

JOINT ROYAL MELBOURNE INSTITUTE OF TECHNOLOGY AND  
COMMONWEALTH SCIENTIFIC AND INDUSTRIAL RESEARCH ORGANISATION  
PARALLEL SYSTEMS ARCHITECTURE PROJECT

## Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms

TR 112 069 R

*D. Abramson* †

† Division of Information Technology  
C.S.I.R.O.  
c/o Department of Communication and Electronic Engineering  
Royal Melbourne Institute of Technology  
P.O. Box 2476V  
Melbourne 3001  
Australia.

Version 2.0 January 1989

### **ABSTRACT:**

This paper considers a solution to the school timetabling problem. The timetabling problem involves scheduling a number of tuples, each consisting of class, a teacher, a subject and a room, to a fixed number of time slots. A Monte Carlo scheme called simulated annealing is used as an optimisation technique. The paper introduces the timetabling problem, and then describes the simulated annealing method. Annealing is then applied to the timetabling problem. A prototype timetabling environment is described followed by some experimental results. A parallel algorithm which can be implemented on a multiprocessor is presented. This algorithm can provide a faster solution than the equivalent sequential algorithm. Some further experimental results are given.



## 1. INTRODUCTION

This paper considers a solution to the school timetabling problem. The timetabling problem involves scheduling a number of tuples, each consisting of class, a teacher, a subject and a room, to a fixed number of time slots. A number of such tuples may be scheduled in the same time slot providing:

- 1) no class appears more than once in the time slot,
- 2) no teacher appears more than once in a time slot and
- 3) no room is used more than once in a time slot.

This problem has been well studied in the past and some limited success has been reported. An exhaustive search is impracticable because there are too many alternatives. Modelling the problem as an integer programming problem has not been particularly successful because there are too many variables and constraints [2,3,4,6,7,8]. Heuristic searches have had reasonable success, but it is hard to form a heuristic that performs as well as an experienced human [1]. Monte Carlo techniques have been used to a limited extent [5,9]. In this paper we examine the use of a Monte Carlo scheme called simulated annealing as an optimisation technique. We introduce the timetabling problem, followed by the simulated annealing technique. We then show how annealing can be applied to the timetabling problem. We describe a prototype timetabling environment, and give some experimental results. The timetabling package which has been developed is based on the theory given in this paper. It can be used by timetable planners in preparing a usable schedule of classes and teachers. The data is entered as a set of requirements and constraints, as illustrated in section 11. The package manipulates the positions of the various classes until it has minimised the number of clashes in the timetable. The output is either a valid schedule, or one which is as close as possible to a valid schedule. The various input constraints are described in section 5.

A parallel algorithm which can be implemented on a multiprocessor is presented. This algorithm can provide a faster solution than the equivalent sequential algorithm. Some further experimental results are given.

The timetabling problem is important not only because many schools and universities produce timetables each year, but also because it is just one of many scheduling problems. An effective solution to the timetabling problem could be applied to other constraint based scheduling tasks. The parallelisation of the serial algorithm could be effective as a technique for other simulated annealing algorithms.

## 2. TIMETABLING PROBLEM

The problem of creating a valid timetable involves scheduling classes, teachers and rooms in such a way that no *teacher, class or room* is used more than once per period. For example, if a class must meet twice a week, then it must be placed in two different periods to avoid a clash. The timetable is to be distributed across a fixed number of periods per week. A class consists of a number of students. We will assume that students have already been grouped into classes. Initially we consider classes to be disjoint, that is, they have *no students in common*. In this scheme, a correct timetable is one in which a class can be scheduled concurrently with any other class. Later in the paper this restriction will be relaxed to cater for high schools and Universities in which students can take many options. In each period a class is taught a *subject*. It is possible for a subject to appear more than once in a period. A particular combination of a teacher, a subject, a room and a class is called an *element*. An element may be required more than once per week. The combination of an *element* and a frequency is called a *requirement*. Thus, the timetabling problem can be phrased as scheduling a number of requirements such that a requirement, teacher, class or room does not appear more than once per period. The task of forming requirements is handled separately and will not be discussed further. A sample timetable is shown in Figure 1. A CLASS CLASH is shown in period 1 Monday. In this period class 1 appears more than once. There is no scheduling problem between Periods 1 and 2 because they are different time slots.

It is possible to define an *objective* or *cost function* for evaluating a given timetable. This

function calculates the number of clashes in any given timetable. An acceptable timetable has a cost of 0. The cost of any period is the sum of three components:

- 1) a class cost,
- 2) a teacher cost and
- 3) a room cost.

The class cost is the number of times each of the classes in the period appears in that period, less one if it is greater than zero. Thus, if a class appears no times or once in a period then the cost of that class is zero. If it appears many times the the class cost for that class is the number of times less one. The class cost for a period is the sum of all class costs. The same computation applies for teachers and rooms.

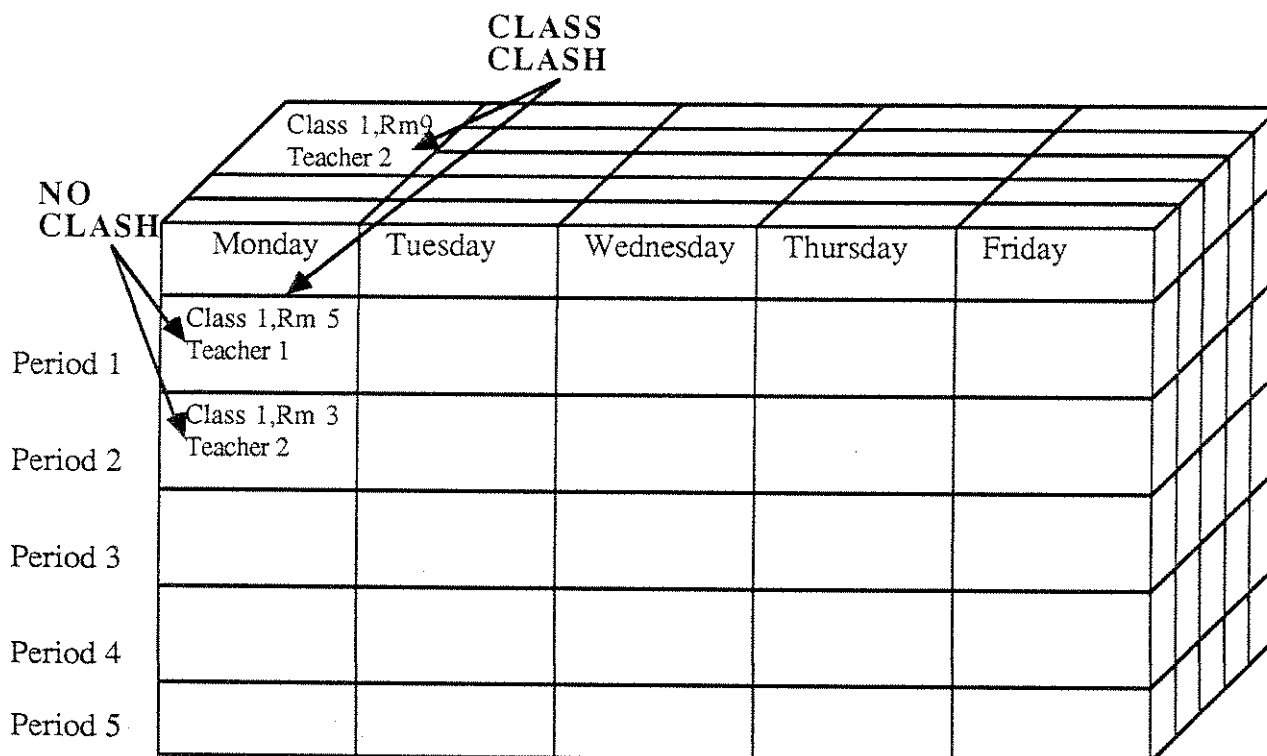


Figure 1 - A sample timetable

The cost of the total timetable is the sum of the period costs.

### 3. SIMULATED ANNEALING

Simulated annealing is a Monte-Carlo technique which can be used to find solutions to optimisation problems. A good review of the theory and practice can be found in [10]. The technique simulates the cooling of a collection of hot vibrating atoms. When the atoms are at a high temperature they are free to move around, and tend to move with random displacements. However, as the mass cools the inter-particle bonds force the atoms together. When the mass is cool, no movement is possible, and the configuration is frozen. If the mass is cooled quickly then the final system energy may not be minimal. However, if it is cooled slowly, then the final energy may be the lowest possible. At any given temperature a new configuration of atoms is accepted if the system energy is lowered. However, if the energy is higher, then the configuration is accepted only if the probability of such an increase is lower than that expected at the given temperature. This probability is given by  $P(\Delta E) = e^{-\Delta E/KT}$ , where  $K$  is Boltzmann's constant.

By modelling optimisation problems as a set of randomly vibrating atoms, it is possible to find optimal, or sub-optimal, solutions. Many optimisation problems can be considered as a number of *objects* which need to be scheduled such that an objective function is minimised. The vibrating atoms are replaced by the objects, and the value of the objective function replaces the system

energy. An initial schedule is created by randomly scheduling the objects, and an initial cost ( $c_0$ ) and temperature ( $T_0$ ) are computed. Subsequent permutations are created by randomly choosing two objects, interchanging them, and computing a change in cost ( $\Delta c$ ). If  $\Delta c \leq 0$  then the change is accepted. However, if  $\Delta c > 0$  then the probability of that change is calculated,

$$P(\Delta c) = e^{-\Delta c/T}$$

If the probability is greater than a randomly selected value in the range (0,1) then the change is accepted. After a number of successful permutations the temperature is decreased by a cooling rate,  $R$ , such that  $T_n = T_{n-1} * R$ .

One of the advantages of simulated annealing over algorithms which always seek a better solution (hill climbing algorithms) is that simulated annealing less likely to get caught in local minima, because the cost can increase as well as decrease.

#### 4. APPLYING SA TO THE TIMETABLING PROBLEM

##### 4.1 Technique

The application of simulated annealing to the timetabling problem is relatively straight forward. The atoms are replaced by elements. The system energy is replaced by the timetable cost. An initial allocation is made in which elements are placed in a randomly chosen period. The initial cost and an initial temperature are computed. The cost is used to reflect the quality of the timetable, just as the system energy reflects the quality of a substance being annealed. The temperature is used to control the probability of an increase in cost and relates to the temperature of a physical substance. At each iteration a period is chosen at random, called the **from** period, and an element randomly selected from that period. Another period is chosen at random, called the **to** period. The change in cost is calculated from two components:

- 1) The cost of removing the element from the **from** period
- 2) The cost of inserting the element in the **to** period.

The change in cost is the difference of these two components. The element is moved if the change in cost is accepted, either because it lowers the system cost, or the increase is allowed at the current temperature. Unlike the classic simulated annealing technique which would actually swap two elements, an element is removed from one period and placed into another. This allows the number of elements in one period to increase or decrease, and for all periods to contain different numbers of elements. If two elements were swapped then it would not be possible to change the number of elements per period.

The cost of removing an element consists of a class cost, a teacher cost and a room cost. Likewise, the cost of inserting an element consists of a class cost, a teacher cost and a room cost. If after removing an element from a period the number of occurrences of that class is  $> 0$ , then the class cost saving is 1. Similarly, if there are one or more occurrences of the teacher after that teacher has been removed then the teacher saving is 1. This technique also applies for rooms. The cost of inserting an element can be calculated using the same basic technique. In this way it is possible to determine the change in cost incrementally without recalculating the cost of the entire timetable. This attribute is particularly useful when the parallel version of the algorithm is implemented.

Since the simulated annealing algorithm relies on a random set of permutations it cannot guarantee that the true minimum cost value is actually found, or that two different annealing runs will yield the same solution. It is possible to guarantee reproducible results by using a pseudo-random number generator so that the same random sequence is generated for a given *seed* value. Thus, different sequences can be generated by starting the random number generator with a different seed value each time. It is often necessary to perform multiple runs to determine whether the cost is the best that can be achieved. Some of the results presented in this paper were produced by running the annealing program many times, each time with a different random number seed. The global minimum cost solution was then chosen.

## 4.2 Run Time Parameters

There are a number of parameters which govern the cooling schedule. A good description of these can be found in [10]. The following values must be computed:

- (1) Starting temperature
- (2) When to decrease the temperature - MAXSWAPS and MAXSUCCESSSWAPS
- (3) Termination of Run
- (4) Cooling schedule - COOLINGRATE

The initial starting temperature should be sufficiently high to allow most increased in cost. A technique for choosing the initial temperature is described in [11], and involves computing the standard deviation in the changes in cost over a number of swaps. This technique allows the initial temperature to be high enough to allow most swaps, but not too high that many unnecessary swaps are required.

Each time the temperature is decreased a number of swaps is attempted. It is necessary to limit the number of unsuccessful swaps at each temperature as well as the number of successful swaps. These two limits are called MAXSWAPS and MAXSUCCESSSWAPS, and are chosen to be proportional to the number of elements. Thus, as the problem grows, the time allowed to find a solution also increases. At any given temperature, the number of successful swaps is limited to MAXSUCCESSSWAPS. The total number of successful and unsuccessful swaps is limited to MAXSWAPS.

There are two conditions for terminating a run. The first is if the cost becomes zero. In this case an ideal timetable has been computed, and there is no point continuing. The second condition arises if the cost has not changed for a certain number of iterations. The program implemented allows an arbitrary number of temperature changes without a change in cost before it concludes that the configuration has frozen.

A simple cooling schedule is chosen. A new temperature is computed by multiplying a COOLINGRATE by the current temperature.

## 5. PRACTICAL CONSIDERATIONS

There are a number of practical problems associated with the simple timetable model described in the previous section. These are:

- 1) It cannot handle classes which have students in common,
- 2) It is not flexible in assigning rooms to elements,
- 3) It does not allow multiple periods,
- 4) It does not allow one class to be scheduled always with another class.
- 5) It does not allow one type of clash to be more important than another.

### 5.1 Class Clashes

In the simple model classes are assumed not to clash with any other class. Thus, the cost of scheduling a class to a period can be calculated by checking if that class is already scheduled in the period (likewise for the teacher and the room). Such an assumption is impracticable in real timetables. For example, year 7 may be divided into three groups, 7A, 7B and 7C. Each group contains different students, and thus they can be scheduled together. However, when the subject ART is taught to year 7 it may be taught to some selected students from 7A, 7B and 7C. If we call this class 7D, then there is no way of indicating that 7D cannot be taught simultaneously with A, B or C.

The solution involves maintaining a class clash list for each class, and using this information in computing the cost of a period. For example, the cost should reflect that if 7D appears in the same period as 7A then a clash has occurred. This clash information can be included in the simple cost

function relatively easily.

## 5.2 Room Assignment

Static assignment of rooms to requirements removes some of the flexibility in scheduling the timetable. For example, if a school has a number of rooms each of which hold 30 students, then there is no reason to statically assign a particular class, subject and teacher to any one room. It would be possible to simply choose one room from those available. This problem is solved by using *room groups*. A requirement is assigned to a room group, which may contain more than one room. When the element is scheduled to a period, the room cost does not increase until all of the rooms in the room group have been assigned. Thus, if there are 20 rooms which can hold 30 students each, then the room cost does not increase for this room group until 21 classes are scheduled for the group. It is not necessary to actually assign the rooms when the element is scheduled, but simply to decrement the number available in the group. The rooms can then be assigned once a feasible timetable has been generated.

Flexible room assignment may be performed in which a room is contained in more than one room group. In this way, it is possible to request that classes which need special purpose rooms receive them, but if the special purpose rooms are not required in a period then they can be used for general subjects. The only added complication with this operation is that a requirement which requires a room which is also in a room group needs to also claim a room from the room group when it is scheduled.

## 5.3 Multiple Periods

Practical timetables often call for a class to be scheduled over more than one consecutive time period. For example, practical subjects often need 2 or 3 hours of contiguous time. The simple model described takes no account of multiple period assignments; a perfect timetable is one in which there are no clashes. It is possible to include multiple periods by introducing a cost measure which increases the timetable cost if the periods of a multiple period are not consecutive. In this way a perfect timetable not only has no clashes, but also all multiple period assignments are satisfied. If a new configuration wishes to split a multiple period request, then it is only accepted if the change in cost is allowed at the given temperature.

## 5.4 Groups of Requirements

The complexity of the scheduling task can be greatly reduced by removing unnecessary requirements. If it is known that a set of classes are always scheduled together in the same period, then there is no need to have a separate requirement for each one. For example, a year level may take a number of art/craft subjects, each of which is a separate class. However, such elements are always scheduled together in the same period because they involve a complete set of students. While the cost function will indicate that such requirements can be scheduled together, there is no fixed requirement that they are. If the scheduler knows to always allocate a group of requirements to the same period, then the entire group is moved when one is moved. The change in cost reflects moving all of the individual elements.

## 5.5 Weighted Cost Function

When creating a timetable certain scheduling requirements may be more important than others. For example, it may be more important that no class appears more than once in a period than a room is used more than once, because it may be possible to find a spare room after the timetable has been generated. This *preference* can be expressed by weighting the components of the cost function. Thus, the cost function is the sum of the weighted cost values for class cost, teacher cost, room cost and multiple period cost. If any one component is more important than another then the weight can be increased appropriately.

## 5.6 Preferences for periods

It may be necessary to express a preference that a requirement appear in a particular period, or range of periods. This information can be easily added into the cost function, by including a cost component which reflects how close a requirement is to its desired period. When a requirement is close to its preferred period the cost is low, however, as the requirement moves further from the period the cost is increased. A preference for more than one period may be indicated, so that a number of preferred times can be nominated. This cost component can be computed very quickly by using a statically computed cost table, which indicates the cost of placing a requirement in every period of the timetable.

## 5.7 Limiting number of periods taught

Many schools wish to express a preference that a teacher is not scheduled more than a certain number of times in a given period range. For example, a particular teacher may not wish to teach more than 3 period on a Friday. This request can be easily incorporated into the cost function by keeping track of how many times a teacher has been scheduled in a given period. The cost computation requires more than one period of the teacher clash array to be consulted when a requirement is moved.

## 6. SOME EXPERIMENTAL RESULTS

In this section we show some results of the application of simulated annealing to some randomly constructed data and some real data from an Australian High School. The simulated annealing algorithm has been implemented with a 1300 line Pascal program. The complexity of the code is much less than an equivalent (incomplete) heuristic search program which required about 4000 lines of Pascal code [12].

The program was tested with two types of data. One type is generated automatically by a program which creates a random timetable with zero cost. The timetable is generated by choosing a class, teacher and room combination from a random pool, and then placing it in a period which does not increase the cost of the configuration. This technique means that there is always *at least* one solution to the timetable, although there may be more than one legal solution.

The second type of data was taken from an existing timetable in a secondary high school. This data was much more complex, and needed room groups, clashing classes and groups of requirements. The results of these experiments are summarised in Table 1. Timetables 1 through 9 are random timetables with no class clashes, multiple periods or grouped requirements. Timetables 1 through 5 do not use grouped rooms, that is, each room group only has one room available. Timetables 6 through 9 use grouped rooms, as described in the notes below Table 1. Timetable 10 was taken from real high school. Timetable 10 used groups of requirements, clashing classes and room groups.

It is difficult to design a measure of timetable complexity because there are so many variables. In general, the complexity is proportional to the number of elements, and inversely proportional to the numbers of rooms, teachers and classes, although the function is clearly not linear. However, grouped requirements, room groups and class clash lists all alter the complexity of any given data set. Consequently, a simple and effective method of computing the complexity of each of these timetables is to calculate the **initial cost**. Since the initial configuration is random, the initial cost gives quite a good measure of "how hard the timetable is to schedule". If there is a great degree of flexibility in the possible assignments, then the initial cost is likely to be low. However, if there is very little freedom, the initial cost will be high. The use of initial cost as a complexity measure automatically takes into account the number of elements to be scheduled, the number of periods available, the number of teachers, rooms and classes, and the difficulty of scheduling classes which clash with each other.



Test Data Set	Number Teachers	Number Rooms	Number Classes	Number Elements	Initial Cost	Final Cost	Cooling Rate	Execution Time on Sun 3/60
1	15	15	(a) 15	100	32	0	0.9	11 secs
2	15	15	(a) 15	150	62	0	0.9	26 secs
3	15	15	(a) 15	200	116	0	0.9	50 secs
4	15	15	(a) 15	250	177	0	0.9	89 secs
5	15	15	(a) 15	300	229	0	0.9	131 secs
6	36	24	(b) 46	400	186	0	0.9	140 secs
7	36	24	(b) 46	600	344	0	0.9	238 secs
8	30	24	(b) 30	600	410	0	0.9	296 secs
9	20	24	(b) 20	600	506	3	0.99	5548 secs
10	37	24	(b) 101	757	810	0	0.95	14 <u>hours</u>

## Notes:

- (a) indicates that all room groups contained only 1 room
  - (b) indicates that there were 3 rooms in one group, 14 rooms in another group and all other groups contained one room.
- All timetables contained 30 periods.

Table 1 - Results of Test Data

The results are shown in Table 1. The number of periods per week, teachers, rooms, classes and elements is shown for each data set. The initial cost gives an indication of the complexity of the schedule. The final cost indicates whether the program found a correct solution. For the randomly constructed data, the final cost should be zero. The cooling rate shows how quickly the temperature was decreased. The execution time on a Sun 3/60 computer is shown for comparison with other scheduling techniques.

The results show that increasing the number of elements for a given number of classes, rooms and teachers increased the complexity of the problem. In all cases more elements produced a higher initial cost. The numbers used in tests 1 through 5 were thought realistic for a small school timetable, and those for tests 6 through 9 for a typical high school. Unfortunately, the cost of Test Data 9 could only be reduced to 3, and the true minimum (0) could not be found. However, this test data is extremely complex, and whilst an optimum value is possible, the data has very little flexibility in the way it must be scheduled. (This is evident because 20 classes with 30 periods has a theoretical maximum of 600 requirements for a zero cost solution. Further, there are only 20 teachers, and 24 rooms.) Data set 10 reached its theoretical minimum cost, but only with a very long cooling cycle. Many solutions with non zero costs could be found with much faster cooling cycles. For example, a timetable with a few clashes could be found in a few hours using a faster cooling rate.

Table 2 contains another set of set data and results. This data was constructed using the random timetable builder program, but the parameters were taken from a number of real school timetables produced in [1]. The only information available was the number of elements, teachers, classes and rooms for each data set. The actual distribution was not available. The percentage of correctly scheduled elements for the heuristic used in [1] is shown, as well as the percentage achieved using simulated annealing. One complication which arises from not having the actual data is that the number of elements used in the real school data is often larger than theoretically possible given the number of classes, teachers and rooms. For example, it is not possible to have a cost free timetable of 860 elements as in data set 1 with only 26 classes and 30 periods, because there are only 780 unique class assignments. In practice the actual number can be larger than the theoretical limit because some of the elements contain *null* classes, teachers and rooms. These null assignments do not require an actual class, teacher or room, and therefore do not generate any clashes. Because this information was not available, a random timetable was generated with the theoretical maximum number of possible elements in such cases. These are shown in a separate column of the table. The table indicates that simulated annealing performed at least as well as the heuristic in the paper, if not

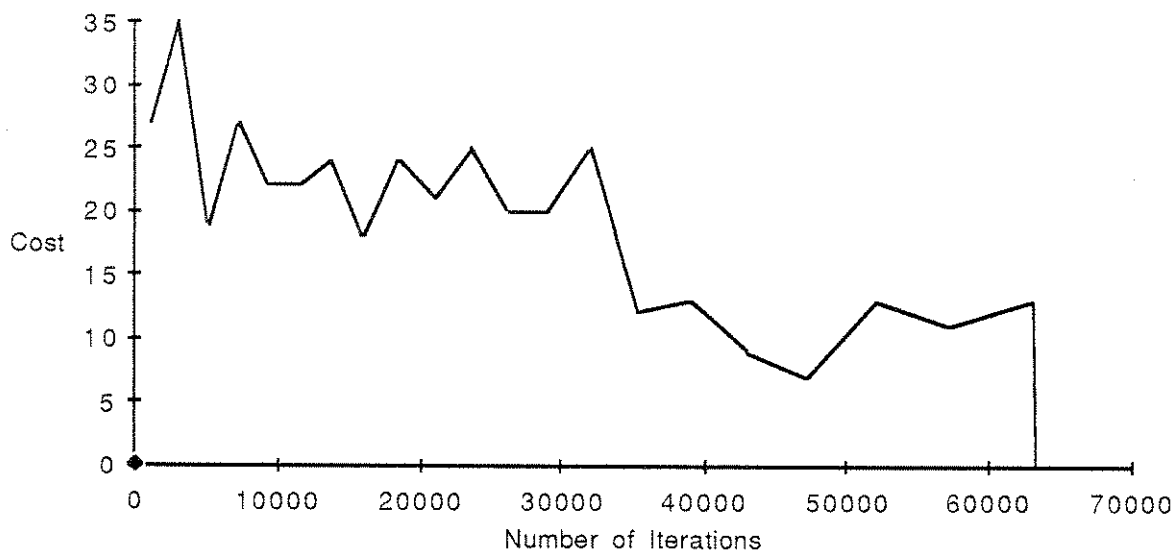
better. It should also be noted that each set of test data was only processed once. Thus, those solutions which did not have zero cost may not be the lowest possible cost. However, the comparison should be treated carefully because different data was used in the simulated annealing test data.

Test Data Set	Number Teachers	Number Rooms	Number Classes	Number Elements	Actual Elements	Initial Cost	Final Cost	% deGans assigned	% annealing assigned
1	44	28	26	860	780	723	1	99.4	99.8
2	75	59	72	1504	1279	1279	0	97.6	100.0
3	67	61	31	1463	930	718	0	94.2	100.0
4	58	39	27	1053	810	670	0	89.3	100.0
5	24	20	15	480	450	407	0	99.6	100.0
6	20	11	11	340	330	318	0	98.2	100.0
7	34	43	31	943	930	898	4	96.7	99.5
8	47	56	55	1398	1398	1372	5	99.2	99.6
9	20	30	25	588	588	550	0	99.2	100.0
10	71	48	41	1350	1230	1163	0	98.7	100.0
11	51	54	44	1302	1302	1312	1	99.8	99.9
12	77	58	48	1593	1440	1312	0	99.4	100.0
13	93	107	111	2252	2252	1922	0	98.3	100.0
14	44	19	28	484	484	339	0	97.3	100.0
15	71	52	33	1451	990	816	0	94.1	100.0
16	63	37	42	1114	1110	1029	0	96.6	100.0
17	65	59	91	1567	1567	1343	0	98.3	100.0
18	22	24	17	454	454	378	0	99.3	100.0
19	58	43	31	1029	930	817	0	97.4	100.0
20	65	51	29	1348	870	682	0	98.0	100.0
21	28	15	12	360	360	306	0	91.9	100.0

Note: All data was only annealed once  
30 Periods per week.

Table 2 - Comparison of deGans results to simulated annealing

Graph 1 shows a typical cooling schedule, in which the cost is plotted against the iteration count. This shows the cost decreasing, with increases allowed at high temperatures. At low temperatures (high iteration counts) the probability of an increase in cost is too low, and the algorithm resorts to seeking decreases in cost only.



Graph 1 - Plot of cost vs iteration count

## 7. A PARALLEL ALGORITHM

### 7.1 Simple Algorithm

The results shown in the previous section illustrate that while effective, simulated annealing can be extremely slow (for example, data set 10 of Table 1 took 14 hours of processor time). The speed of the simulated annealing algorithm described can be improved by using a parallel algorithm rather than a serial one. In the serial algorithm, each permutation of the elements is performed sequentially, and the new configuration either accepted or rejected. A new configuration is not generated until the previous one is performed. However, it is possible to perform multiple permutations concurrently, providing each permutation is independent of the other permutations.

A concurrent algorithm can be implemented by assigning multiple *processes* to the task of permuting the timetable. The timetable must be held in a *shared* memory area accessible to all processes. Each process independently chooses an element to move (from a **from** period), and a **to** period. In order to prevent other processes from choosing the same element and **to** period, they must *lock* the element. It is not actually desirable to lock the entire **to** period, as this would severely limit the number of concurrent swaps which were possible. Instead, they only need lock the *teacher*, *class* and *room* in the **to** period. Similarly, the *teacher*, *class* and *room* must be locked for the **from** period. These items must be locked so that no other process can effect the cost computation of a given process. The incremental cost computation technique allows a process to calculate the change in cost without recomputing the cost of the entire timetable. Once these items have been locked a process can determine the change in cost independently from all other potential swaps. If a process chooses an element, teacher, class or room which is already locked, then it must abandon the choice and try another.

The maximum number of concurrent processes depends on the size of the timetable. If there are too many processes for a given number of elements, then the number of processes abandoned swaps will be too high. Every time a choice is abandoned the effective speedup is decreased.

The *locks* described above can be implemented by simple read-modify-write variables in shared memory. A process can read a lock, and write a special marker value into the lock with an indivisible cycle. If the process reads the special lock value then it knows that the lock is already current and can abandon the choice. Such read-modify-write variables are not uncommon for multiprocessor machines. True semaphores are not required because the process does not wish to suspend when a lock is already claimed. Deadlock is not possible because a process *backs-off* any transaction which it cannot complete.

Implementing the parallel algorithm on a shared memory multiprocessor is relatively simple. The timetable must be held in shared memory, together with the lock variables. Once the timetable has been initialised the master process can *fork* and spawn as many child processes as necessary. Each child process permutes the timetable until the system is frozen, or the timetable has been solved. Each process can maintain its own temperature, or access a shared temperature variable. Similarly, each child process may share a common random number generator or maintain its own. If they use separate random number generators then each must use a different initial seed to avoid the same pseudo random sequences.

## 7.2 Complex Algorithm

The introduction of class clashes, multiple periods, room groups and groups of requirements all complicate the the parallel algorithm, and also decrease the amount of concurrency possible. The use of class clashes means that the clashing elements, rooms, teachers and classes must be locked for the **from** and **to** period as well as the element which was chosen. This is because a change to any of these elements can effect the cost computation performed by a process.

Multiple periods mean that the requirement must be locked for the period chosen as well as contiguous periods. This prevents a process from affecting the cost computation of another process.

Groups of requirements also demand that the original teacher/class/room combination is locked as well as the teachers, classes and rooms of the elements which are part of the same group. This prevents other processes from affecting the cost computation of the process.

The use of room groups poses the most serious problem for the parallel algorithm. In the simple algorithm the room is locked for the **from** period as well as the **to** period. In the complex algorithm the entire room group is locked. In a school which has a large number of rooms which are equivalent, this may effectly lock the entire period, severely reducing the potential concurrency. The solution to this problem involves relaxing the locking requirement when the number of free rooms is not near zero. When the number of free rooms is high, several concurrent processes may consume rooms and compute a cost change without affecting each other, providing they lock the room when they actually update the count of free rooms. In this way, a process may read the room clash counter without locking it, and then proceed to compute the potential change in cost. If the change is accepted then the room cost must be updated correctly by locking the room for the periods concerned. During the time that the room was not locked, the room cost could have been altered by another process. Providing the count is  $< 0$  then the permutation may still be accepted even though the cost value is different.

In spite of these complications, the parallel algorithm can be extremely effective in reducing the execution time.

## 8. EXPERIMENTAL RESULTS FOR PARALLEL ALGORITHM.

The parallel algorithm has been implemented on a conventional shared memory multiprocessor, a 10 processor Encore Multi-Max. Some of the test data from Table 1 (T1-1 through T1-8), and Table 2 (number T2-13) was presented to the parallel program, and the effective speedup was measured. The results are shown in Table 3. The execution time is shown for the purely serial code on the Encore, and then the parallel code using 1, 2, 4, 6 and 8 processors. The Peak speedup is defined as the time for the single process run divided by the smallest multiprocess time. The peak speedup for the small problems is low because there are not sufficient resources to keep the processors busy. In general, the larger the problem, the greater the speedup. It can be seen from these examples that whilst not ideal, the speedup in many cases is significant.

Test Data	Serial Time	Time in Secs for Number of Processors					Speedup
		1	2	4	6	8	
T1-1	43	41	20	16	13	13	3.2
T1-2	79	97	52	29	27	22	4.3
T1-3	139	180	87	54	38	33	5.4
T1-4	211	255	142	85	71	72	4.4
T1-5	390	729	409	218	157	137	5.3
T1-6	402	529	288	159	103	78	6.7
T1-7	807	842	528	256	165	150	5.6
T1-8	774	906	473	265	194	135	6.7
T2-13	5700	6900	3840	2100	1440	1020	6.8

Table 3 - Results of Parallel Execution

When evaluating a parallel algorithm, it is important not just to consider speedup, but also absolute speed. The efficiency of the parallel algorithm can be expressed as the time taken to solve the problem using the parallel code with one processor, divided by the time taken using a serial version of the program with one processor. A number of experiments have indicated that the parallel code varies from equal, to at worst two times slower than the serial version. Thus, in this worst case two processors are required before the parallel code overcomes the cost of the locking and synchronisation code. The serial time is shown in Table 3. In all of these examples the parallel version was only slightly slower than the serial code. Large timetable data sets could easily be expected to use up to 32 MIMD processors, providing a significant speedup. Further, it is possible to omit much of the locking code, which would reduce the cost of the parallel solution substantially. The times in Table 3 should not be compared to those of Table 1 because they have been produced on different computer systems.

## 9. A PROTOTYPE ENVIRONMENT

This section briefly describes the way that the scheduling program can be used. It describes a prototype tool set which has been developed.

Timetable specifications are entered into a text file, and are manipulated using a conventional text editor. A compiler accepts specifications and creates a number of files, which are then used by a scheduling program. Syntax errors, and illegal requests are reported if the specification is in error. These must be corrected before the scheduler can be run.

The compiler translates all teacher, class, room and subject names into numeric data. A number of translation files are also emitted so that the numeric data can be later printed in symbolic form. A teacher, room and class usage report is also generated. This report gives the number of periods required by each teacher, room and class.

A pretty printer program generates timetable reports for all or selected teachers, classes, rooms and subjects. The pretty printer also performs room assignment where appropriate. The flow of information is shown in Figure 2.

It is beyond the scope of this paper to describe the syntax of the timetable specification language, however, it can be illustrated by the following small fragment of a University timetable specification. Each requirement is denoted by the 'Schedule' statement, which names the class, subject, teacher, room and a frequency. This timetable contains a number of requirements which are locked into specific periods. A number of other requirements are specified with preferences for certain periods. Grouped requirements are denoted by the use of the 'together with' clause. The weightings are entered with the 'Importance' statements. Interested readers are referred to [13] for a full description of the language.

---

TIMETABLE DATA 9 periods per day 5 days per week

breaks after Monday 5, Tuesday 5, Wednesday 5, Thursday 5, Friday 5

(\* the immovable items first \*)

(\* starting with maths \*)

schedule yrlengA taught maths191\_04 by taa in rm1 1 times fixed Tuesday 2

schedule yrlengA taught maths191\_04 by abd in rm1 1 times fixed Friday 2

schedule yrlengB taught maths191\_04 by abd in mathsroom 1 times fixed Monday 5

schedule yrlengB taught maths191\_04 by daa in mathsroom 1 times fixed Wednesday 4

schedule yrlengB taught maths191\_04 by gke in mathsroom 1 times fixed Friday 4

schedule yrlengA taught maths191\_04 by rao in lect1 1 times fixed Monday 4

together with yrlengB taught maths191\_04 by mlr in nil

schedule yrlengB taught eng161\_04 by daa in E1 2 times

with low preference for Monday 2 through Monday 5,

Tuesday 2 through Tuesday 5,

Wednesday 2 through Wednesday 5,

Thursday 2 through Thursday 5,

Friday 2 through Friday 5

together with yr1cscB taught eng161\_04 by nil in nil

together with yr1chmB taught eng161\_04 by nil in nil

cooling rate is .9 depth is medium

Importance of Classclashes is 5

Importance of Teacherclashes is 5

Importance of Roomclashes is 5

Importance of Blocks is 4

Importance of preference is 1

Importance of relationships is 1

END OF TIMETABLE

---

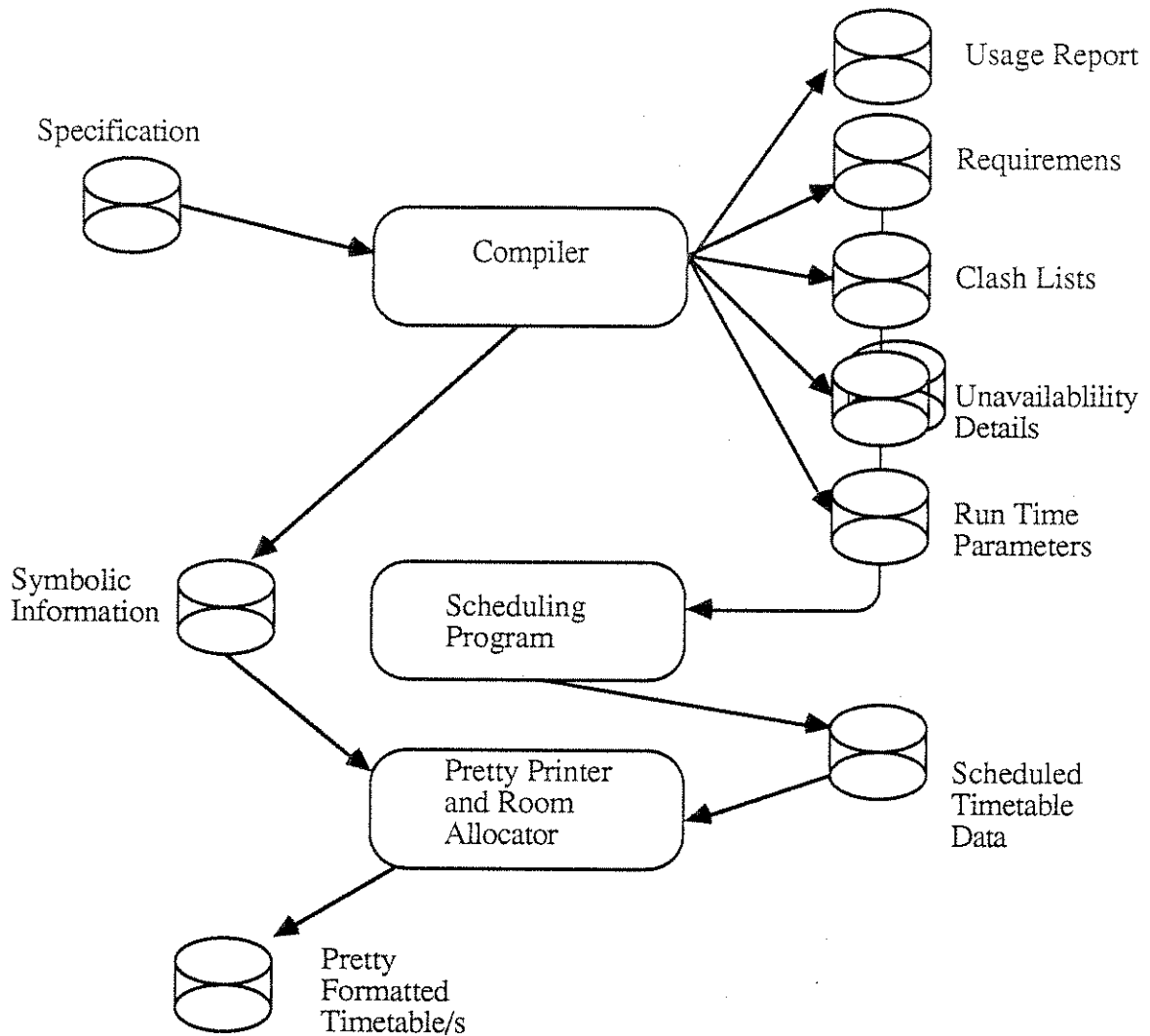


Figure 2 - Timetable Computation Tool Set

## 10. CONCLUSION

Simulated annealing is a relatively new technique for solving optimisation problems. The traditional solutions for the timetabling problem have either involved linear programming or heuristic search techniques. In this paper we have examined the use of simulated annealing to solve this problem. The results of the serial algorithm are very promising and warrant further work. It would be possible to add cost components to include the more complex scheduling constraints that arise in schools. The weighting of these components allows one component to be made more important than others.

The speed of the algorithm can be further improved by implementing a parallel program as described. These results show good speedup until there are too many competing processors. The problem is easily parallelised because it is possible to compute changes in cost with little interaction with other potential changes. This property is not evident in all simulated annealing problems. The program can be implemented on a conventional shared memory multiprocessor (MIMD). It would be difficult to execute the program on a message passing machine because it is necessary to have fast access to the timetable structure. At the time this paper was written, the algorithm was implemented as a parallel program for a conventional MIMD. This program uses no more than standard System V Unix Forks and shared memory. The parallel program can also be compiled as a serial version for a uniprocessor, in which case the locking and unlocking code is omitted.

One outstanding problem for the parallel algorithm is how many instances of the permute code should be run concurrently. The optimal value can only be found by increasing the number of

processes until the speedup starts to decrease. A more useful solution would be to have the algorithm monitor the speedup as it executes. One possibility is to measure the amount of congestion being caused by locked elements, teachers, rooms and classes. If the amount of interaction rises too high then some of the processes should be shut down until the measure decreases. However, this technique requires further investigation.

Another area which warrants further investigation is the removal of some of the synchronisation locks from the parallel code. These locks decrease the speedup experienced over the serial code. It is possible to omit many of the locks and allow the code to have incorrect views of the data from time to time. It is not clear what then effect will be on the convergence of the algorithm, and further experimental work is required.

## ACKNOWLEDGEMENTS

The Parallel Systems Architecture Project at RMIT is being supported by the Commonwealth Scientific and Industrial Scientific Organisation (CSIRO) under an Information Technology joint research project.

The programs used for producing the experimental results were written by Tony Starr and Mark Gazzola. The initial interest in simulated annealing came from the CSIRO Division of Buildings Research, in particular Bertil Marksjo and Ron Sharp. Thanks must also go to the RMIT Department of Computer Science for providing access to their Encore multiprocessor. The referees helped improve the clarity of this paper.

## REFERENCES

- [1] O.B. de Gans, "A computer timetabling system for secondary schools in the Netherlands", *European Journal of Operations Research*, Vol 7, 1981, pp 175-182.
- [2] E.A. Akkoyunlu, "A linear algorithm for computing the optimum university timetable", *Computer J*, 16(4), 1973, pp 347-350.
- [3] J. Csima and C.C. Gottleib, "Tests on a computer method for construction of school timetables", *Comm. ACM*, 7(3), 1961, pp 160-163.
- [4] J.S. Folkers, "A computer system of time-table conditions", Ph.D. Thesis, Delft University of Technology, 1967.
- [5] K. Fujino, "A preparation for the timetable using random number", *Information processing in Japan* 5, 1965, pp 8-15.
- [6] C.C. Gottleib, "The construction of class-teacher time tables", in C.M.Popplewell, Ed., *Proc IFIP Congress Munchen 1962* (North-Holland, Amsterdam, 1963).
- [7] B. Greko, "School scheduling through capacitated network flow analysis", *Swed. Off. Org. Man.*, Stockholm, 1965.
- [8] N.L.Lawrie, "An integer programming model of a school time-tabling problem", *Comput. J.* 12 (1969), pp 307-316.
- [9] N. Macon and E.E. Walker, "A Monte Carlo algorithm for assigning students to classes", *Comm. ACM* 9(6), 1966, pp 339-340.
- [10] P.J.M. van Laarhoven and E.H.L. Aarts, "Simulated Annealing: Theory and Applications", D. Reidel Publishing Company, 1987, Kluwer Academic Publishers Group.
- [11] S.R. White, "Concepts of Scale in Simulated Annealing", *Proc. IEEE Int Conference on Computer Design*, Port Chester, November 1984, pp 646-651.



- [12] N. Chng, "A solution to the timetabling problem", Honours Report 1983, Department of Computer Science, Monash University, Clayton, Australia.
- [13] D. Abramson. "A Technique for Specifying Timetable Requirements", Department of Communications Engineering report TR-112-71R, Royal Melbourne Institute of Technology, Melbourne, Australia, 1987.

