NPAC technical report SCCS-781

Academic Scheduling using Simulated Annealing with a Rule-Based System

Saleh Elmohamed, Paul D. Coddington and Geoffrey C. Fox

Northeast Parallel Architectures Center
Syracuse University, Syracuse, NY 13244

{saleh, paulc, gcf}@npac.syr.edu

ABSTRACT

In this study we tackle the NP-hard problem of academic class scheduling, or timetabling, at the university level. We have used simulated annealing, with three different cooling schedules being implemented and compared. A rule-based expert system is used as a preprocessor to provide a good starting point for the annealing, which greatly improves the results. A variant of the rule-based system is also used to select the trial schedules in the annealing procedure, which gives a much higher acceptance of the moves. This approach provides a combination of a good starting point, a good cooling schedule, and a good choice of trial moves for the simulated annealing, which together enable us to obtain valid schedules for the timetabling problem for a large university using a complex cost function that includes student preferences.

Keywords : scheduling, timetabling, optimization, simulated annealing, reheating, annealing schedule, expert system.

1 Introduction

The primary objective of this study is to derive an approximate solution to the problem of university class scheduling, or timetabling, which can be summarized as follows: given data sets of classes and their days, enrollments, and instructors; rooms and their capacities, types, and locations; distances between buildings; priorities of each building for different departments; and students and their class preferences; the problem is to construct a feasible class schedule satisfying all the hard constraints and minimizing the medium and soft constraints. Hard constraints are space and time constraints that must be satisfied, such as scheduling only one class at a time for any teacher, student, or classroom. Medium and soft constraints are student and teacher preferences that should be satisfied if possible.

The timetabling problem is a high-dimensional, non-Euclidean, multi-constraint combinatorial optimization problem, and is consequently very difficult to solve. This problem has been tackled by many researchers, mostly in the field of operations research. A number of different heuristics have been tried on different instances of the problem, from high school to university course scheduling (see the reviews by de Werra [9] and Shaerf [43]). For small to medium size problems, some of these methods work well, however no particular method has yet been shown to produce good results for large-scale, real-world problems. Very few of these studies have used real-world data, and none of these methods has been used on as large and complex a problem as we have addressed in this work, which also takes into account constraints due to student preferences.

We have used data for classes at Syracuse University. Currently this problem is handled by the university scheduling department in a semi-automated fashion. A scheduling program is used to find a partial solution, and substantial manual effort is required to iterate towards a final solution. Also, when scheduling a certain semester (e.g. fall 1996), a template of a previous semester (e.g. fall 1995) is used as part of the input data.

We tackle this problem using a multi-phase approach. First, a rule-based expert system is used as a preprocessor to derive a partial solution. This is used as the starting point in the second phase, which uses a simulated annealing algorithm, for which we have tested three different cooling schedules: standard geometric cooling, adaptive cooling, and reheating as a function of cost. Using a preprocessor in conjunction with simulated annealing was found to greatly improve the quality of the solution compared to a rule-based system and simulated annealing used separately. A variant of the rule-based system is also used to select the trial schedules in the annealing procedure, which gives a much higher acceptance of the moves. Combining these two approaches of simulated annealing and a rule-based system, and using the best cooling schedule, we were able to find feasible solutions to the complex class scheduling problem for a large university. Prior to this work, we also experimented with using the mean field annealing of Peterson et al. [41], but this gave relatively poor results [16, 17].

This paper is organized as follows. In section 2 we give a brief review of the timetabling problem, and present the three categories of constraints along with examples of each. Section 3 provides a background to some of the approaches used for tackling the timetabling problem. Sections 4 and 5 describe the two methods, a rule-based expert system and simulated annealing, that we have used in our approach. A discussion of the experimental results of our method appears in section 6, and finally a summary and conclusion are presented in section 7.

2 The Timetabling Problem

Many combinatorial optimization problems, such as the well-known traveling salesman problem (TSP), are defined in terms of minimizing an objective function. Another class of optimization problems are constraint satisfaction problems, which do not have a well-defined objective function. This includes the NP-complete *teachers and classes* timetabling problem [3, 12, 20]. Briefly, the general problem is stated as follows:

> For a certain school with N_p teachers, N_q classes, N_x classrooms and lecture halls, and N_s students, it is required to schedule N_l teacher-class pairs within a time limit of N_t time slots producing a legal schedule. A legal schedule needs to be found such that no teacher, class, or student is in more than one place at a time, and no room is expected to accommodate more than one lesson at a time or more students than its capacity.

Figure 1 sketches the approximate relationship between the various entities of the timetabling problem.

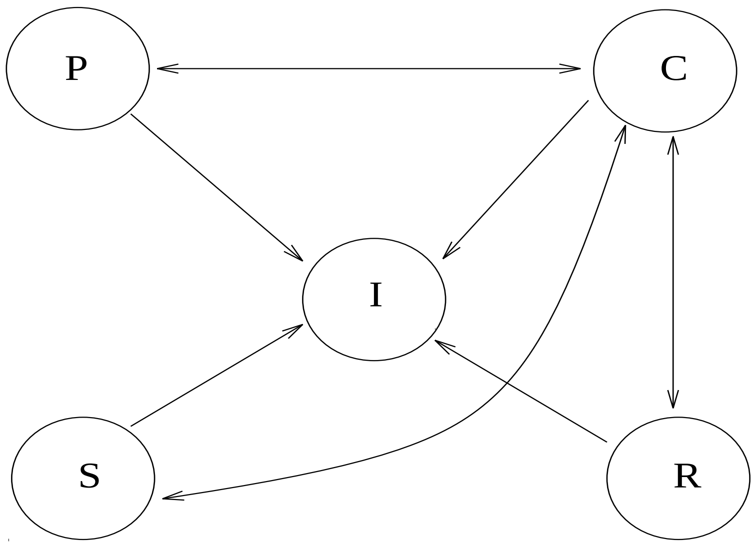

Figure 1: The relationship between various entities of the timetabling problem, where P is professors, C is classes, S is students, R is rooms, and I is time periods. The direction of the arrow indicates the direction of the assignment.

Constraint satisfaction problems have have been widely studied in the artificial intelligence community [38, 33, 36]. A solution to a given constraint satisfaction problem is an assignment of values to the variables that is consistent with all of the constraints. Deciding whether or not a given problem instance has a solution is NP-complete in general [36].

2.1 Constraints of the Timetabling Problem

Timetabling is the assignment of time slots to a set of events, subject to constraints on these assignments. Generally, a constraint places a restriction on when and where classes may be scheduled. The constraints for this problem can be hard, medium or soft. The medium and soft constraints have an associated cost (or penalty), and if they are not satisfied, the goal is to minimize this cost. The hard constraints must be satisfied, so their associated cost must be reduced to zero. A feasible schedule is one that satisfies all the hard constraints.

In short, the three types of constraints used are:

1. Hard - These are constraints that must be satisfied.
2. Medium - These are constraints that should be satisfied, if possible, but are not required for a valid solution or a legal schedule.
3. Soft - These constraints have a lower priority (and thus lower cost) but it is preferable to minimize this cost.

2.1.1 Hard Constraints

The most common and important hard constraint is what is sometimes referred to as an *edge* constraint between two events, which simply states that a given pair of events (e_1, e_2) must not overlap in time. The name is due to the similarity of any simple timetabling problem with vertex coloring in graph theory [9, 10].

Events that must not overlap in time include:

- classes taught by the same professor,
- classes held in the same room,
- a class and a recitation or a lab of the same class.

Other examples are space or room constraints:

- Some classes, such as laboratories, require a certain type of room.
- A class cannot be assigned to a particular room unless the capacity of the room R_c is greater than or equal to the class enrollment C_e. The smaller the difference between these two sizes, the better the assignment.

A cost is associated with any violation of these constraints. The room constraint is implemented using a non-linear function for the cost C_{rs} associated with the room size:

$$C_{rs} = \begin{cases} C_e - R_c & \text{if } C_e > R_c\,, \\ \tanh((1.0 - \frac{C_e}{R_c} - \beta)\gamma\pi)(R_c - C_e) & \text{otherwise.} \end{cases} \tag{1}$$

In our experiments, the parameters β and γ were set to 0.5 and 0.8, respectively.

There is also a cost assigned to any violation to the constraint that certain classes cannot be scheduled at the same time or have any overlap in the scheduled times. Here is how the exclusion cost C_x is obtained from this constraint:

$\forall i, j$ such that $i \neq j$, let c_i denote class(i), $s(c_i)$ the starting time of c_i, $end(c_i)$ the end time of c_i, $len(c_i)$ the duration of c_i, ϵ is some chosen constant, and $Rlen(c_i, c_j)$ is the relative length between c_i and c_j as defined below.

$$C_x = \begin{cases} 0.0 & \text{if } (s(c_i) > s(c_j)) \text{ and } (end(c_i) + \epsilon \geq s(c_j)), \\ 0.0 & \text{if } (s(c_j) > s(c_i)) \text{ and } (end(c_j) + \epsilon \geq s(c_i)), \\ (end(c_i) - s(c_j))Rlen(c_i, c_j) & \text{if } s(c_i) > s(c_j), \\ (end(c_j) - s(c_i))Rlen(c_i, c_j) & \text{if } s(c_j) > s(c_i). \end{cases} \tag{2}$$

$$Rlen(c_i, c_j) = \begin{cases} len(c_i)/len(c_j) & \text{if } len(c_i) \geq len(c_j), \\ len(c_j)/len(c_i) & \text{otherwise.} \end{cases} \tag{3}$$

2.1.2 Medium Constraints

Medium constraints are usually considered to be those constraints that fall into the gray area between the hard and soft constraints [15]. In our implementation, we define medium constraints to be constraints such as time and space conflicts which, like hard constraints, cannot physically be violated (for example, it is not possible for a student to be in two different classes at the same time). However we consider these constraints to be medium rather than hard if it is possible to relax the constraints by by making adjustments to the specification of the problem. The primary example is student preferences. We cannot expect to be able to satisfy all student class preferences — in some cases, certain students will have to adjust their preferences or postpone taking a particular class since some classes will have time conflicts, or will be oversubscribed. In the problem we have studied, students specify their class preferences, plus a number of alternate choices. Our program attempts to satisfy a student's main preferences, but in many cases it will have to choose from the list of alternates, and in some cases will not be able to schedule the number of classes requested, even using the list of alternates.

Medium constraints have a high penalty attached to them, although not as high as that associated with the hard constraints. In the final schedule the penalty of these constraints should be minimized and preferably reduced to zero. Some examples of medium constraints are:

- Avoid time conflicts for classes with students in common.
- To sign up for a class it must be open (classes can open or close depending on the class size).
- Eligibility criteria for the class must be met.
- Enroll students in all parts of a class when associated sections (i.e. recitation, labs) are required.
- Do not enroll athletes in classes that conflict with their sport practice time (of course, depending on the sport).

2.1.3 Soft Constraints

Soft constraints are preferences that do not deal with time conflicts, and have a lower penalty (or cost) associated with them. We aim to minimize the cost, but do not expect to be able to reduce it to zero. Some examples are:

- Lunch and other break times may be specified.

- Professors may request periods in which their classes are not taught.
- Professors may have preferences for specific rooms or types of rooms.
- Balance or spread out the lectures over the week.
- For each student, balance the three-day (*Mon, Wed, Fri*) as well as the two-day (*Tue, Thu*) schedules.
- Classes may request contiguous time slots.
- Balance enrollment in multi-section classes.
- Minimize the distance between the room where the class is assigned and the building housing its home department.

Some soft constraints may have higher priority (and thus higher cost) than others. For example, preferences involving teachers will have higher priority than the preferences of students.

We have dealt with the distance minimization constraint in the following way. Given the various building preferences of the departments, we have constructed a matrix M_{IJ} between all the academic departments I and all the buildings involved in scheduling J. Using M_{IJ} in conjunction with the (appropriately scaled) distance between all buildings involved in the process, a final distance matrix Q is derived and directly used in the scheduling process. Also, unless otherwise stated, a department's home building is always the first preference for the department classes to be assigned.

Let B denote the set of all k buildings, D denote the set of all n departments. Also for department d_i where $1 \leq i \leq n$ the distance to all buildings B is a vector denoted by $< d_i, b_j >$ where $b_j \in B$ and $1 \leq j \leq k$. At each step of the scheduling process and according to d_i space preferences, d_i courses would be scheduled into buildings $B^{'}$ where $|B^{'}| \leq |B|$ and the distance is denoted by the vector $\ll d_i, b_j \gg$ where $b_j \in B^{'}$. In addition, let $|B^{'}| = k^{'}$.

Now the cost associated with the distance of department d_i is computed as follows:

$$C_{d_i}^{dist} = \frac{\sum_{j=1}^{k^{'}} \ll d_i, b_j \gg}{\sum_{j=1}^{k} < d_i, b_j >} \; k^{'} \tag{4}$$

Note that throughout the scheduling process, the denominator of equation 4 stays fixed while the numerator may change.

2.2 Cost Function

The cost function measures the quality of the current schedule and generally involves the weighted sum of penalties associated with different types of constraint violations. Summing these penalties or costs is not strictly necessary, provided that the components (individual costs) can be combined in some fashion to reflect the overall quality of the schedule. The aim of the optimization technique is to minimize the cost function. We consider the following as part of the cost function:

- *Class cost* is the sum of costs incurred from scheduling a class in a particular time period, including having a certain class and its lab scheduled in the same time period; spreading classes over the week and within each of the weekdays; and gluing to fill in holes in the timetable and allow contigous time periods for a class that requests them.

- *Professor cost* is the sum of costs associated with each professor, including teaching two or more classes that overlap in time; being assigned more classes than the maximum number allowed to be taught; and not having a long enough time interval between classes taught by the same professor.
- *Student cost* is the sum of costs associated with each student, including clashes in class times; not having the classes evenly spread out over the week; having to schedule a class from the list of alternates rather than the list of first preferences; or classes that do not satisfy eligibility criteria.
- *Room cost* is the sum of costs associated with each classroom, including the distance cost obtained from the distance constraint; not being able to schedule a room of the specified type; and the room size cost obtained from the room constraint.

Each of the terms in the cost function are assigned different weights [46], in order to quantify the relative importance of the corresponding constraints. For example, hard constraints should be weighted so that their cost is much greater than the medium constraints, which should similarly be weighted to have higher costs than the soft constraints. The values of each of the terms for the constraint costs need to be taken into account when specifying the weights. For example, soft constraints involving the students require summing the cost over all students, which will usually be much larger than the equivalent sum over all professors, since the number of professors is much smaller than the number of students. However, constraints involving professors are usually given a higher cost than constraints involving students, in which case the weights for the professor costs will need to be much larger than the weights for the student costs.

What we have done is to select individual weights based on the relative importance of the various constraints of a particular type (hard, medium, and soft), and then included an additional overall weighting factor for each type of constraint, with the weighting factor for hard constraints being much larger than for medium constraints, which in turn is much larger than for soft constraints.

3 Methods for Tackling the Timetabling Problem

Timetabling belongs to the class of constraint satisfaction problems and is essentially an example of a resource-constrained scheduling problem, which is NP-hard [19, 5]. Here, resources are physical entities, for example: students, rooms, overhead projectors, and so on. Time periods can also be thought of as resources.

In general, timetabling problems are always NP-complete [18], which means there is no known algorithm that can find an optimal solution in a time that is bounded by a polynomial of the problem size. As a result, these problems are often solved by means of *heuristics* – solution procedures that focus on finding a feasible schedule of "good" (as opposed to optimal) quality within an acceptable amount of time.

Traditionally, timetabling has been approached by means of linear programming (LP) with binary variables, plus some heuristics. For example, if i identifies a teacher, j identifies a time interval and k identifies a class then the binary variable $X_{ijk} = 1$ if teacher i has class k at time-interval j; $X_{ijk} = 0$ otherwise. Suppose we have a set of data as small as 20 teachers, 30 time intervals, and 10 classes. If we use the LP approach to obtain a feasible schedule out of this data, then we need to deal with over 6000 variables to represent the problem, and 2^{6000} possible states. Clearly the problem becomes intractable even for very small numbers of variables.

3.1 Previous Approaches

Most of the previous work on academic scheduling has addressed only a subset or simplified version of the problem, such as examination scheduling, or the smaller scale (but still very difficult) problem of high school class scheduling, which does not require dealing with student preferences (for more information, see the reviews by de Werra [9] and Shaerf [43]).

Traditionally, the approach to the timetabling problem in the operations research community has been to use methods such as integer programming and network flow [9]. Another approach is to reduce it to the well-known problem of graph coloring [37, 9, 34]. More recently, a number of non-conventional global optimization techniques such as neural networks [20, 21], simulated annealing [12, 13, 2, 48, 28, 8, 14], genetic algorithms [6], and tabu search [23, 24, 7] have been used to tackle various instances of the timetabling problem, with varying degrees of success.

Simulated annealing has been successfully used for scheduling examinations [13, 28], laboratories [12], conference seminars [14] and high school classes [2, 8], but not to our knowlege for the more complex problem of university class scheduling.

3.2 Our Approach to the Timetabling Problem

The timetabling problem we have studied can be summarized as follows. Given data sets of classes and their days, hours, enrollments, and instructors; rooms and their capacities, types, and locations; distances between buildings; priorities of each building for different departments; and students and their class preferences; the problem is to construct a feasible academic class schedule satisfying all the hard constraints and minimizing the medium and soft constraints.

If the procedure used cannot assign certain classes to rooms and timeslots, i.e. cannot find a feasible schedule for all classes, these classes are left out of the final schedule and must be assigned manually, or else the input variables must be changed. In some cases there may be no feasible schedule for the given constraints, and the problem must be reformulated to allow a solution. However our goal is to provide a feasible schedule of better quality (i.e. lower cost), and in less time, than that provided using current techniques.

We have adopted a multi-phase approach to tackle the the timetabling problem [43, 46]. In the first phase a partial solution (a feasible schedule for a subset of the classes) is found using a rule-based expert system. In the second phase this partial schedule is improved to obtain a final schedule, using simulated annealing. Simulated annealing is a very time-consuming, computationally intensive procedure, so it is a good practice to start at a good point in the search space or with a partial solution. A good starting point allows the annealing to begin at a lower temperature, provides faster convergence, and usually a better final solution. Using a rule-based preprocessor is a way of quickly providing such a starting point for the annealing process.

Our multi-phase approach is similar to that of Thomson and Dowsland [46] to the problem of exam scheduling, in which they use a rule-based system to solve the hard constraints, and then use simulated annealing to minimize the cost of the soft constraints, while continuing to satisfy the hard constraints. Solving the hard constraints is much more difficult for the course scheduling problem, and we are only able to obtain a partial solution using the rule-based system in the initial phase (the preprocessor). We have also used a variant of the rule-based system to choose the new trial states for the annealing algorithm. This rule-based system only selects trial states that satisfy the hard constraints, while the annealing algorithm takes care of the optimization of the medium and soft constraints. In effect, we are continuing the multi-phase approach at each iteration of the annealing algorithm.

To use simulated annealing effectively, it is crucial to use a good cooling schedule, to allow the temperature to be reduced as quickly as possible while keeping the system close to equilibrium, and also to use a good method for choosing new trial states, in order to efficiently sample the search space. We have experimented with different approaches in both these areas, and these are discussed in section 4.1.

4 The Rule-Based System

We have implemented a fairly complex rule-based expert system for solving the timetabling problem, for three reasons. Firstly, it gives us a benchmark as to how well simulated annealing does in comparison to this standard technique. Secondly, a modified version of this system us used to provide sensible choices for moves in the annealing algorithm, rather than choosing the moves at random. Thirdly, we have used this system as a preprocessor to provide a good starting point for the simulated annealing algorithm.

The rule-based expert system consists of a number of rules (or heuristics) and conventional recursion to assist in carrying out class assignments. We have developed this system specifically for the problem of academic scheduling. The basic data structures or components of the system are:

1. Distance matrix of values between each academic department and every other building under use for scheduling.

2. Class data structure of each class scheduled anywhere in campus. These structures are capable of linking with each other.

3. Room data structure of each room (regardless of type) involved in the scheduling process. Like classes, room structures are also linked with each other.

4. Data structures for time periods to keep track of which hour or time slot was occupied and which was not.

5. Department inclusion data structure giving department inclusion within other larger departments or colleges.

6. Students structures indicating classes of various degree of requirements and preferences for each student.

The basic function of the system is as follows: given data files of classes, rooms and buildings, department-to-building distance matrix, student data, and the inclusion data, using the abovementioned data structures, the system builds an internal database which in turn is used in carrying out the scheduling process. This process involves a number of essential sub-processes such as checking the distance given in the distance matrix, checking building, room type and hours occupied, checking and comparing time slots for any conflicts, checking rooms for any space conflict, and keeping track of and updating the hours already scheduled.

The rule-based system uses an iterative approach. The basic procedure for each iteration is as follows. The scheduling of classes is done by department, so each iteration consists of a loop over all departments. The departments are chosen in order of size, with those having the most classes being scheduled first. The system first loops over all the currently unscheduled classes, and attempts to assign them to the first unoccupied room and timeslot that satisfies all the rules governing the constraints. The classes are chosen in a specific order. We chose to separate the updates into independent sets based on room type, so for example, laboratories are scheduled separately from lectures. We carry out the scheduling of lectures first, followed by scheduling of laboratories, making sure that during the course of this process no lecture and its associated laboratory are scheduled in the same time period. Also, since constraints involving capacity of rooms are very difficult to satisy, larger classes are scheduled first, to try to avoid not having large enough rooms later for those class sections with large enrollments.

In some cases the only rooms and timeslots that satisfy all the rules will already by occupied by previously scheduled classes. In that case, the system attempts to move one of these classes into a free room and timeslot, to allow the unscheduled class to be scheduled.

Next, the system searches through all the scheduled classes, and selects those that have a high cost, by checking the medium and soft constraints such as how closely the room size matches the class size, how many students have time conflicts, whether the class is in a preferred time period or a preferred building, and so on. Selecting threshold values for defining what is considered a "high" cost in each case is a subjective procedure, but it is straightforward to choose reasonable values. When a poorly scheduled class is identified, the system searches for a class to swap it with, so that the hard constraints are still satisfied, but the overall cost of the medium and soft constraints is reduced.

Our strategy is to first try to swap a class with another class in the same department. If this cannot be achieved within the rules, we attempt to swap classes within different departments in the same school, and finally we try swaps between classes in different schools. We refer to the first two types of swaps as local, while the third is considered global. Local swaps have the advantage that classes in the same departments or schools are associated with the same subset of rooms that will usually satisfy constraints involving distance and department preference, however global swaps are less likely to have students and professors in common, which may help alleviate time conflicts. A department may specify that they do not want any classes scheduled outside their home buildings, so global swaps will not be used on their classes.

This process of swapping rooms continues provided all the rules are satisfied and no "cycling" (swapping of the same classes) occurs. Once all the departments have been considered, this completes one iteration. The system continues to follow this iterative procedure until a complete iteration produces no changes to the schedule.

The rule-based system is usually unable to legally assign all of the classes, and thus produces a partial schedule as an output, which is divided into two parts: a list of scheduled classes and their assigned rooms and timeslots; and a list of classes that could not be assigned due to constraint conflicts. This is what is used as input for the simulated annealing program.

4.1 Example Rules

Here are few examples of some of the implemented rules dealing with space and hours, type of room, and priority of room.

Space and Hours

IF [room(capacity) > class(space-requested)] and [no time conflict in this room]

THEN assign the room to the class.

If more than one room having an identical availability time is present in the database, then choose the one with the minimal magnitude of the difference between room(capacity) and class(space-requested).

Type of Room

IF [room(capacity) > class(space-requested)] and [no time conflict in assigning this room] and [type(room) matches type(requested)]

THEN assign the room to the class.

If identical rooms in type are present in the database, then choose the one with the minimal magnitude of the difference between room(capacity) and class(space-requested).

Room Priority

Rooms are classified as either general purpose, which are used by all, or priority rooms with a usage based on certain priority of the class or the department, or special purpose rooms which are only assigned to specific users. Here is a sample of rules showing how these room categories are used:

IF [room(priority) == general] and [room(capacity) > class(space-requested)] and [no time conflict in assigning this room] and [type(room) matches type(requested)]

THEN assign any user to this room.

IF [room has a priority user] and [the user is already assigned] and [room(capacity) > class(space-requested)] and [no time conflict in assigning this room] and [type(room) matches type(requested)]

THEN assign other users.

IF [room(priority) == special purpose] and [user belongs to the special users of this room] and [room(capacity) > class(space-requested)] and [no time conflict in assigning this room] and [type(room) matches type(requested)]

THEN assign the room to the user.

IF [room has a priority user] and [the user is already assigned] and [room(capacity) > class(space-requested)] and [no time conflict in assigning this room] and [type(room) matches type(requested)]

THEN assign other users.

IF [room(priority) == special purpose] and [user belongs to the special users of this room] and [room(capacity) > class(space-requested)] and [no time conflict in assigning this room] and [type(room) matches type(requested)]

THEN assign the room to the user.

5 Simulated Annealing

Simulated annealing (SA) has been widely used for tackling different combinatorial optimization problems, including academic timetabling problems [12, 13, 2, 48, 28, 8, 14], The basic algorithm is described in Figure 2. The results obtained depend heavily on the cooling schedule used. We initially used the most commonly known and used schedule, which is geometric cooling, but later tried adaptive cooling, as well as the method of geometric reheating based on cost [3].

A comprehensive discussion of the theoretical and practical details of SA has been given in a number of books and review articles on the subject [1, 47, 40]. It suffices here to say that the elementary operation is the Metropolis Monte Carlo update, which involves the generation of some new candidate configuration (in this case, a schedule), which is then automatically accepted if it lowers the cost (C), or accepted with probability $\exp(-\Delta C/T)$, where T is the temperature parameter, if it would increase the cost by ΔC. In Figure 2, s is the current schedule (configuration) and $s^{'}$ is a neighboring schedule obtained from the current neighborhood space ($\mathcal{N}_s$) by assigning a class to a timeslot and room or swapping two classes in time and/or space. The annealing procedure starts at a high temperature, and then slowly decreases the temperature to zero, following a specified *cooling schedule*. Simulated annealing is essentially a generalization of a local optimization strategy, but improves upon it since at non-zero temperatures, there is a chance for the system to escape from local minima.

1. Generate an initial schedule s.
2. Set the initial best schedule $s^* = s$.
3. Compute cost of s : $C(s)$.
4. Compute initial temperature T_0.
5. Set the temperature $T = T_0$.
6. While *stop criterion* is not satisfied do:
   (a) Repeat *Markov chain length* (M) times:
      i. Select a random neighbor $s^{'}$ to the current schedule, $(s^{'} \subset \mathcal{N}_s)$.
      ii. Set $\Delta(C) = C(s^{'}) - C(s)$.
      iii. If $(\Delta(C) \leq 0)$ {downhill move}:
         - Set $s = s^{'}$.
         - If $C(s) < C(s^*)$ then set $s^* = s$.
      iv. If $(\Delta(C) > 0)$ {uphill move}:
         - Choose a random number r uniformly from $[0, 1]$.
         - If $r < e^{-\Delta(C)/T}$ then set $s = s^{'}$.
   (b) Reduce (or update) temperature T.
7. Return the schedule s^*.

Figure 2: The Simulated Annealing Algorithm

The SA algorithm has advantages and disadvantages compared to other global optimization techniques. Among its advantages are the relative ease of implementation, the applicability to virtually any combinatorial optimization problem, the ability to provide reasonably good solutions for most problems (depending on the cooling schedule and update moves used), and the ease with which it can be combined with other methods, such as expert systems and branch-and-bound techniques, forming quite useful hybrid methods for tackling a range of complex problems. Some of the drawbacks of SA are that to obtain good results the update moves and the various tunable parameters used (such as the cooling rate) need to be carefully chosen, the runs often require a great deal of computer time, and many runs may be required. Despite these problems, SA is a robust technique and has been successfully applied to a wide variety of problems. Detailed comparisons of SA with other optimization techniques for some well-known problems has shown that SA is competitive with many of the best heuristics [30].

5.1 Timetabling Using the Annealing Algorithm

The mapping of timetabling problem into the SA algorithm involves the following constructs:

1. a **state** (or configuration) is a timetable with the following sets of variables:
   - P: a set of professors.
   - C: a set of classes.
   - S: a set of students.
   - R: a set of classrooms.
   - I: a set of time intervals.
2. a **cost** (or energy) function $E(P, C, S, R, I)$, as outlined in Section 2.1.3.
3. a **move** is the assignment of a class to a time period and a room, or swapping two classes C_i and C_j, by exchanging their time periods I_i and I_j and/or classrooms R_i and R_j.

Along with all of the necessary constraints and the output of the rule-based preprocessor, the simulated annealing algorithm also takes as input data the following: a list of classes that have not been already scheduled by the preprocessor and their associated professors and room types, a list of rooms provided by the registrar's office, a department to building distance matrix, a list of students and their class preferences, and a list of classes that are not allowed to be scheduled simultaneously.

5.2 The Annealing Schedule

The effectiveness of simulated annealing depends greatly upon the rate at which the temperature is changed, which is known as the annealing (or cooling) schedule. Three annealing schedules have been used in our experiments to update the temperature of the SA algorithm: geometric cooling, adaptive cooling, and reheating as a function of cost.

The first annealing schedule we used is **geometric cooling**, where the new temperature $T^{'}$ of the SA algorithm is computed using

$$T^{'} = \alpha T \ , \qquad (5)$$

where α ($0 < \alpha < 1$) denotes the cooling factor. Typically the value of α is chosen in the range 0.90 to 0.99. This cooling schedule has the advantage of being well understood, having a solid theoretical foundation, and being the most widely used annealing schedule. The results obtained from using this standard cooling schedule can be used as a baseline for comparison with those using the other two schedules, which allow the rate of cooling to be varied.

The second annealing schedule is the method of **reheating as a function of cost** (RFC), which was used for timetabling by Abramson *et al.* [3], but the ideas behind it are due to Kirkpatrick *et al.* [31, 32] and White [49]. Before introducing this schedule we first summarize a few relevant points on the concept of specific heat (C_H), which are taken from the principles of statistical mechanics that are the foundation of the simulated annealing technique. Specific heat is a measure of the variance of the energy (or cost) values of states at a given temperature. The higher the variance, the longer it presumably takes to reach equilibrium, and so the longer one should spend at the temperature, or alternatively, the slower one should lower the temperature.

Generally, in combinatorial optimization problems, phase transitions [25, 39] can be observed as sub-parts of the problem are resolved. In some of the work dealing with the traveling salesman problem using annealing [35], the authors observe that the resolution of the overall structure of the solution occurs at high temperatures, and at low temperatures the fine details of the solution are resolved. Applying a reheating type procedure, depending on the phase, should allow the algorithm to spend more time in the low temperature phases, thus reducing the total amount of time required to solve a given problem [3].

In order to calculate the temperature at which a phase transition occurs, it is necessary to compute the specific heat of the system. A phase transition occurs at a temperature $T(C_H^{max})$ when the specific heat is maximal (C_H^{max}), and this triggers the change in the state ordering. If the best solution found to date has a high energy or cost then the super-structure may require re-arrangement. This can be done by raising the temperature to a level which is higher than the phase transition temperature $T(C_H^{max})$. Generally, the higher the current best cost, the higher the temperature which is required to escape the local minimum. To compute the maximum specific heat, we employ the following steps [3, 47, 40].

At each temperature T, the annealing algorithm generates a set of configurations $\mathcal{C}$. Let C_i denote the cost of configuration i, $C(T)$ be the average cost at temperature T, and $\sigma(T)$ be the standard deviation of the cost at T. The average cost is just:

$$C(T) = \sum_i C_i P_i(T)\,, \tag{6}$$

where $P_i(T)$ is the probability of obtaining configuration i at temperature T, and the sum is over all possible configurations. The Metropolis Monte Carlo update used in SA generates configurations with a Boltzmann probability distribution [4, 39, 40, 1, 49]:

$$P_i(T) = \frac{e^{\frac{-C_i}{T}}}{\sum_j e^{\frac{-C_j}{T}}}\,. \tag{7}$$

A particular run of the annealing program will generate (with probability $P_i(T)$) a set $\mathcal{C}$ of N_T configurations i at temperature T, from which an estimate $< C(T) >$ of the average cost can be computed using:

$$< C(T) > = \sum_{i \in \mathcal{C}} C_i / N_T\,. \tag{8}$$

Similarly, the average square cost can be computed as:

$$< C^2(T) > = \sum_{i \in \mathcal{C}} C_i^2 / N_T \,. \tag{9}$$

The variance of the cost is:

$$\sigma^2(T) \ = < C^2(T) > - < C(T) >^2 \ . \tag{10}$$

Now, the specific heat is defined as:

$$C_H(T) \ = \ \frac{\sigma^2(T)}{T^2} \,. \tag{11}$$

The temperature $T(C_H^{max})$ at which the maximum specific heat occurs, or at which the system undergoes a phase transition, can thus be found.

Reheating sets the new temperature to be

$$T = K \cdot C_b + T(C_H^{max}) \,, \tag{12}$$

where K is a tunable parameter and C_b is the current best cost. Reheating is done when the temperature drops below the phase transition (the point of maximum specific heat) and there has been no decrease in cost for a specified number of iterations, i.e. the system gets stuck in a local minimum. Reheating increases the temperature above the phase transition (see equation 12), in order to produce enough of a change in the configuration to allow it to explore other minima when the temperature is reduced again.

The third cooling schedule we have tried is **adaptive cooling**. In this case, a new temperature is computed based on the standard deviation of all costs obtained at the current temperature, i.e. the specific heat. The idea here is to keep the system close to equilibrium, by cooling slower close to the phase transition, where the specific heat is large. There are many different ways of implementing this idea, we have chosen the approach taken by Huang *et al.* [27], which was shown to yield an efficient cooling schedule. Let T_j denote the current temperature, at step j of the annealing schedule. After calculating $\sigma(T_j)$ from equation 10, the new temperature T_{j+1} is computed using:

$$T_{j+1} = T_j \cdot e^{-\frac{aT_j}{\sigma(T_j)}} \,, \tag{13}$$

where a is a tunable parameter. Following suggestions by Otten and van Ginneken [40] and Diekmann *et al.* [11], $\sigma(T_j)$ is smoothed out in order to avoid any dependencies of the temperature decrement on statistical fluctuations in the measurement of the standard deviation σ. The smoothed standard deviation $\bar{\sigma}$ is given by

$$\bar{\sigma}(T_{j+1}) = (1 - \omega)\sigma(T_{j+1}) + \omega\sigma(T_j) \,, \frac{T_{j+1}}{T_j} \tag{14}$$

where we have set ω to 0.95. This smoothing function is used because it follows (from the form of the Boltzmann distribution) that it preserves the key relationship

$$\frac{d}{dT} C(T) = \frac{\bar{\sigma}^2(T)}{T^2} = C_H \,. \tag{15}$$

Note that reheating can in principle be used in conjunction with any cooling schedule. We have used it with adaptive cooling.

5.3 The Choice of Moves

The performance of any application of simulated annealing is highly dependent on the method used to select a new trial configuration of the system for the Metropolis update. In order for the annealing algorithm to work well, it must use efficient update moves that can sample the parameter space effectively, and have a reasonably high probability of being accepted.

In our implementation, we have used a sophisticated rule-based system for choosing the moves, which is a modified version of the rule-based expert system used as the preprocessor for annealing, which is described in Section 3.2. Using the rule-based system ensures that all the trial moves satisfy the hard constraints, while the Metropolis update used in the annealing algorithm takes care of minimizing the costs of the medium and soft constraints.

One of the main modifications to the rule-based system is that while the version used in the preprocessor is completely deterministic, the version used in choosing the moves for annealing selects at random from multiple possibilities that satisfy the rules equally well. Also, some of the rules dealing with the medium and soft constraints are softened or eliminated here, since reducing the cost of these constraints is done using the annealing algorithm.

This extra freedom in choosing new schedules, plus the extra degree of randomness inherent in the annealing update, helps prevent the system getting trapped in a local minimum before it can reach a valid schedule, which is the problem with the standard deterministic rule-based system.

For class scheduling, a move involves assigning a class to an unused timeslot and room, or swapping two classes in time (timeslots) and/or space (rooms). The simplest method for doing this is to choose the classes, timeslots and rooms at random. However this is impractical, since random allocation or swapping of classes will almost always violate one of the hard constraints or greatly increase the cost of the medium and soft constraints (especially if we already have a fairly good schedule), and will thus almost always be rejected in the Metropolis procedure. This simple method is therefore extremely inefficient, since a lot of computation is required to compute the change in cost and do the Metropolis step, only to reject the move in the vast majority of cases.

What is needed is a strategy for choosing moves that are more likely to be accepted. A simple example is in the choice of room. If a new room is randomly chosen from the list of all rooms, it will most likely be rejected, since it may be too small for the class, or an auditorium when, for example, a laboratory is needed. A more effective strategy is to create a subset of all the rooms which fulfill the hard constraints on the room for that particular class, such as the size and type of room. A candidate new room for that class can then be chosen from this subset of feasible rooms, and the acceptance probability will be much higher, saving time that would otherwise be wasted on moves that would just have been rejected by the Metropolis algorithm. This is precisely what is done by our rule-based system (see Section 3.2), although rather than choosing at random from a list of feasible rooms that obey the hard constraints, it goes a step further and attempts to chose the "best" available room, for example by selecting the room that most closely matches the size requirement, and preferentially selecting rooms from the home building of the department.

Using the rule-based system to select the moves means that the classes to be moved are chosen not at random, but following the same rules used for the preprocessor, for example, unscheduled classes are handled first, and classes are selected in order based on size. Since the classes are not chosen at random, it is very unlikely that the detailed balance criterion is satisfied for the Metropolis updates, which means that the configurations produced at each temperature will not have a Boltzmann probability distribution [4, 39]. However we will assume that the actual distribution approximates a Boltzmann distribution well enough that certain standard annealing techniques that assume a Boltzmann distribution, such as adaptive cooling, will still be effective. Many of the rules used in the rule-based system address the medium and soft constraints, in an effort to reduce the overall cost. In some cases these are relaxed in choosing trial moves for annealing, in order to allow a wider selection of moves. However most of them are still included (although sometimes in modified form), since they generally improve the acceptance probability. For example, swapping a higher level class (e.g. graduate) with a lower level class (e.g. first or a second year) generally has a higher acceptance, since there is little overlap between students taking these classes. We have done a few experiments to test modifications of these "soft" rules to try to improve the acceptance and the final results. For example, we found that at high temperatures there is not much difference in the acceptance rate between local (within the same school) and global (between different schools) class swaps, and that most swaps are accepted. At low temperatures the acceptance is much lower, but about the same for local and global swaps. In the mid-range temperatures, local swaps have a slightly higher acceptance rate. Thus the rule of first attempting local class swaps, then global swaps, works well with annealing.

Generally, the rules used to choose the moves can be considered as heuristics for pruning the neighborhood or narrowing the search space, which provides much more efficient moves, as demonstrated by the good acceptance rates as shown in Figure 6.

6 Experimental Results

Our computations were done with a number of goals in mind. The main objective was to provide a schedule which satisfied all hard constraints and minimized the cost of medium and soft constraints, using real-life data sets for a large university. We also aimed to find an acceptable set of annealing parameters and move strategies for general timetabling problems of this kind, and to study the effect of using a preprocessor to provide the annealing program with a good starting point. Finally, we wanted to make a comparison of the performance of the three different cooling schedules, geometric cooling, adaptive cooling, and reheating based on cost.

6.1 The Timetabling Data Set

Our study case involved real scheduling data covering three semesters at Syracuse University. The size and type of the three-semester data set is shown in Table 1. Nine types of rooms were used: auditoriums, classrooms, computer clusters, conference rooms, seminar rooms, studios, laboratories, theaters, and unspecified types. Staff and teaching assistants are considered part of the set of professors. Third semester (summer) data was much smaller than other semesters, however there were additional space and time constraints and fewer available rooms.

Our data set is quite large in comparison to data used by other researchers. For example, high school data used by Peterson and colleagues [20, 21] consists of approximately 1000 students, 20 different possible majors, and an overall periodic school schedule (over weeks). In the case of Abramson *et al.* [2], their data set was created randomly and was relatively small, and they stated that problems involving more than 300 tuples were very difficult to solve.

Timetabling problems can be characterized by their *sparseness*. After all the required lessons have been scheduled, there will be $N_{sp} = (N_x N_t - N_l)$ spare space-time slots, and the *sparseness* *ratio* of the problem is defined as $N_{sp}/(N_x N_t)$. Dense problems have a low sparseness ratio, since they have relatively few spare space-time slots, and are thus harder to solve than sparse problems with many slots to spare. Table 2 shows the sparseness ratio (see section 3) of the data for each of the three semesters. For university scheduling, the sparseness ratio generally decreases as the data size (particularly the number of classes) increases, so the problem becomes more difficult. The first two semesters are of similar difficulty, whereas the third semester has fewer classes, and is sparser and relatively easier to solve.

Including student preferences makes the problem much harder, but these are viewed as medium constraints and thus are not necessarily satisfied in a valid solution.

6.2 Analysis of the Cooling Schedules

The three cooling schedules used each have a number of tunable parameters, such as the initial temperature, the rate of cooling and reheating, and the number of iterations at each temperature. We spent quite some time attempting to find optimal values for these parameters.

Specifying an initial temperature for annealing is usually a straightforward procedure. Infinite temperature means that the changes are always accepted (i.e. the acceptance ratio is 1), which produces random configurations. In practice the initial temperature should not be too large, or else the cooling will take too long, and the annealing will spend too much time at higher temperatures where little useful work is being done. The initial temperature is usually chosen to be the lowest possible value that still gives an acceptance ratio that is fairly close to 1. White [49] has suggested that a good value for the starting temperature is the standard deviation in the cost at infinite temperature.

Choosing an initial temperature is much more difficult when we use a preprocessor to provide a good starting configuration, rather than the usual random initial configuration. In that case, we must be very careful that the initial temperature and the acceptance ratio are not so high that the good initial configuration is randomized to the point where is loses its usefulness. However, the initial temperature must be high enough so that we can still make effective changes in the configuration. Figure 7 shows the results of various runs using different initial temperatures for geometric cooling and for adaptive cooling with reheating. In both cases there is an optimal value of the initial temperature, although using the optimal value appears to have much greater effect in the latter case. We have used this initial temperature value of $T_{max} = 497$, for all of our runs. The acceptance ratio at this temperature value is approximately 0.6.

We also need to specify a criterion for stopping the annealing procedure, when it is believed that the solution is good enough that we do not expect to obtain much improvement by further cooling. We used a standard stopping criterion [29, 30], whereby the run was terminated when no moves had been accepted for five steps (a step is a full Markov chain at a single temperature).

The goal in choosing an annealing schedule is to reduce the temperature as rapidly as possible while still keeping the system close to equilibrium at each temperature. This is easy to do at high temperatures, where the system equilibrates very rapidly, but near the phase transition the number of iterations required to equilibrate increases dramatically. For an infinite system it actually diverges at the phase transition, while for a finite system it increases as a power of the specific heat near the phase transition, and as a power of the system size at the phase transition (this is referred to as "critical slowing down") [4, 39]. Johnson *et al.* [30] noted in their SA implementation for the traveling salesman problem (TSP) that the number of iterations N_T (or the size of the Markov chain) at each temperature needed to be at least proportional to the "neighborhood" size in order to maintain a high-quality result. From our experiments we found the same to be true for the scheduling problem, even though it is very different from the TSP. In our case the neighborhood size can be taken to be the number of classes.

A number of techniques have been suggested for dynamically determining when the system has reached equilibrium at a given temperature during annealing [27, 40]. However, for large systems such as we have used, and particularly for temperatures near or below the phase transition, which are the most crucial, it is extremely difficult to determine when the system has equilibrated. We have therefore adopted an empirical approach. The rate at which the temperature is decreased, and the number of iterations at each temperature, are closely interrelated. For example, if we halve the number of iterations at each temperature value, but use twice as many temperature values, the results will be very similar. We have attempted to find values for the combination of these two variables that keep the system close to equilibrium, and produce near-optimal results.

For geometric cooling, the temperature reduction factor α is usually chosen to be in the region 0.9 to 0.99. We found only a small variation in the results for this parameter range, and decided to arbitrarily fix α to be 0.95, and use N_T as a variable with which to compare solution quality. We found good results if we set N_T to be the product of the number of classes (the system size) and the variance of the cost values.

For adaptive cooling, we set N_T to be the same as for geometric cooling, and did a number of runs with different values of the parameter a, which controls the rate of cooling (see equation 13). Slower cooling (smaller a) should give better results, but there should be a point at which further reduction of a does not markedly improve the result. Our aim is to identify this point, i.e. the largest value of a (the fastest cooling) which still gives near-optimal results. Figure 8 shows the results of these runs, indicating a plateau for a less than about 0.0001. For our adaptive cooling runs we used $a = 0.00011$.

Some time could probably be saved at high temperatures by specifying N_T using an adaptive approach based on one of the techniques for determining when the system has reached equilibrium [27, 40]. However when we use a preprocessor, we do not start at a very high temperature, so this would not be a great reduction in the overall simulation time.

The adaptive cooling and reheating methods both rely on the measurement of the specific heat, which is shown in Figure 5. There is a clear maximum, indicating a phase transition. Reheating increases the temperature so that it is above the phase transition, which gives the system a better chance to change the structure of the schedule and thus escape local minima.

If the temperature increase used in reheating is very small or very large, the reheating should have little effect, so we expect to be able to find an optimal value somewhere in between. Figure 9 shows that there is indeed a clear optimum value of the parameter K, which controls the amount of reheating (see equation 12), and we have used this value in our reheating runs.

The cost as a function of temperature is shown in Figure 3 for individual annealing runs using geometric and adaptive cooling. The results are fairly similar, apart from the expected fluctuations in the region of the phase transition (maximum of the specific heat) at $log_2(T) \approx 3.5$, but on average the adaptive cooling gives better results, and is also faster.

Figure 4 shows the cost as a function of the number of steps (temperature values), this time for annealing runs using geometric cooling and adaptive cooling with reheating. Again, the two are quite similar, until we get towards the end of the runs, at low temperature. Here the cost using geometric cooling reaches a plateau, while the cost for adaptive cooling continues to decrease slightly, until the reheating kicks in, causing the procession of spikes. Note that the reheating phase results in a further reduction in the lowest cost, without substantially more computational effort.

6.3 Scheduling Results

Our overall results are shown in Tables 3 and 4. These tables show the percentage of classes that could be scheduled in accordance with the hard constraints. In each case (apart from the expert system, which is purely deterministic), we have done 10 runs with the same parameters (just different random numbers), and the tables show the average of the 10 runs, as well as the best and worst results.

As expected, each of the methods did much better for the third (summer) semester data, which is much sparser. Our results also confirm what we expected for the different cooling schedules for simulated annealing, in that adaptive cooling performs better than geometric cooling, and reheating improves the result even further.

When a random initial configuration is used, simulated annealing performs very poorly, even worse than the expert system. However, there is a dramatic improvement in performance when a preprocessor is used to provide a good starting point for the annealing. In that case, using the best cooling schedule of adaptive cooling with reheating as a function of cost, we are able to find a feasible class schedule every time. None of the other cooling schedules were able to consistently produce feasible schedules.

Student preferences are included only as medium constraints in our implementation, meaning that these do not have to be satisfied for a valid solution, but they have a high priority. For the valid schedules we have produced, approximately 80% of the student preferences were satisfied for the first two semesters, and 94% for the third semester. This is quite a good result, particularly since other automated approaches do not deal with student preferences at all, but we are aiming to improve upon it.

6.4 The Code and Simulation Times

The programs were all written in ANSI C, and the simulations were run on a number of IBM RS/6000 workstations, and on a 12-processor IBM SP2, which uses RS/6000 processors. For the SP2, individual jobs were run separately on different processors using different sets of parameters, or different cooling schedules, or just different random numbers. Note that with multiple processors, it is possible to do many different annealing runs (using different random numbers) and then chose the result with the lowest cost. In this way, parallel processing is used not to speed up the time to obtain a result, but to obtain a better result in the same amount of time.

For the first and second semester data, the average time for running the preprocessor (the expert system) was about three hours, and the average time for each simulated annealing run was approximately 22 hours for geometric cooling and about 15 hours using adaptive cooling with reheating, which means the program could be run overnight. This is much faster than the current semi-automated procedure used at Syracuse University to produce a produce a final schedule for a semester, which involves substantial manual effort to iterate to a solution, and can take a few weeks.

7 Conclusions

We have successfully applied simulated annealing to the difficult problem of academic scheduling for a large university. Feasible schedules were obtained for real data sets, including student preferences, without requiring enormous computational effort.

We found that using a preprocessor to provide a good starting point for simulated annealing greatly improved the quality of the results. We used a fairly complex rule-based expert system for the preprocessor, however the type of preprocessor may not be crucial. Other fast heuristics could possibly be used, for example a graph coloring approach [37], and just utilizing the schedule from the same semester for the previous year should provide good results.

Another critical part of the annealing program was the choice of moves. We used techniques that drew from our work on the rule-based system in order to carefully select good moves that have a relatively high probability of acceptance, rather than wasting computational effort on randomly chosen moves, most of which would be rejected. The rule-based system ensured that all the moves satisfied the hard constraints, while the annealing process took care of minimizing the cost of the medium and soft constraints.

As expected, for the simulated annealing, adaptive cooling performed better than geometric cooling, and using reheating improved the results even further. The best results were obtained using simulated annealing with adaptive cooling and reheating as a function of cost, and with a rule-based preprocessor to provide a good initial solution. Using this method, and with careful selection of parameters and update steps, we were able to generate feasible solutions to the class scheduling problem using real data for a large university. None of the other methods were able to provide a complete feasible solution.

Our main conclusion from this work is that simulated annealing, in combination with a rule-based system, can be used to solve the complex class scheduling problem at a large university, including student preferences. A similar approach should prove fruitful for other hard scheduling problems. However, finding a valid schedule requires a good cooling schedule, optimized parameters, carefully selected update moves, and a good initial solution provided by a preprocessor. A poor implementation in any one of these areas could be the difference between the success and failure of this method.

Acknowledgments

We would like to thank Andrew Gee and Martin Simmen for helpful comments and suggestions, Seong-Joon Yoo for useful discussions on the preprocessing phase, and Hon Yau for comments and suggestions on the paper. The first author is very grateful for the valuable discussion and help of Robert Irwin in converting and formatting the registration data prior to the scheduling process. Many thanks go to Karen Bedard for providing us with the data and answering so many questions we had about it, Meg Cortese for providing us with a set of building constraints for various departments, and Ben Ware, Vice President for Research and Computing at Syracuse University, for his support and encouragement.

	First Semester	Second Semester	Third Semester
Rooms	509	509	120
Classes	3839	3590	687
Professors	1190	1200	334
Students	13653	13653	2600
Buildings	43	43	11
Schools/Colleges	20	21	17
Departments/ Course Prefix	143	141	108

Table 1: Size of the data set for each of the three semesters.

Academic Time Period	Sparseness Ratio
First Semester	0.50
Second Semester	0.53
Third Semester	0.62

Table 2: Sparseness ratios for the data sets for each of the three semesters. Lower values indicate a harder problem.

Academic Semester	Algorithm	Classes Scheduled (average) %	Classes Scheduled (highest) %	Classes Scheduled (lowest) %	Preferences Satisfied (average) %
First	SA (geometric)	65.0	67.5	56.8	48.7
	SA (adaptive)	67.8	70.1	61.2	56.0
	SA (reheating)	70.2	72.3	68.8	59.2
	Expert System	76.6	76.6	76.6	58.0
Second	SA (geometric)	65.6	68.0	57.1	49.0
	SA (adaptive)	68.5	70.1	60.8	57.5
	SA (reheating)	75.1	77.7	70.8	60.0
	Expert System	79.0	79.0	79.0	59.5
Third	SA (geometric)	83.1	86.4	68.5	55.2
	SA (adaptive)	85.8	89.0	70.7	61.0
	SA (reheating)	91.2	95.2	85.0	65.1
	Expert System	96.8	96.8	96.8	64.5

Table 3: Percentage of classes scheduled and student preferences satisfied. The averages and highest and lowest values for simulated annealing were obtained using 10 independent runs. No preprocessor was used for simulated annealing.

Academic Semester	Algorithm	Classes Scheduled (average) %	Classes Scheduled (highest) %	Classes Scheduled (lowest) %	Preferences Satisfied (average) %
First	SA (geometric)	93.9	95.1	85.2	69.0
	SA (adaptive)	98.8	99.2	95.0	77.0
	SA (reheating)	100.0	100.0	100.0	77.95
Second	SA (geometric)	95.0	98.9	89.4	68.6
	SA (adaptive)	99.0	99.5	98.5	77.5
	SA (reheating)	100.0	100.0	100.0	79.6
Third	SA (geometric)	97.6	98.9	90.9	79.0
	SA (adaptive)	100.0	100.0	100.0	93.7
	SA (reheating)	100.0	100.0	100.0	94.1

Table 4: Percentage of classes scheduled and student preferences satisfied. The averages and highest and lowest values were obtained using 10 independent runs. An expert system was used as a preprocessor for the simulated annealing.

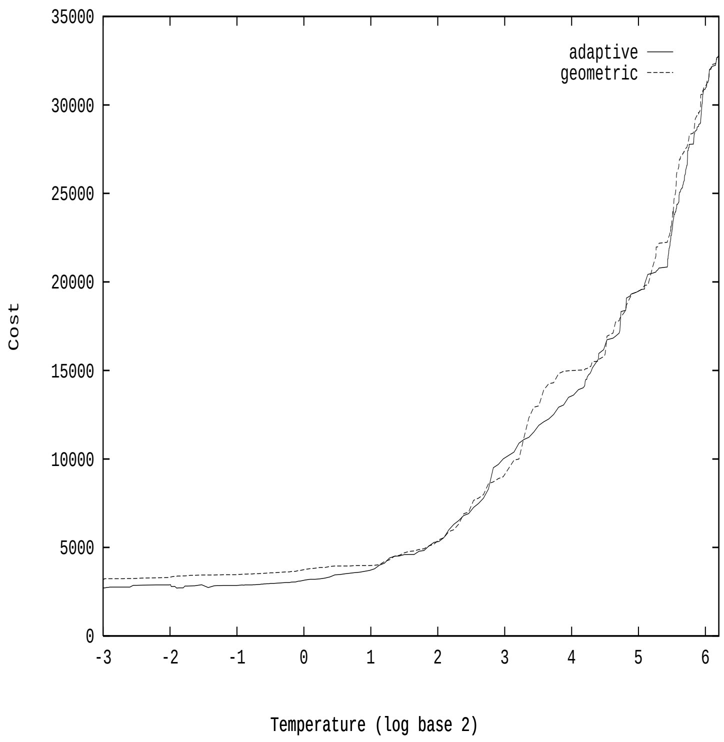

Figure 3: Cost vs. log (base 2) of temperature for simulated annealing using geometric and adaptive cooling.

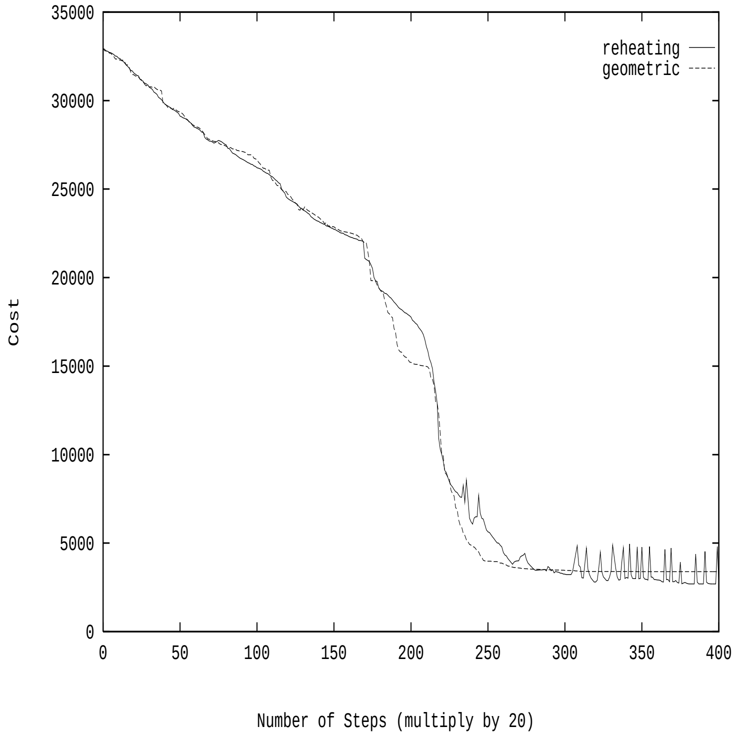

Figure 4: Cost vs. number of annealing steps for adaptive cooling with reheating and geometric cooling.

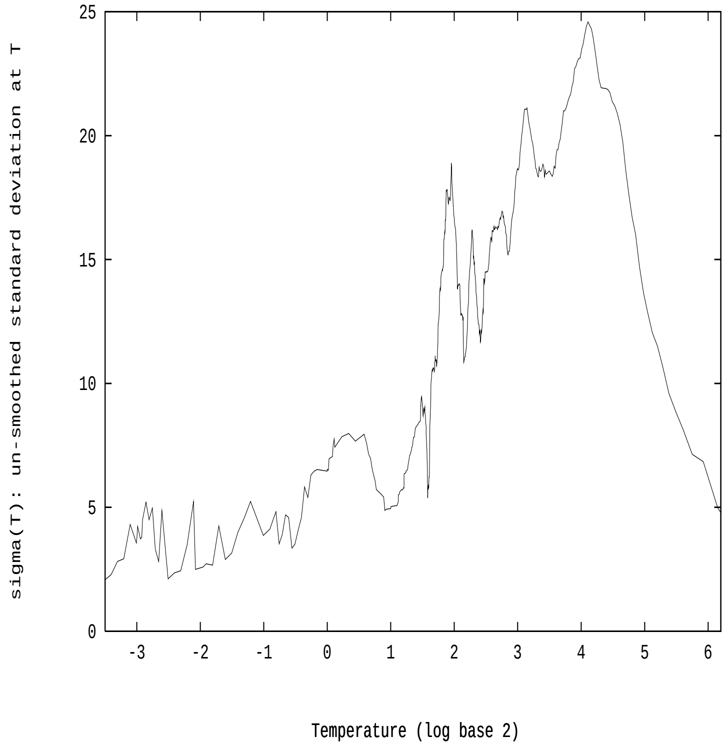

Figure 5: Average specific heat vs. log (base 2) of temperature.

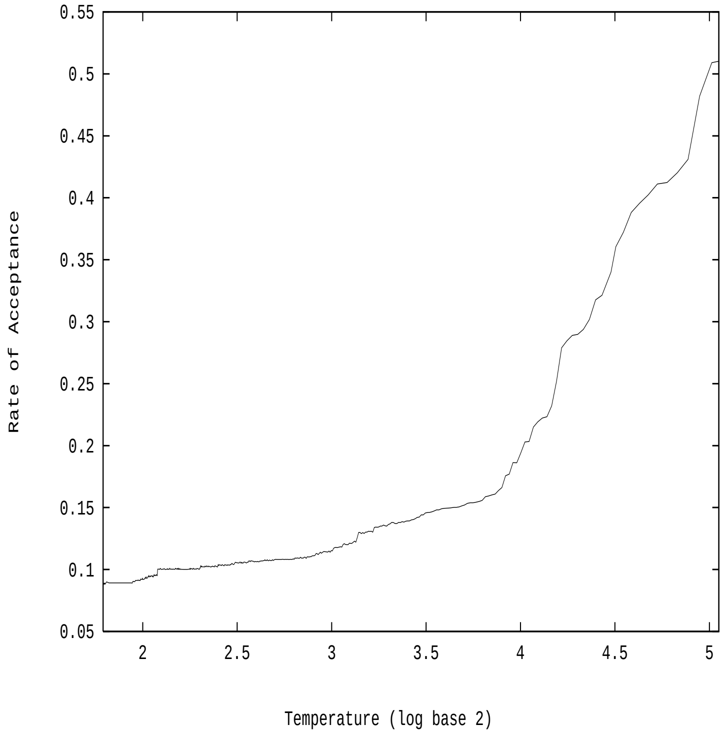

Figure 6: Average rate of acceptance vs. log (base 2) of temperature for annealing using adaptive cooling.

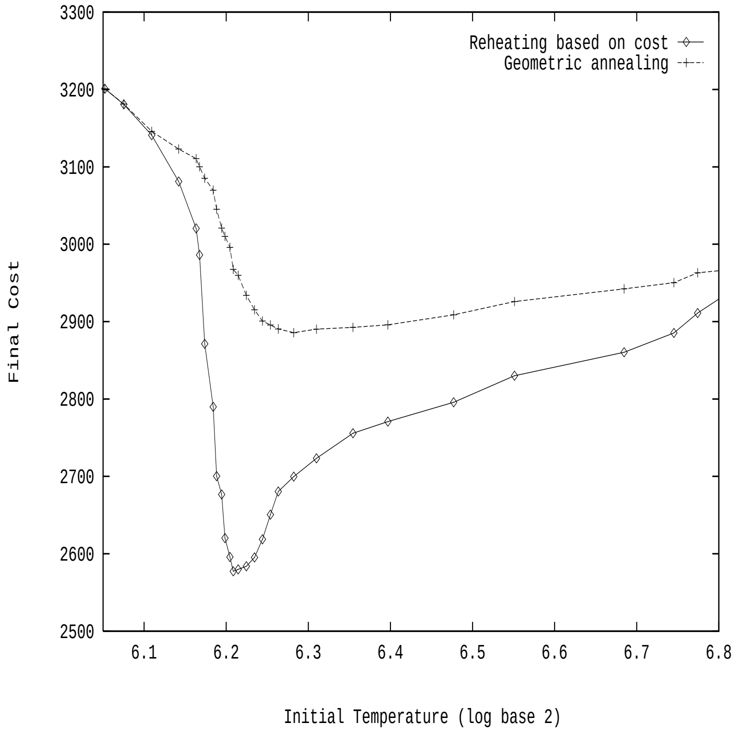

Figure 7: Initial annealing temperature vs. average final cost for adaptive cooling with reheating and geometric cooling.

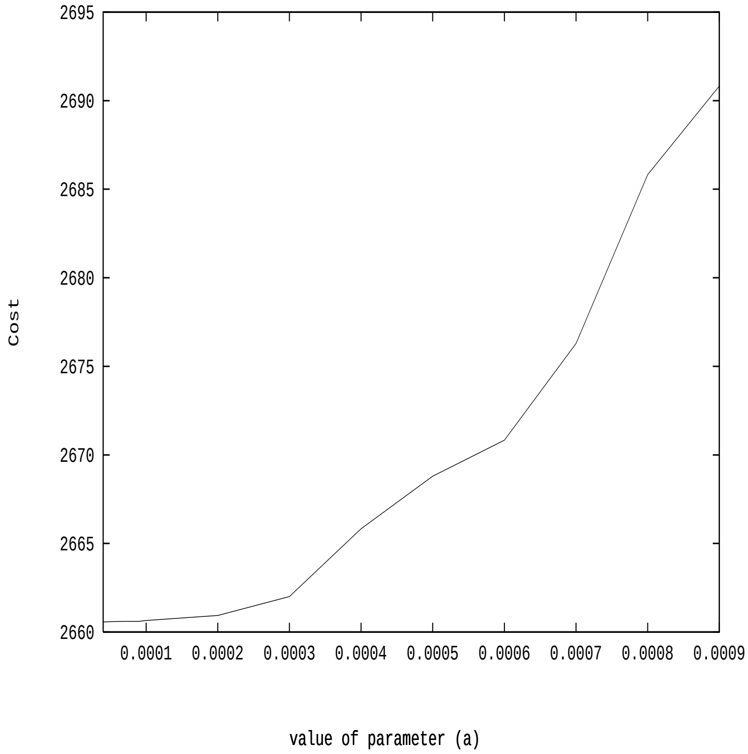

Figure 8: Final cost vs. the value of the parameter a, which governs the rate of cooling in the adaptive cooling schedule.

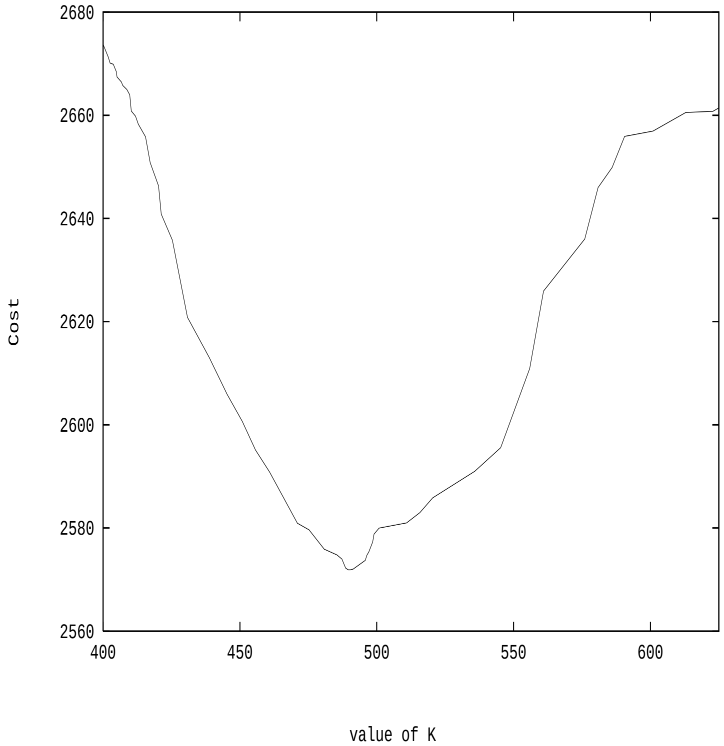

Figure 9: Final cost vs. the value of K, which governs the amount of reheating when using reheating as a function of cost.

References

[1] Aarts, E. H, J. Korst, and P. J. van Laarhoven, "Simulated annealing," in Local Search in Combinatorial Optimization, E. H. Aarts and J. K. Lenstra (eds.), John Wiley and Sons. (to appear in 1997)

[2] Abramson, D, "Constructing school timetables using simulated annealing: sequential and parallel algorithms," *Management Science*, 37(1), 98-113, 1991.

[3] Abramson, D., H. Dang, and M. Krishnamoorthy, "An Empirical Study of Simulated Annealing Cooling Schedules," Griffith Univ. report, Nathan, Qld, Aus. 1994; "Simulated Annealing Cooling Schedules for the School Timetabling Problem," submitted to Asia Pacific Journal of Operations Research, 1996.

[4] Binder, K. ed., Monte Carlo Methods in Statistical Physics, Springer-Verlag, Berlin, 1986.

[5] Carter, M, and C. Tovey, "When is the classroom assignment problem hard?," Working Paper 89-03, Dept. of Industrial Engineering, Univ. of Toronto, 1989. (submitted to Operations Research).

[6] Colorni, A, M. Dorigo, and V. Maniezzo, "On the use of genetic algorithms to solve the timetable problem," report 90-060, Dipartimento Di Elettronica E Informazione, Politecnico Di Milano, Milan, Italy.

[7] Costa, D, "A Tabu Search algorithm for computing an operational timetable," European Journal of Operational Research, 76 (1994), 98-110.

[8] Dige, P., Carlos Lund and Hans F. Ravn, "Timetabling by Simulated Annealing," in Applied Simulated Annealing, R.V. Vidal ed., Lecture Notes in Economics and Mathematical Systems, Springer, 1993.

[9] de Werra, D, "An introduction to timetabling," *European Journal of Operational Research*, 19, 151-162, 1985.

[10] de Werra, D, "Extensions of coloring models for scheduling purposes," *European Journal of Operational Research*, to appear 1996.

[11] Diekmann, R, R. Lüling, and J. Simon, "Problem independent distributed simulated annealing and its applications," in Applied Simulated Annealing, R. V. Vidal (ed), Lecture Notes in Economics and Mathematical Systems, Springer 1993.

[12] Dowsland, K., Using Simulated Annealing for Efficient Allocation of Students to Practical Classes. Stat and OR Group, EBMS, Univ College of Swansea, UK, 1993.

[13] Dowsland, K. and J. Thompson, "Variants of Simulated Annealing for the Examination Timetabling Problem," Working Paper, Statistics and OR Group, European Business Management School, Swansea SA2 8PP, UK. (1994)

[14] Eglese, R.W. and Rand, G.K., "Conference seminar timetabling," *Journal of the Operational Research Society*, **38**, 591-598 (1987).

[15] Eiselt H. A, and G. Laporte, "Combinatorial Optimization Problems with Soft and Hard Requirements," *J. Operational Research Society*, vol. 38, No. 9, pp. 785-795, 1987.

[16] Elmohamed, S., G. C. Fox, P. Coddington, "Course Scheduling using Mean-Field Annealing, Part I: algorithm and Part II: implementation," Northeast Parallel Architectures Center technical report SCCS-782, Syracuse University, 1996.

[17] S. Elmohammed, P.D. Coddington and G.C Fox, "A comparison of annealing techniques for academic course scheduling", Northeast Parallel Architectures Center technical report SCCS-777, submitted to *PATAT'97*, the 2nd international conference on the Practice and Theory of Automated Timetabling.

[18] Even, S, A. Itai, and A. Shamir, "On the complexity of the timetable and multicommodity flow problems," *SIAM J. Computing* 5 (76), 691-703.

[19] Gary, M. and Johnson, D. Computers and Intractability: A guide to the theory of NP-completeness. Freeman, San Francisco (1979).

[20] Gislén, L, Bo Söderberg, C. Peterson, "Teachers and Classes with Neural Nets," *Inter. J. of Neural Systems* 1, 167 (1989).

[21] Gislén, L, Bo Söderberg, C. Peterson, "Scheduling High Schools with Neural Nets," Lund University Preprint LU-TP-91-9, Lund, Sweden (1991), (appeared in Neural Computation 1992).

[22] Gosselin and M. Truchon, "Allocation of Classrooms by Linear Programming," *Journal of Operational Research Society* 37, 561 (1986).

[23] Hertz, A, "Tabu search for large scale timetabling problems," *European Journal of Operational Research* 54, 39-47, 1991.

[24] Hertz, A, "Finding a feasible course schedule using Tabu search," *Discrete Applied Mathematics* 35, 255-270, 1992.

[25] Hogg, T, B. Huberman, and C. Williams (editors), Artificial Intelligence, special issue on *Phase transitions and the search space*, 81, 1996.

[26] Hopfield, J. J, and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics* 52, 141 (1985).

[27] Huang, M, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," IEEE International Conf. on Computer Aided Design (ICCAD), 1986, pp. 381-384.

[28] Johnson, D., "Timetabling university examinations," *Journal of the Operational Research Society* **41**, 39-47 (1990).

[29] Johnson, D, C. Aragon, L. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: an Experimental Evaluation, Part I (Graph Partitioning)," *Operations Research* 37, 865-892 (1989).

[30] Johnson, D., and L. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," in Local Search in Combinatorial Optimization, E. H. Aarts and J. K. Lenstra (eds.), Wiley and Sons. (to appear in 1997).

[31] Kirkpatrick, S, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science* 220 (13 May 1983), 671-680.

[32] Kirkpatrick, S, "Optimization by simulated annealing: Quantitative studies," *J. Stat. Physics* 34 (1984), 976-986.

[33] Ladkin, P. and R. Maddux, "On binary constraint problems," *Journal of the ACM* 41 (1994), pp. 435-469.

[34] Leighton, F. T, "A graph coloring algorithm for large scheduling problems," *Journal of Research of the National Bureau of Standards* 84, 1979, 489-506.

[35] Lister, R.,"Annealing Networks and Fractal Landscapes," *Proc. IEEE Int. Conf. on Neural Nets*, March 1993, Vol. I, pp 257-262.

[36] Mackworth, A. K, "Consistency in networks of relations," *Artificial Intelligence* 8 (1977) pp. 99-118.

[37] Miner, S., S. Elmohamed, and H. W. Yau, "Optimizing Timetabling Solutions Using Graph Coloring," 1995 NPAC REU program, NPAC, Syracuse University, 1995.

[38] Montanari, U., "Networks of constraints: fundamental properties and applications to picture processing," *Information Sciences* 7 (1974), pp. 95-132.

[39] Mouritsen, O. G., Computer Studies of Phase Transitions and Critical Phenomena, Springer-Verlag, Berlin, 1984.

[40] Otten, R, and L. van Ginneken, The Annealing Algorithm, Kluwer Academic Publishers, 1989.

[41] Peterson, C, and Bo Söderberg, "Artificial Neural Networks and Combinatorial Optimization Problems," To appear in *Local Search in Combinatorial Optimization,* E.H.L. Aarts and J.K. Lenstra, eds. Wiley and Sons. (to appear 1997)

[42] Peterson, C, and Bo Söderberg, "A New Method for Mapping Optimization Problems onto Neural Nets", International Jr. of Neural Systems 1, 3 (1989).

[43] Schaerf, A, "A survey of automated timetabling," Department of Software Technology, Report CS-R9567, CWI, Amsterdam, The Netherlands.

[44] Shrijver, A, Theory of Linear and Integer Programming, John Wiley and Sons, Chichester, UK, 1986.

[45] Shrijver, A, A Course in Combinatorial Optimization, to appear in 1997.

[46] Thompson, J, and K. Dowsland, "General Cooling Schedules for Simulated Annealing Based Timetabling Systems," Proceedings of the 1st International Conf. on the Practice and Theory of Automated Timetabling, Napier Univ., Edinburgh 1995.

[47] van Laarhoven, P. J. and E. H. Aarts, Simulated Annealing: Theory and Applications. D. Reidel, Dordrecht (1987).

[48] Rene Vidal (ed), Applied Simulated Annealing, Lecture Notes in Economics and Mathematical Systems no. 396, Springer-Verlag, 1993.

[49] White, S. R, "Concepts of scale in simulated annealing," *Proceedings of the IEEE Int. Conference on Circuit Design*, pp 646-651, (1984).