickly and locally recomputed, and so we chose simulated annealing as the optimizing technology.

The derivation of an annealing schedule, however, is a problem requiring intensive computation when individuals are evaluated, and so we used a genetic algorithm to carry out the derivation. Results described by Davis[8] show that GAs may be applied to a wider range of applications than had been thought. The techniques used require that one depart from bit string representations of solutions, while preserving the crossover potential of the representations used. Optimization of annealing schedule parameters for the scheduling problem turned out to be a problem that genetic algorithms were well suited for.

In the sections that follow, we describe the annealing approach to the problem, and then the genetic approach to optimizing the annealing schedules.

Distribution Generating Functions

Simulated annealing relies on the system visiting states which have a Boltzmann distribution (lower energy states are exponentially favored). This distribution is generated by the way perturbations are accepted. The choice is done probabilistically by comparing a random number between 0 and 1 with the value of the acceptance probability generating function. We will refer to this equation as the distribution generation function. There are two functions of this type in current use. The Boltzmann Machine [9] uses eq. 2, and Kirkpatrick[1] and Metropolis[2] use eq. 3.

The first equation actually comes closer to replicating reality. The function does not unfairly favor motion that increases or decreases the system's energy. This can be verified by noticing that the function sums to 1 for P(-deltaE*) + P(deltaE). However, the other equation, first used by Metropolis et. al., violates a fundamental law of physics which states that a system can not use information about its present state in a way different from information about the state that it may transition into. Always accepting a transition that lowers the energy is an example of using information in this way. This presupposes that it was known that the initial state, and not the final state, was higher in energy, rather than considering the pair without regard to which was which.

```
             (Boltzmann)                            (2)
P(deltaE)  =   1/(1 + exp(-deltaE/kT))
                  for all deltaE

             (Metropolis)                           (3)
P(deltaE)  =   1.0                  deltaE <= 0
           =  . exp(-deltaE/kT)     deltaE > 0
```

Each equation has been used successfully. Given that one equation was known to better approximate the physical metaphor but was more computationally expensive, a series of experiments was designed and carried out to determine their relationship.

An annealer that stayed at each temperature step until 1400 flips were accepted from T = 1000 to .01, changing by deltaT**, was used in the first series of runs. It was run on the example data supplied by the professor 15 times for each deltaT, and the results were averaged. These values are shown in Figure 1†.

The results were pleasing. The distribution generation function that more closely replicated reality provided the better results. The values were inversely proportional to deltaT, so the values were improved proportional to the log of the total number of flips. But when the total number of flips used was compared, the more correct generation function was also using significantly more flips because it wasn't accepting all the negative deltaE flips, as the less realistic function was doing. A second annealer which did a set number of flips at each temperature was used to normalize the resources available to each algorithm. The results from these runs are shown in Figure 2.

This time the two distribution generation functions did not differ in a statistically significant way (Metropolis 228.11 vs. Boltzmann 228.08). The Metropolis distribution generation function was chosen for the annealer in this paper. It offers the same performance, and a small speedup because it doesn't do any calculation when it automatically accepts a beneficial flip, and has a simpler equation to calculate when the proposed change is increases the cost and it must probabilistically choose.

*$\text{deltaE} = E_{final} - E_{initial}$

**$T_{i+1} = \text{deltaT} * T_i$

†Values on both graphs are rounded to the nearest integer by the graphing routine

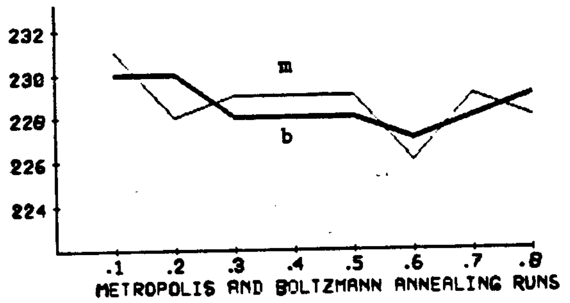

Figure 1: Average value of 15 runs vs. deltaT for fixed number of acceptances

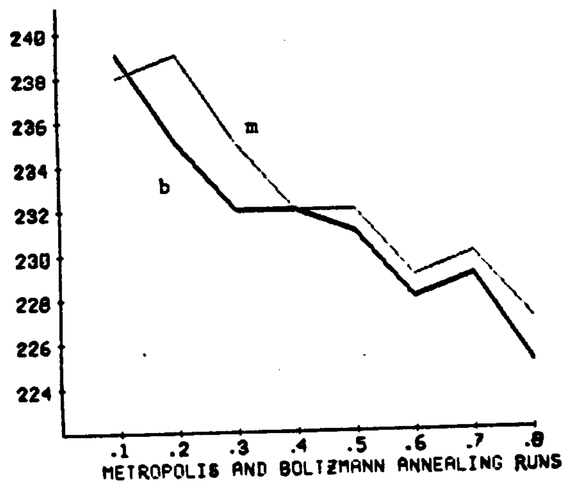

Figure 2: Average value of 15 runs vs. deltaT for fixed number of flips

Figures 1 and 2 should be reversed.

<u>Operators for the Genetic Selection of Annealing Parameters</u>

Although we did not know what combinations of rate of change (TCHANGE), maximum temperature (TMAX), and minimum temperature (TMIN) would perform well in the scheduling domain, trial and error had given us bounds within which we were confident those values would fall. Finding good combinations within those bounds, however, was complicated by the stochastic nature of the annealing process. Successive runs of an annealer that sampled 100,000 individuals varied widely in their final score, from 220 (best) to 238 (worst), with a fairly normal distribution of values around the score 227 when the parameters were reasonable. Each run of the annealer over 100,000 individuals consumed slightly more than five minutes' time on a Symbolics 3640 Lisp Machine.

The selection of parameters for the annealing schedule was a different sort of problem; the evaluation process is noisy; we knew little about the relation between the domain and the technology; and evaluation time was by far the most cpu-intensive part of the program, because the effect on an individual caused by altering a parameter's value could not be calculated without a complete re-evaluation. Accordingly, we decided to use a genetic algorithm to optimize the values of TCHANGE, TMAX, and TMIN.

The design of the genetic algorithm was as follows:

1. Our population was made up of 30 3-tuples. Each 3-tuple was composed of a value for TCHANGE, TMAX, and TMIN generated randomly within bounds of reasonableness determined empirically. The small size of the population was forced by time (and budget) constraints.

2. We applied the GA to these populations through 30 generations. Thus, in our first run, 900 individuals were evaluated.

3. The evaluation of each combination of parameter values was the result of performing a simulated annealing over 100,000 individuals using those values. This evaluation was carried out in every generation for every individual in the population. Thus, each evaluation of a given triplet of parameter values could (and did) vary significantly.

4. One of our genetic operators was a traditional one, but the others were not, for our parameter values were not coded as binary strings. We hypothesized that the optimization search space was not one with the sort of periodicity that lends power to bit string crossover. Rather, we expected there to be a limited number of isolated minima that would be difficult to detect owing to the amount of noise in the evaluation process. Grefenstette and DeJong[7],[5], have carried out experiments with meta-level genetic algorithms using bit string representations of parameter values, but our experiment appears to be the first one in which genetic algorithms employed operators that manipulated objects interpreted directly as numbers.

We used four operators: CROSSOVER, AVERAGE-CROSSOVER, MUTATE-VALUE, and CREEP-VALUE. Their effects are described below.

CROSSOVER functioned like the traditional crossover. Given the two parents (1 2 3) and (4 5 6), CROSSOVER produced one of these four lists as offspring: (1 5 6), (1 2 6), (4 2 3), or (4 5 3).

AVERAGE-CROSSOVER applied to two parents produced a list of the average of their respective members. For example, AVERAGE-CROSSOVER applied to the parents (1 3 2) and (1 7 4) produces the child (1 5 3).

MUTATE-VALUE replaced a value in a parent with a randomly generated value within the bounds of reasonableness for that value. MUTATE-VALUE applied to the second member of the parent (1 3 2), for example, would produce (1 n 2), where n is one of the reasonable values the second member can take on.

CREEP-VALUE replaced a value in a parent with the result of creeping that value up or down from 1-5 times the creep-factor associated with that value. More precisely, where V is the current value, R is a random number from 1 to 5, C is the creep value of that member obtained by dividing the difference of the upper and lower bounds on that member by 100, and S is +1 or -1, randomly chosen, then CREEP-VALUE replaces V with V + (S * C * R) unless this result falls outside the reasonable bounds for this parameter. In that case, V is replaced by the boundary value that is exceeded.

5. The probability of application of these operators over the course of the run from generation 1 to generation 30 was interpolated as follows: CROSSOVER from 40% to 10%; AVERAGE-CROSSOVER from 40% to 10%; MUTATE-VALUE from 4% to 1%; and CREEP-VALUE from 1% to 4%. These probabilities were derived by trial and error from test runs on a different problem that allowed near-instantaneous evaluation. They, too, could have been derived by genetic techniques similar to those described here, but at some point in the meta-hierarchy the values must be set by theory, by guess, or by experimentation. The interpolation of parameters is a technique that is not generally used by genetic researchers, but our experimental results with interpolation were consistently better than those without it, so we used it here.)

6. The reasonable bounds for the parameters were as follows: TCHANGE from .1 to .95; TMIN from .0001 to .1; TMAX from 20 to 1000. Using 1/100 of the difference between the maximum and minimum bounds as the creep factor kept the grain size consistent for each parameter.

7. Evaluations returned from the annealer were processed before reproduction took place. Given E (the list of evaluations of the 30 current parents) and MIN (the smallest value in E), each value V in E was replaced by the greater of 1 or V - (.09 * MIN).

8. The result of the genetic run was taken to be the average of values of the ten best parents in the final generation. The population had converged by that time, with most of its diversity apparently due to creeping. Averaging yielded the result (TCHANGE = .516, TMIN = .061, TMAX = 560), a combination that was representative of the members of the final generation.

Assessment of the Annealing Schedules

We carried out evaluation of these results in two ways. The first consisted of a series of trial-and-error experiments that were carried on without knowledge of the genetic population's composition during the two weeks in which the genetic strain was evolving. We had hoped that these hand-derived values (.85 .01 1000) would be similar to the genetically-derived values, but they differed significantly. This discrepancy caused us to question the genetic values. The nature of the values themselves was suspicious, in that the genetic values lie close to the midpoint of the reasonable ranges given them. It appeared that the AVERAGE-CROSSOVER operator might have inappropriately driven the population toward the center of the ranges of values, rather than exploring the areas in the neighborhood of the best values. In addition, the value of TMAX derived by us lay on the boundary of our previously-derived range of reasonable values, and we worried that we had inappropriately delimited TMAX's parameter's values.

Given these considerations, we compared the two results by running a simulated annealer over 100,000 individuals 200 times for the hand-derived parameters and 200 times for the genetically-derived parameters. The average final value for the hand-derived parameters was 227.2, while that for the genetic parameters was 226.6. This improvement was statistically significant with a confidence level > 99% for n = 200. Some of our doubts were allayed by this test. The genetic algorithm had given us a better combination of parameters than our own explorations had found. (The fact of the difference, rather than its magnitude, is important. Our use of a large raw evaluation function has made the improvement seem small. The evaluations sampled ranged in value from 220 up. One could scale them by subtracting 219 from each, or by rewriting the evaluation function so that a student's receiving a preferred class costs 0, second choice costs 1, and so forth. Using 219 as a basis, the improvement from 226.6 to 227.2 is an improvement of 8%.

Our second test of the genetic approach lay in carrying out another run, different from the first in three ways: the probability of AVERAGE-CROSSOVER was diminished to extend from 20% to 5%, the upper limit on TMAX was raised to 1300 from 1000, and we used the logarithms of the values of TCHANGE, TMAX, and TMIN in each parent so that the midpoints of the parameter ranges were changed. The result of this run, computed as before after 30 generations, yielded the logarithms of the parameter values (.566 .0042 67.7). The first value, .566, was close to the comparable value from the first run, and far from .308, the midpoint of its logarithmic range. The second, .003, was 33% greater than its midpoint, but far from the comparable value found in the first run. The third, 67.7, was closer to 161, the midpoint of its range, than it was to 560, the comparable value from the first run. The average of 200 runs with these values was 226.8.

The values .516 and .566 for TCHANGE could be independently validated. As shown in Figure 3-2, when TMIN and TMAX were held constant, the best value of TCHANGE lay in the vicinity of .58. We concluded from these results that the AVERAGE-CROSSOVER operator did not overpower the most critical parameter value in any significant way, but that it might have caused drift in parameter values that are less important.

Summary

We believe that there are several lessons and techniques that are worth noting for others who wish to apply these general algorithms in similar domains:

- The annealing technique out-performed the humans on the problem in a fraction of the time, and the genetic algorithm found better parameter settings for the annealer than the humans found.
- The results shown in Figures 1 and 2 above indicated that either Boltzmann distribution generating function produces roughly the same behavior for this problem. Using the Metropolis function (Eq. 2) should allow the Boltzmann Machine and other programs based on simulated annealing to run faster, with no loss of performance.
- For the individual considering applying these technologies, Figure 1 bears out the theoretical prediction that each increment of improvement yielded by a simulated annealer costs exponentially more.
- Our numerically-oriented genetic operators such as CREEP-VALUE appear to optimize function parameter values successfully.
- Employing the logarithms of parameter values in order to alter the shape of the search space resulted in a useful check on the results of GA runs over other shapings of the search space.
- The interpolation of probability settings for operators in a GA produced useful results; our suspicion that genetic algorithms with interpolated values will out-perform ones with constant values is another area for further work.
- Kirkpatrick et al. in[1] describe experiments in which simulated annealing yielded solutions that were 10 to 15 percent better than those found by greedy algorithms or human performance. We have determined here that meta-level applications of such technologies can improve the performance of untuned SA. While the effects of a similar improvement will not be as great at the meta-level, they are significant nonetheless, and in a production environment one would surely want the option of acquiring them.

Acknowledgements

Marta Turnbull and Jeff Bradley provided the initial problem definition, and numerous insights into it. Marta wrote the initial implementation of the annealer in PASCAL. We would also like to thank Martha Steenstrup for her constructive comments on an earlier draft of this paper.

References

1. Kirkpatrick, S., Gelatt Jr., C.D., and Vecchi, M.P., "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, May 1983, pp. 671-680.

2. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller A., and Teller, E., "Equation of State Calculations by Fast Computing Machine", *Journal of Chemical Physics*, Vol. 21 1953, pp. 1087+.

3. Holland, John H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, l975.

4. Goldberg, David, and Thomas, Amanda L., "Genetic Algorithms: A Bibliography 1962-1985", Tech. report 86001, Clearinghouse for Genetic Algorithms, University of Alabama, 1986.

5. DeJong, Kenneth, *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD dissertation, University of Michigan, 1981.

6. Bethke, Albert D., *Genetic Algorithms as Function Optimizers*, PhD dissertation, University of Michigan, 1981.

7. Grefenstette, John J., "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-16, No. 1, Jan/Feb 1986, pp. 122-128.

8. Davis, Lawrence W., "Applying Adaptive Algorithms to Epistatic Domains", *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, IJCAI, 1985, pp. 162-164.

9. Ackley, David, Hinton, Geoffrey E., and Sejnowski, Terrence J., "A Learning Algorithm for Boltzmann Machines", *Cognitive Science*, Vol. 9, No. 1, January 1985, pp. 147-169.