Simulated Annealing, and non-Euclidean Combinatorial Optimization Problems: The case of Academic Course Scheduling, Theory and Implementation

Saleh Elmohamed
Geoffrey C. Fox
Paul D. Coddington
Hon W. Yau
Northeast Parallel Architectures Center,
Syracuse University
Syracuse, NY 13244
{saleh, gcf, paulc, hwyau}@npac.syr.edu

ABSTRACT

In this study we tackle a complex problem of academic course scheduling (or timetabling) at the university level. The method used is that of simulated annealing using two different cooling schedules. An expert system is used as a pre-processor in both approaches, which contributed greatly to the improvement of the final output. In summary, this multi-phase approach to timetabling produces much better solutions (schedules) than the single method approach.

— need to be revised a little —

Key words : stochastic updating, simulated annealing, reheating, cost functions, temperature, annealing schedule, phase transition, rule-based system.

June 18, 2000

Contents

List of Figures

List of Tables

1 Introduction

The primary objective of this study is to derive an approximate solution to a high-dimensional and non-geometric combinatorial optimization (CO) problem, namely academic scheduling at the university level. Our method for tackling this problem is simulated annealing using two different cooling schedules: the standard geometric cooling, and the geometric re-heating as a function of cost.

The problem of academic scheduling or timetabling (TTP) has been tackled by many researchers [34], mostly in the field of operations research. This is a very difficult problem due to its nature as a multi-constraint satisfaction non-Euclidean combinatorial optimization problem for which no particular heuristics or method has been shown to produce the best (or optimal) results for the problem.

Lately, there have been a number of *non-conventional* (or novel) methods utilized to tackle this problem from different angles. Among them are neural networks [22], simulated annealing [26, 4], genetic algorithms [24], tabu search [25], and various combinations of two or more of these techniques. None of these methods are used with as large and complex problem as we have, and with the exception of [22, 25], none of the others have used real world scheduling data. In this paper, we shall confine ourselves to simulated annealing using the cooling schedules of geometric annealing and re-heating as a function of cost. In addition, a rule-based system is used as a pre-processor in the application of each of these schedules.

In section 2 we give a brief review of TTP and its complexity. Also, we state the three categories of constraints along with samples of each that we have implemented in our experiments. Section 3 outlines an approach to carry out optimization of the problem over integral polytopes and in section 4 we state our approach to deal with the timetabling problem. Section 5 is on simulated annealing and the mapping of the different TTP components onto the annealing cost function. We state the cooling schedules in section 6 and then discuss our experimental results in section 7. Lastly, in section 8 we summarize and conclude the paper.

2 The Timetabling Problem

A problem that does not have a well-defined objective function (as in the traveling salesman problem (TSP), for example) to optimize and belongs to the class of constraint satisfaction problems is the NP-complete *teachers and classes* timetabling problem studied in [4, 1, 19]. Briefly, the 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 lessons (defined as teacher-class pairs) within a time limit of N_t lesson periods (or time slots). 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.

2.1 The Constraint Satisfaction Problem

We now briefly describe the general constraint satisfaction problem which has been widely studied in the Artificial Intelligence community [7, 8, 9].

An instance of a constraint satisfaction problem consists of:

- A finite set of variables, N_i, where $i = 1, 2, \ldots, n$.
- A domain of D of possible values of N_i .
- A set of constraints $\{C(S_1), C(S_2), \ldots, C(S_c)$, in which each S_i is an ordered subset of the variables, and each constraint $C(S_i)$ is a set of tuples indicating the mutually consistent values for the variables in S_i .

The length of the tuples* in a given constraint will be called the *arity* of that constraint†. In particular, unary constraints specify the allowed values for a single variable, and binary constraints specify the allowed combinations of values for a pair of variables. A solution to a constraint satisfaction problem is an assignment of values to the variables which is consistent with all of the constraints. Deciding whether or not a given problem instance has a solution is NP-complete in general [9], even when the constraints are restricted to binary constraints.

2.2 Constraints of the Timetabling Problem

In a nutshell, 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. Also, as alluded to by Eiselt and Laporte [29], a constraint can be hard, medium or soft. The medium and soft constraints have an associated cost and if they are not satisfied, the goal is to minimize this cost. In regard to the hard constraints, one must derive their associated cost (or penalty) to zero. 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 but always preference is to minimize them.

*A tuple here is a combination of a class, teacher, and a classroom. The time periods are assumed to be given.

†Strictly speaking we also need to associate an arity with an empty set of tuples, but we will neglect this special case, in order to simplify the presentation.

2.2.1 Hard Constraints

The most common and important hard constraint is what is sometimes referred to as an *edge* constraint between two vertices or events (e.g. e_1 and e_2). This reference is due to the similarity of any simple timetabling problem with *vertex coloring* in graph theory; which simply states that any given pair of events (e_1, e_2) must not overlap in time. Those events that must not overlap in time include:

- two classes taught by the same professor,
- a class and a recitation or a lab of the same class,
- classes with students in common.

Other examples of hard constraints are

- *Room constraint*: essentially says that a class cannot be assigned to a particular room unless size of the room is greater than or equal to the class enrollment. The smaller the difference between these two sizes, the better the assignment. This constraint is implemented using a non-linear function computing the magnitude of the difference between the size of room and the class enrollment as follows: Let CE stands for class enrollment and RC for room capacity, then C_{rs} (cost associated with the room size) is computed as follows:

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

  In our experiments, β and γ are set to 0.5 and 0.8, respectively.

- *Exclusion constraint*: deals with any violation to the given list of classes that can not be scheduled at the same time or have any overlapping scheduling time. Here is how the exclusion cost C_e is obtained from this constraint:

  $\forall i, j$ such that $i \neq j$, let c_i denotes class(i), c_j denotes class(j), $s(c_i)$ is the starting time of c_i, $end(c_i)$ is the end time of c_i, $len(c_i)$ is 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. Also, the same notation applies to class c_j.

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

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

- Section meeting time may not conflict with other "enrolled" sections.

Also, there are hard constraints of relevance to students, such as:

- To sign up for a class or section, it must be open.
- Eligibility criteria for the section must be met, etc.
- Do not enroll athletes in courses that conflict with their sport practice time (of course, depending on the sport), and so on.
- Enroll students in ALL parts of a course when associated sections (i.e. recitation, labs) are required.

These constraints have to be respected at all costs.

2.2.2 Medium Constraints

These kinds of constraints fall into the shaded region between the *hard* and *soft* constraints in the sense that they do not directly deal with time conflicts as do the hard type, nonetheless, have a high penalty attached to them. Also, this penalty is not as high as that associated with the hard constraints. In the final (legal) schedule the penalty of these constraints is minimized and it is quite preferable to reduce it to 0.

In short, these kind of constraints should be satisfied but are nevertheless relaxable. Some examples of medium constraints:

- For each student, balance the three-day (*Mon, Wed, and Fri* (MWF)) as well as the two-day (*Tue, and Thu* (TuTh)) schedules.
- Balance or spread out the lectures over the week.
- Classes may request contiguous time slots.
- Classes may be grouped with other classes.
- The overall schedule should have as few "holes" as possible; that is lessons should be glued together or the separating intervals are as short as possible.
- It is not the case, in general, that all classrooms are available for all subjects. In reality, many subjects require a special purpose rooms.

2.2.3 Soft Constraints

Like medium constraints, they also do not deal with time conflicts at all but unlike the medium constraints, have a low penalty (or cost) associated with them. Furthermore, this cost is lower than that of the medium constraints and much lower than that of the hard ones. In addition, in the final schedule, this cost is minimized but does not often approach zero. Here are few examples of soft constraints:

- Balance enrollment in multi-section courses.
- Lunch and other break times may be specified.
- Professors may request periods in which their classes are not taught.
- Professor or students preferences for specific rooms even if they are taken from a group of rooms.
- An enrollment balance by gender should be obtained in Honors seminar sections.
- Optimize first choice course enrollments by attempting all first choice courses for all students before attempting second and third choices[‡].
- *Distance constraint*: deals with the distance between where the class is assigned and the building which is housing its home department. Before actually stating what this constraint is we will first outline how we have dealt with the aspect of distance.

  Given the various building preferences of the departments, we have constructed a matrix $M[I, J]$ between all the academic departments $[I]$ and all the buildings involved in scheduling $[J]$. Using this matrix in conjunction with the real —appropriately scaled— distance, a final state matrix is derived and directly used in the scheduling process. Therefore, unless otherwise indicated, for a certain department the home building is always the first preference where the department courses to be assigned.

  Let B denotes the set of all buildings and let say $|B| = m$, and D to denote the set of all departments, $|D| = n$, and $dist$ denotes the distance. Now, for each $d_i \in D$ where $1 \leq i \leq n$, $dist(d_i) =$ vector $\bar{m} < d_i, b_j >$ for $1 \leq j \leq m$ of matrix $M[D, B]$. Also, let $dist(d_i)_{min}$ = minimum of $\bar{m} < d_i, b_j >$ for $1 \leq j \leq m$. Now C_{d_i} (the distance cost of d_i) is computed as follows:

$$C_{d_i} = \begin{cases} dist(d_i)_{min} \frac{dist(d_i)}{\sum_{j=1, i \neq j}^{m} \bar{m} < d_i, b_j >} + dist(d_i)_{min} & \text{if } dist(d_i) > dist(d_i)_{min}\,, \\ dist(d_i)_{min} & \text{otherwise.} \end{cases} \tag{4}$$

- Professor or students preferences for certain department-to-building distances.

2.3 Cost Function and the Solution Space

The cost function measures the infeasibility of the current solution and may involve the weighted sum of penalties associated with different types of feasibility violation. Our reference to "feasibility" here means that a timetabling schedule is feasible if it satisfies all of the hard constraints.

[‡]This constraint is treated as an optional.

Let C be the set of the overall given constraints. A feasible solution is defined as a schedule that satisfies $C^{'}$ such that $C^{'} \subseteq C$, provided that all hard constraints belong to $C^{'}$. Also, a neighboring solution ($s^{'}$) is obtained by modifying the current solution (s) space (i.e. swapping two or more classes in time and/or space). On the question of determining whether a given constraint belongs to the set $C^{'}$ or not, roughly we follow this rule and that is, a class should never be assigned to a time period if such an assignment induces only class schedules that do not satisfy the given constraints (specifically, all the hard constraints).

The neighborhood of solutions (both feasible and infeasible) that s and $s^{'}$ belong to is denoted by $\mathcal{N}_s$.

We consider the following as part of the abovementioned cost function:

- *Distance cost*: obtained from the distance constraint.
- *Time period cost* which consists of the following costs: cost of class or classes assigned to that period, (for example, the cost that result from having a certain class and its lab scheduled in the same time period); cost of class spreading over the week and within each of the weekdays; cost of class gluing and filling in any holes of the timetable; and students cost which is the sum of various costs associated with each student, such as eligibility criteria and number of classes taken by the student.
- *Professor cost* is computed from the following costs: cost incurred from assigning more classes than the maximum number taught by that professor; cost from not having a long enough time interval between courses taught by the same professor; and cost of assigning to the same professor two or more classes that overlap in time.
- *Room size cost* is obtained from the room constraint.

Summing these costs is not strictly necessary, provided that the components (individual costs) can be combined in some fashion to reflect the overall quality of the solution.

2.4 Complexity of Timetabling

Timetabling belongs to the class of constraint satisfaction problems and it is essentially an example of a resource-constrained scheduling problem, which is an NP-hard [10, 18, 38]. Here, resources are physical entities, for example: students, rooms, overhead projectors, and so on. Also time periods can be thought of as resources.

In general, scheduling problems are almost always NP-complete which means there exist no known polynomially bound algorithm for solving them optimally. As a result, these problems are often solved by means of *heuristics*: that is 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. Also, 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 which represent the problem. Clearly, as the size of the input increases so does the number of variables required to represent it; making the problem intractable.

For a much simpler problem, which is the *class allocation* (but with no timetabling involved, i.e. no allocation of classes to time periods), the computational load (n_c) from tackling this problem using the LP method scales as follows [3, 20]:

$$n_c \propto N_c N_x N_t \quad , \tag{5}$$

where N_c is the number of room categories (i.e. classroom, auditorium, etc.).

3 Some Methods of Tackling Timetabling

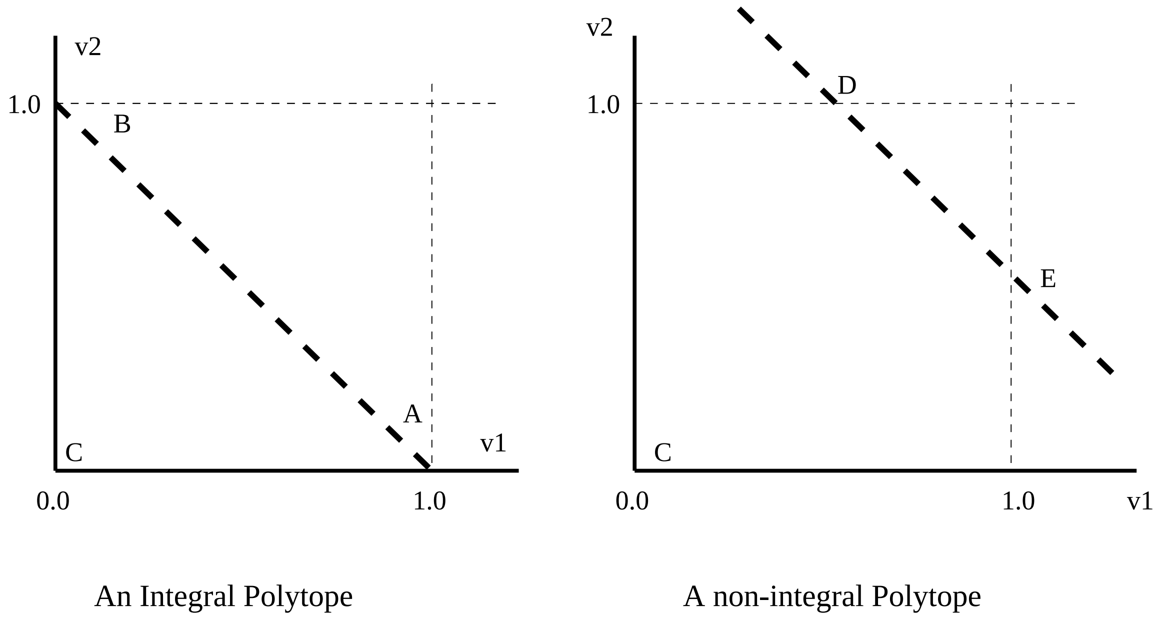

Figure 1: An integral polytope AB has all its vertices at integral points (A = 1.0 and B = 1.0), as shown in the left hand side figure, but polytope DE in the right hand side figure is not integral since its ends don't pass through $[0-1]$ points.

As we have previously stated that TTP is a multi-constraint satisfaction problem with hard, medium, and soft constraints each with their associated cost. A solution from an integer programming point of view is basically to find any integral vertex within the constraint polytope P, as shown in Figure 1 and Appendix A. To elaborate a bit on this, using the work of Schrijver [12, 13]: we think of each constraint as a hyperplane in the solution space. The set of constraints define a polytope or (a bounded polyhedron) within which the state vector must remain if it is not to violate any constraints. The

0 or 1 solution points will lie at some of the vertices of this polytope, and whatever annealing or other local search method is used, it must force convergence to one of these points. The problem with methods such as annealing (certainly depends on the cooling schedule used) is that in this case the best one can expect is convergence to a vertex, not necessarily a 0 or 1 vertex.

Polytopes, for example, like the one in the permutation matrix mapping for the TSP are integral, so there is no problem finding valid TSP solutions (though the minimum tour length is another question altogether). Class scheduling polytopes are a totally different story: Gee [42] and some of his colleagues at Cambridge University have done some experiments on them and found that for timetabling problems as small as five teachers, five classrooms, five classes, and five time-slots, the proportion of 0-1 vertices is only about 1%. This mean that on average one has to run a descent procedure about 100 times to find a valid solution! Some of our experiments with sets of teachers, classes, classrooms, students and time-slots of various sizes confirm some of Gee's results and conclusions.

Timetabling problems can be characterized by their *sparseness*. After all the required lessons have been scheduled, there will be $N_s = (N_x N_t - N_l)$ spare space-time slots, hence, the sparseness of the problem is defined as the ratio $N_s/(N_x N_t)$. The sparseness decreases as $N_s/(N_x N_t)$ goes to zero or as N_s becomes smaller and smaller. Furthermore, Gee [42] notes that for sparse problems, the probability of an arbitrary descent procedure converging to an integral vertex is near one, though this falls off rapidly for less sparse problems. For dense problems, there is an additional correlation involving the problem size: as the problem gets larger (more classes, teachers, rooms, students, etc.), the proportion of integral vertices gets smaller. From this we could deduce that a one-shot descent procedure, operating on an arbitrary "objective function", is not likely to be very successful with difficult constraint satisfaction problems, most probably converging to a non-integral vertex of P, for example. On the other hand, any "partial" or "imperfect" solution to a timetabling problem found by methods such as simulated annealing (using some good adaptive schedules) is likely to be of some use. Since simulated annealing does not search over the interior of the unit hypercube, but only over its vertices, the solution it finds will be a unique timetable, possibly violating few soft and/or medium constraints. As part of the post-processing operation, we could at least remove some lessons to obtain a valid (or legal) timetable which fulfills most of the teaching requirements.

4 Our Approach in Tackling the Problem

Given data sets of courses and their days, hours, enrollments, and instructors; rooms and their capacities, specification, and locations; distances between buildings; priorities of each building; and students and their courses: the problem was to construct a legal school schedule satisfying all the hard constraints and minimizing the medium and soft constraints. Courses that were not assigned to rooms due to conflict in time or size were left out of the final schedule for the user to handle manually. We took a multi-phase approach to tackle this complex problem [29]. In the first phase a feasible solution is found, and in the second phase the current solution is improved to obtain a final feasible solution.

In the first phase, the preprocessing phase, we used a rule-based system giving a feasible, though partial solution as an output. A partial solution in the sense that due to time conflicts not all of the given courses were assigned to rooms and times slots. Also, the preprocessing output was divided into two parts. The first part consisted of classes, with their associated professors and students, assigned to various rooms; and the second part was a list of classes which could not be assigned due to time and/or space conflict.

In the second phase, the processing phase, the output of the first phase along with the three types of constraints were fed into the main processing method, which was simulated annealing using two different cooling methods, to produce the final schedule.

4.1 The Preprocessing System

As mentioned above, this was an expert (rule-based) system consisting of a number of rules (or heuristics) and conventional recursion to assist in carrying out course assignments. We have developed this system specifically for the problem of academic scheduling. The basic data structures or components of the system are: first, distance matrix of integer values between each academic department and every other building under use for scheduling. Second, course data structure of each course scheduled anywhere in campus. Also, these structures are capable of linking with each other. Third, room data structure of each room (regardless of type) involved in the scheduling process. Like courses, room structures are also linked with each other. Fourth, data structures for time periods to keep track of which hour or time slot was occupied and which was not. Fifth, department inclusion data structure giving department inclusion within other larger departments or colleges.

The basic function of the system is as follows: given data files of courses, rooms and buildings, department-to-building distance matrix, 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.

4.2 Motivation and Rationale for Preprocessing

It is always a good practice when using techniques such as simulated annealing, to start at a good point in the search space or with a partial solution regardless of the difficulty or complexity of the problem at hand. Such a starting point can only have a positive effect on the overall solution, and that is improving its quality. Also, a good starting point will have a positive effect on the convergence of the system, that is, faster convergence to the final solution. Our experimental results, as shown elsewhere in this paper, clearly support this rationale.

4.3 Examples

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

4.3.1 Space and Hours

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

THEN assign the room to the course:

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

Input: CPS-615 requests space for 75 students and meeting hours of [MWF 10:40 - 11:35 am].

In the database we have, for example, the following rooms:

- room: 1-218 CST, capacity: 45 students, and pre-occupied hours: none.
- room: 207 HL, capacity 120 students, and pre-occupied hours: [MW 09:45 - 11:10 am].
- room: Kitt-Aud HBC, capacity: 127 students, and pre-occupied hours: [TuTh 10:40 - 12:00 am].

Output: *Assign room Kitt-Aud to class CPS-615.* Clearly this assignment meets the class requirement of size and room availability at the requested meeting time.

4.3.2 Type of Room

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

THEN assign the room to the course:

[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 course(space-requested)].

Input: BIO-615 (not biology lab) requests space for 35 students and meeting hours in [MW 10:40 - 11:35].

In the database we have, for example, the following rooms:

- room: 1-218 CST, capacity: 45 students, type: classroom, and pre-occupied hours: none.
- room: 124-SIMS, capacity: 40 students, type: laboratory, and pre-occupied hours: none.

Output: *Assign room 1-218 CST to class BIO-615.* This assignment meets the criterion of room type, eventhough size of the second item in the database is more ideal given the class enrollment.

4.3.3 Room Priority

Rooms are classified as either general purpose (GP) which are used by all, or priority rooms (PR) with a usage based on certain priority of the class or the department, or special purpose (SP) 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) > course(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) > course(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) > course(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) > course(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) > course(space-requested)] and [no time conflict in assigning this room] and [type(room) matches type(requested)]*

  THEN assign the room to the user.

5 Timetabling via Simulated Annealing

Simulated annealing (SA), see Figure 2, has been widely used for tackling different combinatorial optimization problems, particularly academic scheduling [1, 2, 26]. The results obtained depend heavily on the cooling schedule used. We initially used the most commonly known and used schedule, which is the geometric cooling, but later tried the method of geometric re-heating based on cost [4].

A comprehensive discussion of the theoretical and practical details of SA is given in [15, 37]. It suffices here to say that the elementary operation in the Metropolis method for a combinatorial problem such as scheduling is the generation of some new candidate configuration, which is then automatically accepted if it lowers the cost (C), or accepted with probability $\exp(-\Delta C/T)$, where T is the temperature, if it would increase the cost by $\Delta(C)$. Thus the technique is essentially a generalization of the local optimization strategy, where, at non-zero temperatures, thermal excitations can facilitate escape from local minima.

The SA algorithm has some advantages as well as some disadvantages. Among its advantages are the relative ease of implementation, the applicability to almost any combinatorial optimization problem, depending on the cooling schedule(s) used it usually provides reasonable solutions, and it can easily be combined with other heuristics, such as rule-based systems, forming quite useful hybrid methods for tackling a range of complex problems. One major drawback of the algorithm is the requirement for long run times.

5.1 Mapping of Timetabling Into Annealing Algorithm

The most obvious mapping of TTP into the SA algorithm involves the following constructs:

Let M_p = maximum number of classes assigned to any professor.

1. a **state** is a timetable containing the following sets:
   - P: a set of professors.
   - C: a set of classes.
   - S: a set of students.
   - R: a set of classrooms.
   - $\mathcal{I}$: a set of time periods.
2. a **cost** or "energy" $\mathrm{E}(P, C, S, R, \mathcal{I})$ such that:
   - $\mathrm{E}(P)$: is the cost of assigning more than M_p classes to the same professor, plus scheduling one or more courses that cause a conflict in the professor's schedule.
   - $\mathrm{E}(C)$: is the cost of scheduling certain classes at/within the same time period in violation of the exclusion constraint, for example.
   - $\mathrm{E}(S)$: is the cost of having two or more classes conflict in time; plus cost of having in the schedule one or more classes that really don't meet the student's major, class requested, or course requirements; plus the cost of not having the courses evenly spread over the week, etc.
   - $\mathrm{E}(R)$: is the cost resulting from assigning room(s) of the wrong size and/or type to a certain class.

- E($\mathcal{I}$): is the cost of having more or less time periods than required, plus cost of an imbalanced class assignments (a certain period will have more classes assigned to than others, etc.).

3. A **swap** (or a move) is the exchange of one or more of the following: class c_i with class c_j in the set C with respect to time periods $\mathcal{I}_i$ and $\mathcal{I}_j$, and/or with respect to classroom R_i and R_j, respectively. Generally, this step is referred to as class swapping.

Along with all of the necessary constraints and the output of the expert system, the algorithm also takes as input data the following: a list of courses that have not been already scheduled 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 course preferences, and a list of courses that are not allowed to be scheduled simultaneously.

5.2 More on Simulated Annealing

SA is a robust technique and has been applied with success to a wide variety of problems. However, to obtain good results the annealing runs often require a great deal of computer time. Johnson et al. [16] criticized much of the earlier research for only comparing SA with the simpler local optimization heuristics (i.e. those using r-Optimization (or r-Opt) style moves) and not with the more competitive Lin-Kernighan (LK) heuristics [28]. They undertook a systematic comparison of the performance of the above three methods on the graph bisection problem, for low degree random and random geometric graphs. When assessed using equivalent amounts of computer time, SA emerged a clear winner over simple local optimization for both types of graph. Against the LK-style algorithm however, SA was only slightly better for random graphs and worse for the random geometrical class. A similar investigation was carried out for the TSP [17, 31] and the results suggest that, matched for run-time, SA beats 3-opt, also can be quite competitive with LK, but not as good as a well-designed and implemented LK or the Iterated Lin-Kernighan (ILK) heuristics [17].

In summary, depending on the given problem, SA appears competitive with many of the best local optimization methods. Furthermore, the inherently serial nature of the approach in general precludes any obvious parallel implementation, though several attempts have been made to overcome this problem such as the recent work by Frost [14], Boissin and Lutton [35], and Verhoeven and Aarts [36]. The most promising approach to parallelizing SA is to distribute the execution of the various parts of the algorithm over a number of communicating parallel processors which may contribute to a significant speedup in the overall runtime of the method.

6 The Annealing Schedules

Two annealing schedules have been used in our experiments to update the temperature of the SA algorithm in Figure (2): geometric cooling and an adaptive re-heating as a function of cost. This later schedule was presented in Abramson et al. [4] but the ideas behind it are actually due to Kirkpatrick et al. [32, 33] and White [27].

Let α $(0 < \alpha < 1)$ denotes the cooling factor. The first schedule, geometric cooling, computes the new temperature $(T^{'})$ of the SA algorithm as follows:

$$T^{'} \; = \; \alpha \, T \; . \tag{6}$$

Typically the value of α is chosen from the range $0.90 \ldots 0.99$. This schedule has three main advantages and these are: it is based on a sound theoretical foundation, well understood by all researchers working on annealing, and the most widely used cooling schedule. Because of these advantages it is used as a base line for comparison with other scheduling schemes. Furthermore, our results obtained from using it were compared with those of the second schedule.

The second annealing schedule we used to update the temperature of the SA algorithm is the method of reheating as a function of cost (RFC). Before stating this schedule we would like to summarize few relevant points to the concept of specific heat (sh). Specific heat is not a new concept and in their seminal work, Kirkpatrick et al. [32, 33] measured it by using the variance of solution values encountered 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 can be observed as sub-parts of the problem are resolved. In some of the work dealing with the TSP using annealing, e.g. [5], the authors often 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. As reported in [4], applying a reheating type procedure, depending on the phase, would 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.

In order to calculate the temperature at which a phase transition occurs, it is necessary to compute the specific heat of the substance. A phase transition occurs at a temperature T_{msh} when the specific heat is maximal (msh), and this triggers the change in the state ordering. If the best solution found to date is poor (i.e. 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_{msh}). Generally, the higher the best cost, the higher the temperature which is required to escape the local minimum. Now, to compute the aforementioned maximum specific heat we employ the following steps which we refer to as the RFC schedule:

Let C_i to denote the cost of configuration i, $C(T)$ is the cost at temperature T, and

$\sigma(T)$ is the standard deviation at T. At temperature T, the acceptance probability is:

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

For a set of configurations $\mathcal{C}$, the expected cost at temperature T:

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

The expected square cost at T:

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

The variance of the cost (or energy) at T is:

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

Using the above derivations, the specific heat:

$$T_{sh}(i) = \frac{\sigma^2(T)}{T^2} \ , \ \text{where} \ i \in \{1, 2, \ldots, n\} \, . \tag{11}$$

Now, the temperature at which the maximum specific heat occurs, or at which the system changes state is computed as follows:

$$T_{msh} = \ \text{Maximum-of} \ (T_{sh}(i)) \ , \ \text{where} \ i = 1, 2, \ldots, n \, . \tag{12}$$

The new temperature is set to:

$$T = P \cdot C_b(s) + T_{msh} \ . \tag{13}$$

P is a constant and $C_b(s)$ is the current best cost of the given configuration s. Also, we have used another variant of equation 13 to compute a new temperature based on not the best cost found so far but rather on the standard deviation of costs at the current T. A similar approach was taken by Huang et al. [39] yielding an efficient cooling schedule.

Let T_j denotes the current temperature T. After calculating $\sigma(T_j)$ from equation 10, the new T which is denoted as T_{j+1} is as follows:

$$T_{j+1} = T_j \cdot e^{-\frac{0.0001 T_j}{\sigma(T_j)}} \ . \tag{14}$$

Following suggestions by Otten and van Ginneken [40] and Diekmann et al. [41], $\sigma(T_j)$ is smoothed out in order to avoid any dependencies of the temperature decrement on large changes in the standard deviation σ. We followed the following the general form of $\sigma(T_{j+1})$ smoothing:

$$\sigma(T_{j+1}) = (1 - \omega)\sigma(T_{j+1}) + \omega\sigma(T_j)\frac{T_{j+1}}{T_j} \tag{15}$$

and set ω to 0.95.

It follows from the Boltzmann distribution (see [27, 11]) that

$$\frac{d}{dT} E_T = \frac{d}{dT} C(T) = \frac{\sigma^2(T)}{T^2} \tag{16}$$

which is essentially equivalent to equation 11.

1. Generate an initial solution (i.e. schedule) s .
2. Set the initial best solution $s^* = s$.
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 solution, $(s' \subset \mathcal{N}_s)$.
      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 solution s^*.

Figure 2: The Simulated Annealing Algorithm

7 The Strategy in Choice of Moves

Besides satisfying the constraints stated in this paper, we also have taken a number of steps throughout the scheduling process to further eliminate any unnecessary and unbeneficial swaps between various categories of courses. For example, we made sure that a course of type lecture cannot be swapped with a course of type laboratory and vice verse. In a sense, we separated those two types and then carried out the scheduling of lectures first followed by scheduling of laboratories, provided that no lecture and its associated laboratory are scheduled in the same time period. Other productive moves we have taken is that swapping a higher level course (e.g. graduate) with a lower level course (e.g. a first or a second year type) is allowed in general since it results in a very negligible cost. Therefore, the degree of conflict (or cost) we have dealt with in this case goes from high to low with the following swaps (highest with the first and lowest with the last):

- Swapping two courses of the same year (e.g. two freshman courses).

- Swapping a first year course with one of a second year.
- Swapping a third year course with one of a fourth year.
- swapping a fourth year course with a graduate level course.
- swapping a third year course with a graduate level course.
- swapping a second year course with a graduate level course.
- swapping a first year course with a graduate level course.

Also, throughout scheduling we have assumed that swapping courses that belong to different academic departments always involve lower costs than that of swapping courses belonging to the same department. The justification for this assumption is that courses that belong to the same department often tend to have students and instructors in common. In addition, they usually are scheduled in the same set of rooms allocated to or chosen by that department. Therefore, part of our strategy of moves is to limit as much as possible the interdepartmental course swaps. Also, one of the steps we have taken throughout scheduling is to schedule larger classrooms and auditoriums first in order to avoid, if possible, not having large enough rooms later for those classes with large enrollments (e.g. greater than 35 students). One other assumption we have made is that those constraints involving courses that are not available during certain time period(s) due to the unavailability of their instructors at those times are harder than constraints of no overlapping courses involving common student(s).

Generally, the techniques we have taken here can very well be considered as prunning the neighborhood or narrowing or limiting the search space heuristics, contributing to a much improved overall results.

8 Experimental Results

We view our computation to be of two kinds. One kind is to determine an acceptable set of parameters that can be used during annealing, and the second type involving the data sets of our study case. In addition, the first computations is conducted with respect to the aforementioned data sets.

8.1 Schedule-Specific Results

As mentioned above we have used two cooling schedules for our experiments, reheating based on cost and geometric cooling. Each schedule takes a set of parameters — the temperature reduction constant (α), time spent at each temperature, initial temperature, and for the second schedule a final temperature. Overall, we found that the effect of the temperature reduction factor was even less important than was predicted on theoretical grounds. (If it was replaced with, say, its square root, and the number of moves at the initial temperature is doubled, the solution quality will be the same). It was arbitrarily fixed at 0.95. Since we desire a comparison of the cooling schedules over a range of run times and solution qualities, the time spent at each temperature was left as an independent variable. In some of the runs we did set this independent variable (or the number of steps at each temperature) to be at least proportional to the neighborhood size or the size of our data.

For geometric cooling, the initial comparison was with a standard sort of stopping criterion [16, 31]: the run was terminated when no moves had been accepted for three generations[§]. Also, we have observed that the choice of initial temperature poses more of a challenge. To be exact, the experimental procedure was this: Two data sets were obtained for both schedules. For the geometric cooling, its set consists of ten runs made from each of ten starting temperatures anywhere between 100 and 500 with increments of 10's. For the reheating schedule, ten choices of the initial T were used anywhere between 128 and 500 also with increments of 10's.

To summarize, there were two data sets, each having 10 samples for each of their 10 parameter combinations. For each combination the *mean run time* as well as the *mean final energy* (cost) over the 10 sample runs were computed. If one parameter combination yielded a longer run time and higher final energy than another, it was discarded. The remaining combinations were those that were most efficient, that is, by construction they had the property that as time increases, the energy decreases.

8.2 Application-Specific Results

Our study case was quite large in comparison to the scheduling case used by Peterson et al. [20]. The largest case they had dealt with involved around 1000 students, 20 different possible majors, and an overall periodic schedules (over weeks). In the case of Abramson et al. [4], the problems they dealt with were quite small and they stated that problems involving more than 300 tuples were very difficult to solve.

Our study case involved real scheduling data covering three semesters here at Syracuse University. Those of first and second semester were roughly of the same size: around 13600 students, 1100 professors, staff, and teaching assistants, 400 rooms (classrooms, seminar rooms and auditoriums), and 2500 classes. With the exception of the number of rooms, the third semester data was much smaller, around 700 classes.

8.3 Observations and Analysis

The main objective here is to generate a solution (i.e. a feasible timetable) subject to all binding (or hard) constraints and to minimize the cost incurred form the medium and soft constraints. In regard to the geometric annealing, when we start from some good solution rather than from the usual random solution, the initial temperature must not be too high to avoid reaching equilibrium and "forgetting" the supplied initial solution. For totally random starts, this was found not to be of much help.

[§]A generation is a full Markov chain at a single temperature, (other stopping criteria are considered).

We also observed that the efficacy of starting annealing (using either adaptive or non-adaptive schedules) from a good solution depends not only on the energy (or cost) of that solution, but also on how it was obtained, that is, on its structure. For example, [16, 31] give few thoughts on this subject but as far as we know we have not come across any kind of analysis on the topic.

Our analysis starts with Figure 4 which shows the relationship between $\log_2(T)$ and cost for the geometric cooling and the RFC method. The interesting thing to notice is that the cost begins to decrease rapidly around $\log_2(T)$ of 3.5 and continue decreasing picking up speed approximately at 0. Figure 3 shows the range of the maximum specific heat which is between 0 and 1. As can be seen, after some fluctuation the specific heat climbs up above 0.4 and reaching a maximum just below 0.45. Unlike the relationship between $\log_2(T)$ and the RFC cost, we could see that this relationship is very much typical for the geometric cooling. The cost continues to decrease, albeit slowly, beyond the point shown in the plot till reaching close to 3000. After that the rate of change becomes even slower with either taking very short upward moves or slightly going down but no sharp downward moves, as partly shown in Figure 5. This pattern continues for at least 10000 more iterations till termination or reaching a stopping point. Also, as shown in Figure 6, in the first 300 or so iterations very little reheating was done since the cost is a bit high. Reheating becomes more frequent as it is shown between steps 400 and 800. As the decrease in cost becomes less and less or no noticeable decrease at all, the frequency of reheating increases sharply as observed around the 1100 iteration step.

Generally, factors contributing to reheating are as follows: The drop of T below the maximum specific heat combined with a slow, no decrease, or an increase in cost. To be precise, as T drops more and more below the maximum specific heat and after a number of specified iterations, if either cost does not decrease significantly or start to increase, reheating immediately will occur in an attempt to escape from that cycle.

Our overall results are shown in tables (1, 2, 3, and 4). Table 1 gives a percentage output of the preprocessor (the rule-based system) for a three-semester set of data as well as the highest and lowest percentage of scheduled data. We also can see that the size of those scheduled classes increases as the input (number of classes) decreases, which was not really surprising since the overall number of rooms and time slots were kept fixed for the three semesters. In addition, both class and students constraints were taken into consideration. Unlike Table 1, Table 2 shows improved results and that was mainly due to not taking students constraints into consideration and only restricting the system to class constraints.

In the case of simulated annealing, Table 3 shows output in a percentage form averaged over 10 runs for only the first semester input and without a preprocessor. Clearly, it shows a rather poor output which is even lower in quality to that of the rule-based system. Also, the use of the RCF schedule did not improve the results by much and remained below those of Table 1.

In contrast to the first three tables, Table 4 shows quite improved and high quality averaged results for both annealing schedules. Such excellent results were mainly due to the use of preprocessed input (output of the rule-based system) to the SA processor. Also the figures were averaged over 10 runs of annealing and one out of those runs yielded a perfect schedule, that is, all classes of the input set were scheduled subject to the given students and class constraints. Clearly, this tells us that the use of preprocessing was not only helpful but quite essential when using methods such as simulated annealing to deal with complex timetabling problems. Furthermore, these high quality averaged results state in an unambiguous terms that a multi-phase approach, as we explained above, gives a much better results to TTP then a single-phase approach.

In regard to runtimes for the first semester, the 10-run average of the preprocessor (the expert system) was over two hours. While it took an average of 22 hours to carry out the processing using the geometric cooling schedule and about 16 hours when using the reheating schedule. For the second semester schedule, it took a 10-run average of an hour and 50 minutes of preprocessing and an average of almost 20 hours of proceesing using the geometric schedule versus about 15 hours when reheating was used. These runtimes are clearly much faster than the 30 days or so time period it usually takes for Syarcuse University to produce a final schedule (students are not included) for a single semester. The batch scheduling system used is called '25E' and the overall scheduling process invloves a number of interactive sub-processes taking place between the central scheduling office and the various schools and departments of the university. The purposes of these interactions are to resolve any conflicts arising from class/room assignments and to satisfy all categories of constraints in a piecemeal fashion. Also, when scheduling a certain semester (e.g. autumn 1993), a template of a previous semester (e.g. autumn 1992) is used as part of the input data, otherwise, the entire process of course scheduling for a single semester will certainly take much longer than a month.

8.4 Code and Simulation Times

The code of the aforementioned two annealing schedules are written in ANSI C and all the simulations were run using the GNU C compiler on the IBM RS6000 and the SP2. In regard to the SP2, individual jobs were run separately on different nodes using different sets of parameters and if needed also different cooling schedules.

9 Summary and Conclusion

This part outlines the basic method and the scheduling constraints. An academic scheduling problem that is based upon equality constraints is considered. We have used simulated annealing with two types of cooling schedules to produce our target timetabling schedules. This target need to be a valid timetable of classes, professors, and students with no violations at all of the hard constraints and preferrabaly satisfying most , if not all, of the soft constraints. The rule-based system which is used as pre-processor for the simulated annealing contributed greatly to the improvement of the final outcome.

We plan to carry out more experiments using perhaps other cooling schedules and an improved pre-processor system.

— This section needs some workout —

Acknowledgments

The authors are very grateful for the valuable help of Robert Irwin in converting and formatting the registration data prior to the scheduling process. We also would like to sincerely thank Karen Bedard for providing us with the data and answering so many questions we had about it, and Meg Cortese for providing us with a set of building constraints and preferences for various departments. Thank you Karen and Meg.

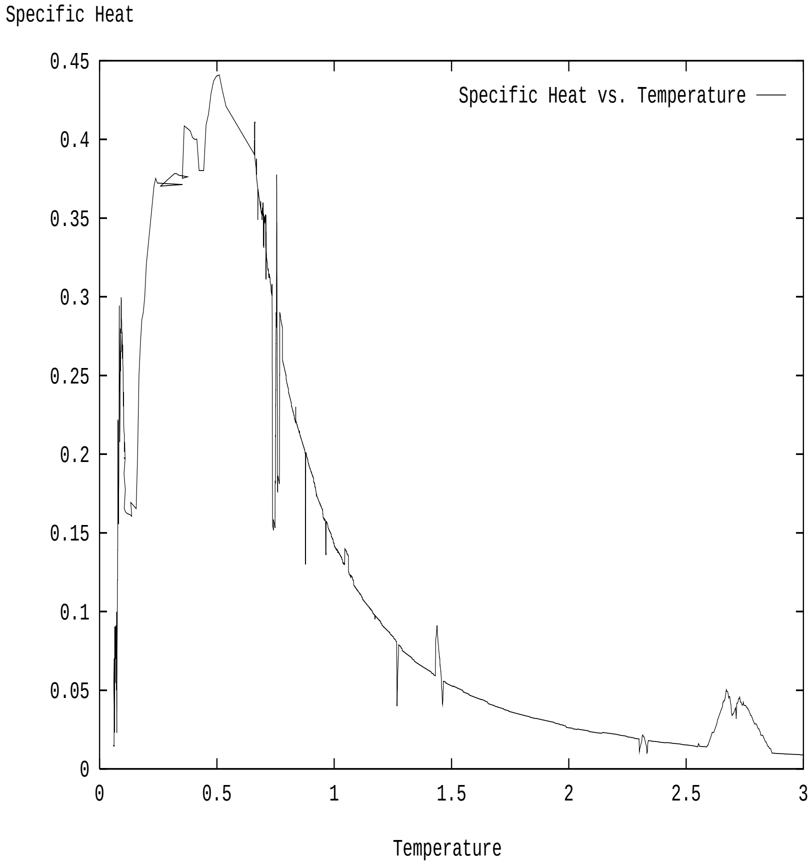

Figure 3: Specific Heat vs. Temperature for the reheating based on cost schedule

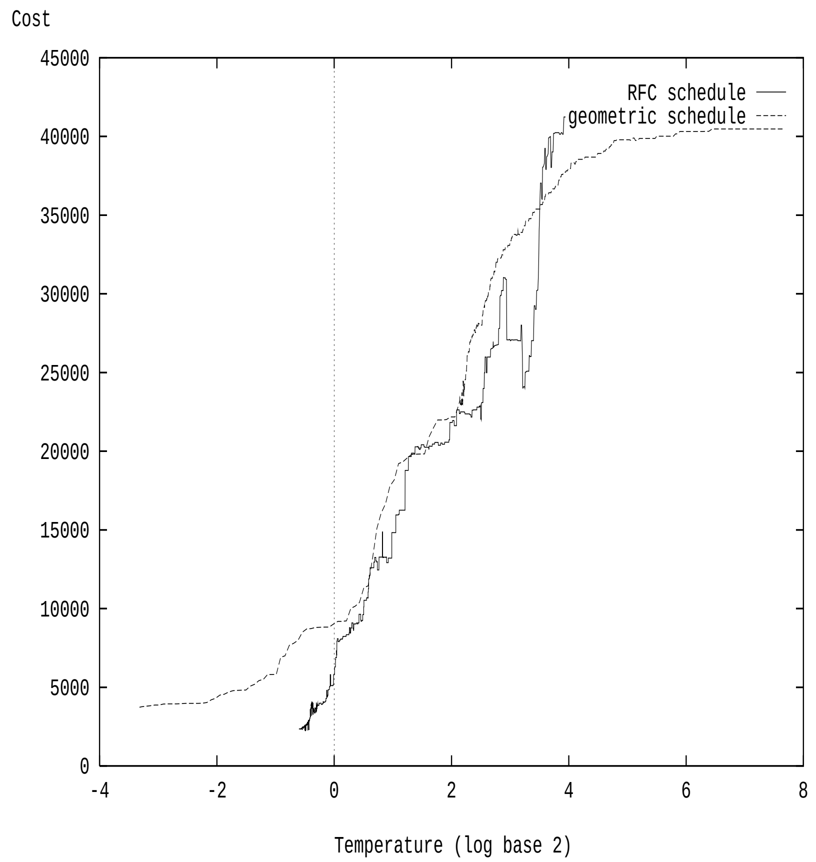

Figure 4: Cost vs. Temperature (log base 2) for the geometric annealing and the reheating based on cost schedules

	Total Courses	Scheduled No.	Not Scheduled No.	Scheduled (average) %	highest Scheduled %	lowest Scheduled %
First Semester	2482	1688	794	68.00	71.75	66.95
Second Semester	2164	1527	637	70.57	72.85	68.50
Third Semester	709	569	140	80.25	85.35	77.95

Table 1: Averaged figures for three terms using a *rule-based system* only. Both class and students constraints are used.

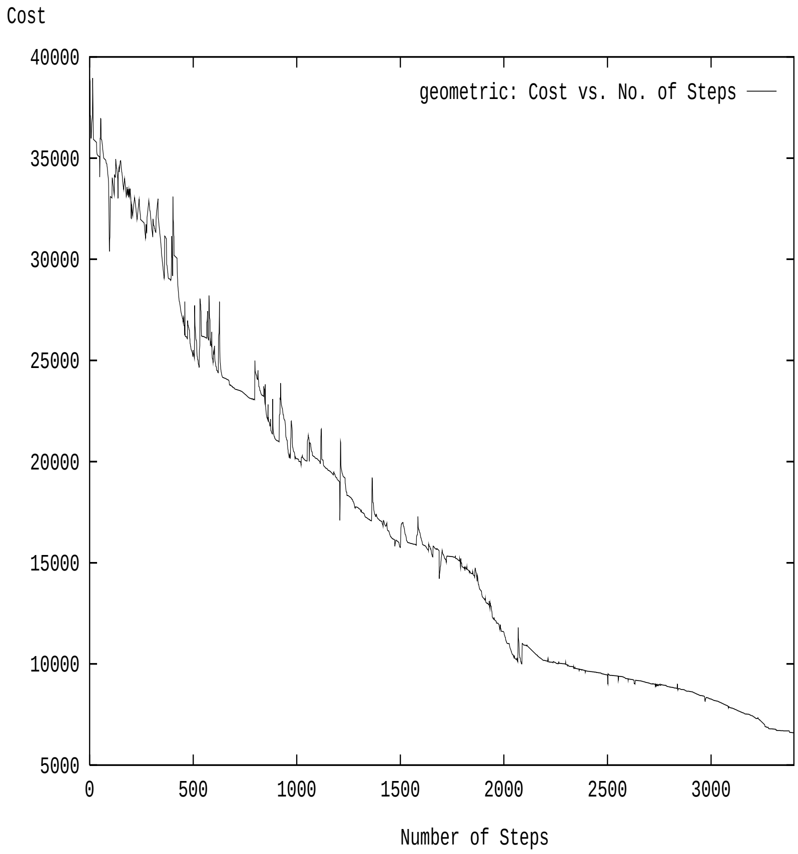

Figure 5: Cost vs. Number of Steps for the geometric annealing

	Total Courses	Scheduled No.	Not Scheduled No.	Scheduled (average) %	highest Scheduled %	lowest Scheduled %
First Semester	2482	1787	695	72.00	75.15	69.55
Second Semester	2164	1690	474	78.12	81.10	75.00
Third Semester	709	631	78	89.00	90.45	88.90

Table 2: Averaged figures using a *rule-based system* (*expert system*). Only class constraints are used.

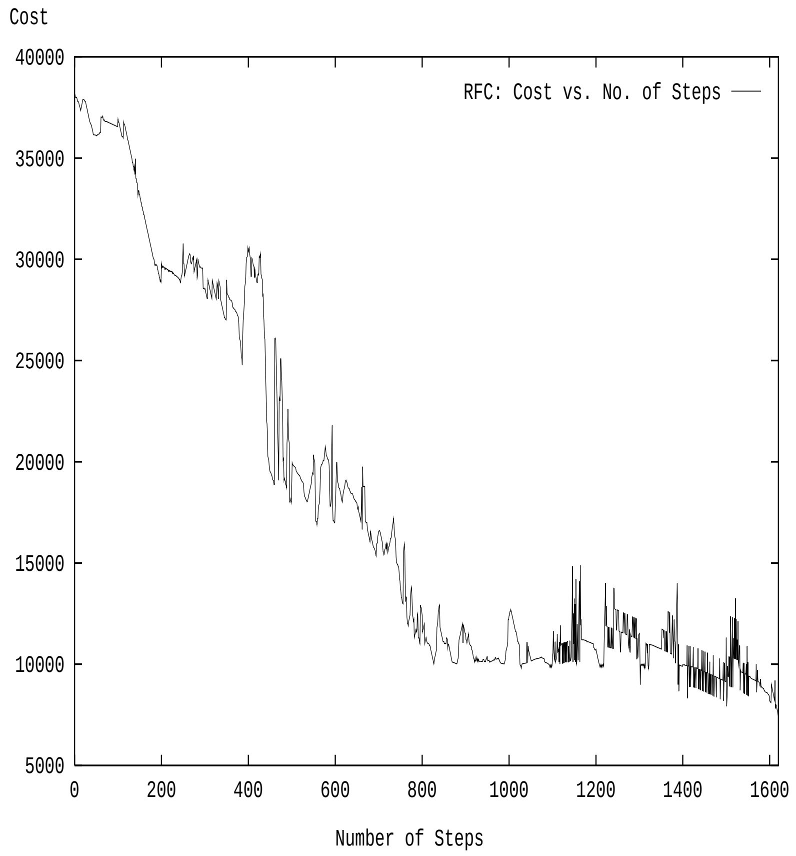

Figure 6: Cost vs. Number of Steps for the reheating based on cost schedule

	Scheduled No.	Not Scheduled No.	Scheduled (average) %	highest Scheduled %	lowest Scheduled %
SA (geometric)	1465	1017	59.70	67.95	55.28
SA (cost-based)	1644	838	66.22	70.47	61.12

Table 3: Averaged figures for two algorithms with no pre-processor used. Both students and class constraints are used and the data is that of the first term. *Note*: Annealing is carried out on space AND time. Also, all shown figures are averaged over 10 runs using first semester data only.

	Scheduled No.	Not Scheduled No.	Scheduled (average) %	highest Scheduled %	lowest Scheduled %
SA (geometric)	2407	75	97.27	98.25	94.45
SA (cost-based)	2444	38	98.47	100.0	95.10

Table 4: Averaged figures for two algorithms using an expert system as a pre-processor. Both students and class constraints are used and the data is that of the first term. *Note*: Annealing is carried out on space AND time. Also, all shown figures are averaged over 10 runs using first semester data only.

Appendix A

From [13], let $\mathbb{R}^n$ be the space of all n-tuples $(x_1, \ldots, x_n)$ with real entries x_i, and $\mathbb{R}$ is the associated set of scalars. Likewise for $\mathbb{R}^m$, where $m \neq n$.

- A subset P of $\mathbb{R}^n$ is a polyhedron if and only if there exists an m by n matrix A and a vector $b \in \mathbb{R}^m$ such that: $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$. Here, $Ax \leq b$ means: $a_1 x \leq b_1, \ldots, a_m x \leq b_m$, where $a_1, \ldots, a_m$ are the rows of A. The matrix A may have zero rows, i.e. $m = 0$. In this case $P = \mathbb{R}^n$.

- A subset P of $\mathbb{R}^n$ is called a polytope if P is the convex hull of a finite number of vectors. That is, there exist vectors $x_1 \ldots u_t \in \mathbb{R}^n$ such that: $P = conv.hull\{x_1 \ldots x_t\}$. Also, a subset P of $\mathbb{R}^n$ is a polytope iff it is a bounded polyhedron. For more information, see references [12, 13].

- A polytope P is called integral if each of its vertices is an integer vector. Clearly, if a polytope $P = \{x|Ax \leq b\}$ is integral, then the LP-problem $max\{c^T x|Ax \leq b\}$ has an integer optimum solution. So in that case,

$$max\{c^T x|Ax \leq b,\ x \text{ integer } \} = max\{c^T x|Ax \leq b\}\ . \tag{A1}$$

References

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

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

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

[4] D. Abramson, H. Dang, and M. Krishnamoorthy, *An Empirical Study of Simulated Annealing Cooling Schedules*, Griffith Univ. report, Nathan, Qld, Aus. 1994.

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

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

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

[8] Ladkin, P.B., and Maddux, R.D., “On binary constraint problems,” *Journal of the ACM* 41 (1994), pp. 435-469.

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

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

[11] Sorkin, Gregory, *Theory and Practice of Simulated Annealing on Special Energy Landscapes*. PhD Thesis, UC-Berkeley, California, 1992.

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

[13] Shrijver, Alexander, A Course in Combinatorial Optimization, to appear in 1996.

[14] Forest, Richard, *Ensemble of Simulated Annealing*, SDSC, San Diego, California, 1994.

[15] Van Laarhoven, and Aarts, Simulated Annealing: Theory and Applications. D. Reidel, Dordrecht (1987).

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

[17] David Johnson, "Local Optimization and the TSP," *Springer Lecture Notes in Computer Science* 443, 446-461, Springer-Verlag, 1990.

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

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

[20] L. Gislén, B. Söderberg, C. Peterson, "Scheduling High Schools with Neural Nets," Lund University Preprint LU-TP-91-9, Lund, Sweden (1991).

[21] C. Peterson, B. Söderberg, "Artificial neural Nets and Combinatorial Optimization Problems," Local Search in Combinatorial Optimization, Aarts and Lenstra (eds.), John Wiley and Sons, UK (first half of 1996).

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

[23] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Science* USA 81, 3088-3092 (1984).

[24] A. Colorni, 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, Milano, Italia.

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

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

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

[28] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling salesman problem", *Operations Research*, 21 (1973), 498-516.

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

[30] J. Paulli, "A computational Comparison of Simulated Annealing and Tabu Search Applied to the Quadratic Assignment Problem," in Applied Simulated Annealing, R. Vidal (ed), Springer-Verlag LNEMS 396, 1993.

[31] David Johnson and Lyle 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.), John Wiley and Sons, UK (to appear in 1996).

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

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

[34] Michael Carter, "A Lagrangian Relaxation Approach to The Classroom Assignment Problem," *INFOR* vol. 27, no.2, 230-246, May 1989.

[35] N. Boissin, and J.-L. Lutton, "A parallel simulated annealing algorithm," *Parallel Computing* 19, 859-872, 1993.

[36] M. G. A. Verhoeven, and Emile H. L. Aarts, "Parallel local search techniques," Manuscript, Philips Research Laboratories, Eindhoven, The Netherlands, 1994.

[37] Emile Aarts, Jan Korst, and Peter von Laarhoven, "Simulated annealing," in Local Search in Combinatorial Optimization, E. H. Aarts and J. K. Lenstra (eds.), John Wiley and Sons, UK (to appear in 1996).

[38] M. Carter, 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).

[39] M. Huang, 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.

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

[41] R. Diekmann, 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.

[42] Andrew Gee, private communication.