EXAMINATION TIMETABLING

E. Parkinson and P.R. Warren
Department of Computer Science
University of Natal, Pietermaritzburg

Abstract

We investigate the performance of simulated annealing and tabu search, two new general problem solving heuristics, on an examination timetabling problem. These two techniques are described here and issues arising from applying them to examination timetabling are discussed. The problem we use as a test case is one that appears in Fang (1992) in which genetic algorithms were used to produce approximate solutions and experimental results were presented showing the performance of genetic algorithms on this problem. Finally, we compare tabu search, simulated annealing and genetic algorithms on the basis of experimental results and draw some conclusions.

Introduction

Timetabling examinations is an activity that takes place regularly in almost all universities and educational institutions. Ever since the advent of digital computers, the possibility of applying them to timetabling has been researched---the idea of reducing the time and manpower required to construct timetables being an attractive prospect. For an overview of this research see for example Schmidt and Ströhlein (1979) which gives an annotated bibliography of work on timetabling that appeared before 1980, de Werra (1985) and Carter (1986) which deals more specifically with examination timetabling.

In order to construct timetables that will be convenient for both students and teachers, sophisticated scheduling algorithms are needed. The timetabling problem however, like many other scheduling problems, usually takes the form of a hard combinatorial optimization problem and no fast exact algorithms exist for solving it, making it necessary to resort to heuristic methods. The types of constraints on the timetable that should be incorporated can differ greatly from one institution to the next and most algorithms are designed for either simple general exam timetable models or for very specific types of constraints and models. Few algorithms can be applied to a broad range of problems and new algorithms and heuristics have to be developed if the problem changes (de Werra 1985).

Over the last few years, three stochastic techniques have been applied to combinatorial optimization and other problems with some success: genetic algorithms (GAs), simulated annealing (SA) and tabu search (TS). These methods are designed to be robust general problem solving algorithms, that is, they perform well over a wide range of problems due to the fact that they make use of very little problem specific heuristics or information. All three techniques proceed by generating possible solutions from a solution space, trying to find a solution that minimizes (or maximizes in some cases) an objective function defined over the solution space. Usually the objective function takes the form of a penalty or cost function that indicates, for each solution in the solution space, the penalty associated with it. In cases were the solution space is too large to find an optimal solution by evaluating the objective function for all solutions, even when controlled backtracking and advanced tree pruning techniques are used, these stochastic methods can find near optimal solutions quickly by evaluating only a very small fraction of the total number of possible solutions. They can also be used to find good approximate solutions to many optimization problems provided that we can associate a cost or penalty with each solution in the search space, and therefore these methods can be used on a very wide range of problems.

The Edinburgh Problem

Fang (1992) describes an examination timetabling problem encountered at the university of Edinburgh. He also goes on to investigate the performance of several genetic algorithms using different parameters and genetic operators on this problem. We won't discuss genetic algorithms in detail here. Instead we will compare tabu search and simulated annealing with the results Fang obtained with genetic algorithms for the Edinburgh problem.

The timetabling problem in Fang (1992) is specified as an optimization problem and is therefore highly suitable to the GA, TS and SA approaches. A number of exams have to be scheduled in a number of days, where each day is divided into 4 sessions — 2 sessions each half day before and after lunch. A timetable in N days is an assignment of sessions numbered N to $4N$ to the set of exams numbered 1 to K, for K exams that have to be scheduled. For each $i, 0 \leq i \leq N\text{-}1$, the sessions $4i+1$ and $4i+2$ are the first two exam sessions of day $(i+1)$ and constitutes the first half day, while sessions $4i+3$ and $4i+4$ are the next 2 sessions of the $(i+1)$'th day and are in the second half day. The constraints which must be taken into account by the scheduling algorithm are described by Fang as:

STRONG CONSTRAINTS:
[1] The same student cannot take two different exams at the same time. In other words, the exams cannot clash for any student. (30)
[2] Very strongly prefer no more than 2 exams per day for a given student. (10)

WEAK CONSTRAINTS:
[3] Strongly prefer not to have exams consecutive on the same half day for the same student. (3)
[4] Prefer not to have exams consecutive on different half days of the same day for the same student. (1)

In order to model these constraints as an optimization problem, a penalty is given to each instance of a violation of a constraint in a timetable being evaluated. Given a timetable, its cost is calculated as follows: For each occurrence of a violation of one of the constraints above, the number in brackets next to it is added to the total cost of that timetable. To do this a list of courses taken by each student is examined and is used to check which constraints have been violated for each course combination taken.

Fang (1992) then compares the performance of several GAs, having different parameter settings and operators, with the timetable generated manually by human experts. Each GA is run 10 times to generate 10 different timetables. The GA that performed the best produced a six day timetable with cost 45 as the best of 10 runs, compared to the timetable generated manually by a human expert which had a cost of 101 in spite of using *seven* days. In addition, the 10 runs of the GA takes about 20-30 minutes while the human expert usually takes about an hour to construct a first draft timetable followed by consultation with the students to evaluate the timetable, and then modifications are made to the first draft.

According to Fang (1992), the GA method is clearly superior to the manual method since it is faster and constructs better timetables. The question we address next is how tabu search and simulated annealing compares with the genetic algorithm approach for this specific timetabling problem. We start by describing simulated annealing and tabu search and how we applied it to this exam timetabling problem.

Tabu Search and Simulated Annealing

Both TS and SA are based on an operations research technique known as local optimization that is often used to solve combinatorial optimization problems (Eiselt and Laporte, 1987). In order to use any of these techniques to find an approximate solution to some instance of a minimization problem, we need to define the set X of feasible solutions and the objective function $c : X \rightarrow \Re$ that we are trying to minimize. That is, we are trying to find a solution S_{best} which $c(S_{best})$ is as close to the minimum of c

over X as we can find with the heuristic method in limited time. In addition, we need to define a neighbourhood structure for the set of feasible solutions. We do this by defining the function $N: X \rightarrow 2^X$ such that for each $S \in X, N(S) \subseteq X$ is the set of solutions defined to be adjacent to S, also called the neighbourhood of S. Note that we are in effect defining a directed neighbourhood graph G = (X,E) where the set of vertices is the set of solutions X and the set of edges $E = \{(S,S') \mid S \in X \wedge S' \in N(S)\}$.

A local optimization algorithm starts with some initial solution $S = S_{init}$, $S_{init} \in X$ and searches $N(S)$ for a solution $S' \in N(S)$, such that $c(S') < c(S)$. Then it assigns $S = S'$ and repeats the process, terminating when a solution $S_{loc} \in X$ is reached for which $c(S_{loc}) \leq c(S')$, $\forall S' \in N(S_{loc})$. Such a solution is known as a local minimum. The fact $N(S_{loc})$ does not contain a better solution does not imply that S_{loc} is a global minimum — a solution for which c is minimized over the entire solution space X.

Both TS and SA also move through the neighbourhood graph defined on the search space X by moving from one solution to a next along the edges of the neighbourhood graph, keeping track of the best solution encountered up to that point in an attempt to find a solution with low cost. They differ, however, in the method used to select the next solution from the neighbourhood of the current one. They also differ from local optimization because they do not get trapped in local minima and do not have to terminate when a local minimum is reached.

Simulated Annealing

Simulated annealing is a process that has its roots in statistical mechanics (see Davis, 1987b). It was first proposed as a possible technique for finding near global minimum solutions to large scale optimization problems by Kirkpatrick, Gelatt and Vecchi (1983). Since then SA has been successfully applied to a number of such problems, for example the travelling salesman problem (Kirkpatrick, Gelatt and Vecchi, 1983) and graph colouring (Chams, Hertz and de Werra 1987 and Johnson, Aragon, McGeoch and Schevon, (1991).

SA is based on local optimization with one important difference: When the current solution is S and a solution $S' \in N(S)$ is randomly selected with $c(S') \leq c(S)$ that solution is immediately accepted and made the current solution. If, however, a solution $S' \in N(S)$ is generated with $c(S') > c(S)$, it has a probability $e^{-(c(S')-c(S))/T}$ of being accepted. Each selection of a solution from $N(S)$ is called a trial, irrespective of whether that solution is accepted. T in the above expression is a variable of the algorithm known as the temperature. The temperature (a term that comes from the statistical mechanics analogy) is initialized to some large value at the start of the SA process and is decreased gradually as the search progresses. This will make it increasingly less likely that solutions with higher costs than that of the current one will be accepted. It is known that, in theory, under certain conditions and when the temperature is decreased sufficiently slowly, the SA process will eventually find a global minimum for many classes of problems (see Faigle and Kern, 1991). However little is known about how to implement the SA process to find good, near optimal solutions to practical problems quickly (see Johnson, Aragon, McGeoch and Schevon, 1991).

To apply the general SA algorithm to any problem, the following parameters have to be specified: The initial temperature T_{init}, together with the temperature factor $a \in (0,1)$ by which the temperature is multiplied every *rep* trials to simulate the gradual lowering of the temperature. We also have to specify the terminating condition for the search. These issues will be discussed at the end of section 3.

Tabu Search

Tabu Search is another stochastic technique based on local optimization that has had some success with combinatorial optimization problems, including graph colouring (Hertz and de Werra, 1987) and also timetabling (Hertz, 1991 and 1992). It was originally proposed by Glover for a specific application and was later generalized (see Glover 1986). It seems less popular than SA, however, and little has been published on the theoretical aspects of TS.

TS operates in a fashion similar to SA and local optimization, traversing the neighbourhood graph by moving from one solution to the next along the edges of the graph, but differs in how, at each step of this iterated process, the next solution to visit is selected from the neighbourhood of the current solution. If the current solution is S, TS generates a random sample of solutions $V^* \subseteq N(S)$ with size $|V^*| = sample\_size$, where $sample\_size$ is a parameter of the TS algorithm. If a solution v_i is found while constructing V^*, such that $c(v_i) < c(S)$, then v_i is immediately accepted as the new current solution and the process is repeated. If, however we generate the entire sample consisting of $sample\_size$ solutions without finding a better solution, the candidate in V^* with the lowest cost is accepted. This process allows TS to escape from local minima where traditional local minimization algorithms would have had to terminate.

In order to prevent TS from cycling indefinitely between a small number of solutions or vertices in the neighbourhood graph, a structure known as a tabu list is employed. When TS moves from the current solution S to a new current solution S', the reverse of the move that transforms S into S' is added to the tabu list and that move is said to be tabu. In general, the exact nature of a move in this sense depends on the application. For assignment problems a move usually consists of changing the value assigned to one of the variables of the problem. For example if we transform the current solution by only changing the value assigned to variable v_i from c_1 to c_2, then the reverse of this move is taken to be any transformation that reassigns value c_1 to variable v_1. In such a case we normally store the pair (i,c_1) in the tabu list to indicate that assigning c_1 to variable v_1 is a tabu move. Whenever a move is made its reverse is added to the tabu list and moves that are tabu are prohibited from being made under a number of conditions we will explain next. Moves do not stay tabu forever, instead only the reverses of the last *T_size* (a parameter of the TS algorithm) moves made are recorded in the tabu list and the tabu status of older moves are dropped when newer ones are added. The tabu list prevents TS from returning to a solution it has already visited during the previous *T_size* moves. However, it also makes moves to solutions not yet visited tabu when we define moves as we have done here. It could happen that a solution with a cost lower than the best solution found up to that point can be reached from the current solution, but that that solution is tabu. In order to reduce the risk of making solutions tabu that are worth investigating, Glover (1986) proposed the use of an aspiration function.

The aspiration function A initialized as $A(c) = c, \forall$ costs c when TS starts. Whenever we move from a solution S_1 with cost c_1=$c(S_1)$ to a solution S_2 with cost $c_2 = c(S_2) < A(c_1)$, we set $A(c_1) = c_2$. The interpretation of the aspiration function is that when $A(x) = y$, then the lowest cost of a solution ever moved to from any solution with cost x is y. Thus whenever we can move from a solution S_1 to S_2 and $c(s_2) < A(c(S_1))$, we know that this transition has not been made earlier during the search. With this in mind, we can safely ignore the tabu status of a move that improves on the aspiration value of the cost of the current solution, knowing that this transition has not been made previously and that this is consistent with the goal of preventing cycling.

Applying TS and SA to the Edinburgh problem.

The above two general algorithms, SA and TS, were applied to the Edinburgh AI/CS M.Sc. 91/92 exam problem as described by Fang. This requires that the exams be scheduled in 6 days subject to the constraints given in section 2. In order to do this, a neighbourhood structure has to be defined for SA and TS, terminating conditions defined and values assigned to the parameters of the two algorithms.

The neighbourhood structure

Both SA and TS require that a solution space, neighbourhood structure and cost function for the problem be specified. The solution space, X, and cost function, c, we take to be equivalent to Fang's (1992) definition as described in section 2. The solution space is defined to consist of all assignments of the $4N$ exam sessions available to the K exams. For this particular problem, $K = 44$ and $N = 6$. The cost function we also take to be identical to that of Fang that we described in section 2. This allows us to compare the costs of solutions found by TS and SA to those found by Fang's GAs.

Unlike the GA approach, SA and TS need a neighbourhood structure imposed on the solution space. We use the same structure for both SA and TS: For any solution $S \in X$ we define $N(S)$ to consist of all assignments in X that differ from S by the session assigned to exactly one exam.

Another aspect to consider is the method used in both SA and TS to generate an initial solution. A number of possibilities are available. One is to use for an initial solution one that was found by some other heuristic algorithm (if such an algorithm was available) and the use SA or TS to attempt to improve on that. We choose the simplest method for generating initial solutions, since it was sufficient for our purposes: The initial solution was taken to be some randomly generated solution from the solution space.

The terminating condition

For both TS and SA we need to define terminating conditions for the search. In order to compare our approach with that of Fang we need some condition that will ensure that the amount of work done in that case of SA and TS is approximately equal to that done by the GAs. Ideally we want to restrict running times to be the same for all 3 methods, but this would depend on the relative speed of the system on which Fang implemented his GAs. Fortunately there is a good alternative. By far the most computationally expensive operation for all 3 these techniques is evaluating the cost of a solution by iterating through the list of students to determine for each student the penalty due to violating the constraints listed in section 2. Thus to ensure that one run of our implementations of SA and TS does approximately the same amount of work as one run of the GA that we are comparing it with, we terminate the search when 15000 solutions have been evaluated -- the same number that one run of the GA evaluates. Note that 15000 is a very small fraction of the approximately $K^{4N} \simeq 2.7 \times 10^{39}$ solutions in the search space.

Parameter tuning

Next we address the question of assigning good values to the parameters of the SA and TS algorithms. Deciding on parameter values is a tricky business. Often the only way to find parameter values is by using a combination of extensive experimentation, intuition and educated guesswork. This approach was used to find the optimal parameter settings for SA and TS too. The parameters for TS were the easiest to decide on. There are only 2: *T_size* and *sample_size*. Glover (1986) noted that the optimum value for *T_size*, the size of the tabu list, appears to be approximately 7 and that this seems to be largely problem independent. We found that smaller values for *T_size* tends to deteriorate the quality of the best solution found in 15000 evaluations, while much larger values have very little effect in terms of the quality of solutions, but that larger tabu lists slow down the search slightly. The best value for *sample_size* was found through experimenting with a number of different values to be 50. This seems to confirm the assertion by Hertz (1992) that a good value for *T_size* when using TS for a scheduling problem is to take *T_size*= number of objects to schedule. Remember that in our problem that the number of exams to be scheduled =K=44. After some experiments with TS seemed to confirm this, we decided that *T_size*=7 and *sample_size*=50 were good parameter settings. (It is interesting to note that Fang (1992) found that with GAs the best *population size* was also about 50).

The 3 parameters for SA were slightly more difficult to decide on. The parameters we needed to find values for are: T_{init}, *a* and *rep*. The parameters *a* and *rep* together determine the rate at which the initial temperature T_{init} is reduced. Unlike TS there are no simple general guidelines for determining parameter settings for SA. After extensive experimentation, good values for these parameters were found to be:
T_{init} =2, *rep*=2000 and *a*=0.8.

Results

Now we report on the performance of TS and SA. Remember that Fang's (1992) best GA found a solution with cost =45 in one of 10 runs with each run evaluating 15000 solutions. Also note that we are restricting SA and TS also to 15000 evaluations and running each algorithm 10 times with the parameter settings we indicated in the previous section.

Table 1. TS, SA and GA performance on 10 runs.

Method	#Evaluations	5000	10000	15000
SA	Best	43.0	40.0	31.0
	Avg.	49.1	47.9	46.6
TS	Best	37.0	37.0	37.0
	Avg.	58.5	56.3	55.1
GA	Best	-	-	45.0

The above table shows for SA and TS the best solution found during 10 runs after 5000, 10000 and 15000 evaluations and also the average cost of the best solution found by each of the ten runs after 5000, 10000, and 15000 evaluations. For Fang's best GA, only the best solution found after 15000 evaluations is known.

Conclusions

From table 1 we can see that both TS and SA found solutions significantly better than those reported by Fang. Even after 5000 iterations both SA and TS in one of the 10 runs of each found solutions better than that reported for the GA after 10 runs of 15000 evaluations. The best run of TS found a solution with cost 37 after only 3947 evaluations without improving it further after that. This might lead one to think that TS converges to a good solution more quickly than SA, but the gradual decrease in the average best solution found by each of the 10 runs of TS would seem to refute this when we compare it to the average values for SA.

Armed with table 1 one might be tempted to conclude that SA is better than TS and that TS is better than GAs. This would be a hasty assumption. Firstly these results apply to only one very specific problem. It is entirely possible that with slight changes to the problem the situation might be different. On the other hand it is reasonable to expect that the greater the similarity a problem has to the one we experimented with, the more relevant our results will be. There are also very many variations of GAs and it may be that other variations may perform better on this problem. The results reported on here provide no evidence of this, however.

SA and TS are obviously worth considering for timetabling problems similar to the one we examined. SA, of course, is good because it produced the best results. But in order for SA to perform well careful selection of the parameters is necessary. If we make bad choices for the parameters SA will not perform very well. In addition, the parameter values for SA tend to be rather problem dependent and we might have to reconsider parameter choices when the problem changes slightly. In contrast to this, TS would be a good method to use if we do not want to spend time optimizing parameters manually. The two parameters of TS are largely problem independent and if we choose *T_size*=7 and *sample_size*= number of exams to schedule we are likely to achieve good results.

SA and TS were also applied to the UPE timetabling problem, with similar results. The algorithms discovered timetables considerably better than the manually generated timetable. While these are only two specific cases, the fact that such good results were obtained with general problem solving algorithms should be encouraging to others who wish to apply SA and TS to exam timetabling.

References

Carter, M.W. (1986). ``A survey of practical applications of examination timetabling algorithms", Operations Research, Vol. 34, pp. 193-202.

Chams, M., Hertz, A. and Werra, D. de (1987). ``Some experiments with simulated annealing for coloring graphs", European Journal of Operational Research, Vol. 32, pp. 260- 266.

Davis, L. (ed.) (1987a). ``Genetic Algorithms and Simulated Annealing", California: Morgan Kaufmann.

Davis, L. (1987b). ``Genetic Algorithms and Simulated Annealing: An Overview", in Davis (1987a), pp. 1-11.

Davis, L. (ed.) (1991). ``Handbook of Genetic Algorithms", New York: Van Nostrand Reinhold, 1991.

Eiselt, A.E. and Laporte, G. (1987). ``Combinatorial Optimization Problems with Soft and Hard Requirements", Journal of the Operations Research Society, Vol. 38, pp. 785- 795.

Faigle, U. and Kern, W. (1991). ``Note on the Convergence of Simulated Annealing Algorithms", SIAM Journal of Control and Optimization, Vol. 29, pp. 153-159.

Fang, H. L. (1992). ``Investigating Genetic Algorithms for scheduling", MSc Dissertation, University of Edinburgh Dept. of Artificial Intelligence, Edinburgh, UK.

Glover, F. (1986). ``Future Paths for Integer Programming and Links to Artificial Intelligence", Computers and Operations Research, Vol. 13, pp. 533-549.

Goldberg, D.E. (1989). ``Genetic Algorithms in Search, Optimization \& Machine Learning", Reading: Addison Wesley, 1989.

Hertz A. and Werra, D. de (1987). ``Using Tabu Search Techniques for Graph Colouring", Computing, Vol. 39, pp. 345- 351.

Hertz, A. (1991). ``Tabu Search for Large Sc le Timetabling Problems", European Journal of Operational Research, Vol. 54, pp. 39-47.

Hertz, A. (1992). ``Finding a Feasible Cours chedule using Tabu Search", Discrete Applied Mathematics, Vol. 35, pp. 255-270.

Johnson, D. (1990). ``Timetabling University Examinations", Journal of the Operations Research Society, Vol. 41, pp. 39-47.

Johnson, S.J., Aragon, C.R., McGeoch, L.A. and Schevon C. (1991). ``Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Colouring and Number Partitioning", Operations Research, Vol. 39, pp. 378-406.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983).``Optimization by Simulated Annealing", Science, Vol. 220, pp. 671-680.

Robertson, G. (1987) ``Parallel Implementation of Genetic Algorithms in a Classifier System", in Davis (1987a), pp. 129-140.

Schmidt, G. and Ströhlein, T. (1979). ``Timetable construction - an annotated bibliography", The Computer Journal, Vol. 23, pp. 307-316.

Werra, D. de (1985). ``An introduction to timetabling", European Journal of Operational Research, Vol. 19, pp. 151-162.