Analytically Tuned Parameters of Simulated Annealing for the Timetabling Problem

JUAN FRAUSTO-SOLIS, FEDERICO ALONSO-PECINA

Instituto Tecnológico y de Estudios Superiores de Monterrey, Cuernavaca, Morelos, México

{juan.frausto, A01125296}@itesm.mx

CINHTIA GONZALEZ-SEGURA

Universidad Autónoma de Yucatán

gsegura@uady.mx

Abstract: - University Timetabling problem (UTT) has a computational complexity that grows exponentially as the size of the problem augments; therefore random algorithms become the best alternative to solve it, and among them, Simulated Annealing (SA) is one of the most efficient. However, SA obtains very good solutions, only if its parameters are well tuned. SA requires an initial solution for solving UTT. Besides, analytical tuning strategies for SA in UTT have not been explored. In this paper a SA Markov tuning strategy and a heuristic to generate feasible solutions are proposed. This strategy improves SA performance for UTT as is shown with experimental instances taken from PATAT benchmark.
.

Key-Words: - Timetabling, Simulated Annealing, Optimization

1 Introduction

This paper deals with an Educational Timetabling problem proposed in PATAT [1], and named University Timetabling (UTT) which has a computational complexity that grows exponentially as the size of the problem is increased. Commonly UTT is realized as the scheduling of courses and teachers, within a given number of rooms (classrooms) and time periods; however as is described in section 2 other events different than courses (v.gr. conferences) can be considered.

A common approach to solve UTT is first to find a feasible initial solution using some procedure specially developed for that purpose; then a metaheuristic [2]-[7] is commonly applied to improve this initial solution. Simulated Annealing (SA) is among these metaheuristics and one of the most used algorithms for UTT [6], [8], [9].

SA obtains very good solutions (close to the optimal or even the global optimum) [5] whether it is well tuned and an adequate neighborhood is used.

In this paper a new SA algorithm for UTT using a Markov tuning strategy is proposed; this new algorithm is named "Tuned Simulated Annealing" (TSA). In addition, two neighborhoods are tested on TSA. Besides, experimentation using instances taken from PATAT benchmark [1] shows TSA is more efficient for UTT than the classical SA.

The paper is organized as follows. Section 2 includes the PATAT problem description and its mathematical model. In section three, the instances to test the algorithms are presented. Section 4 presents TSA variations and experimentally tuned. Section 5 describes the analytical tuning parameters strategy. Finally, Sections 6 and 7 include the results and the conclusions respectively.

2 Problem Description

For UTT, as for any other NP problem, it is not efficient to apply deterministic methods due to their exponential complexity [8]; therefore it is useful to develop random methods to find the best feasible solution. Basically, UTT consists of:

i) a set *E* of events to be scheduled in 45 periods of time (5 days of 9 intervals every one),
ii) a set R of rooms that hold the events,
iii) a set U of students that attend the events, and
iv) a set F of features that should satisfy every room because of the exigencies of events to be held there. Each student

attends specific events and each room has a capacity measured in number of students.

A feasible timetable is one in which all events have been assigned a timeslot and a room, and all of the following hard constraints (hc) must be completely fulfilled:

hc_1: No student attends more than one event at the same time.

hc_2: The room chosen for an event is big enough to house all the attending students and it satisfies all the technical features required by the event.

hc_3: Only one event is hold in each room at any specific timeslot.

In addition, there are other constraints, named soft constraints (sc), which are desirable not be violated. The soft constraints of PATAT are the following:

sc_1: Any student should not have a class in the last slot of the day.

sc_2: Any student should not have more than two classes consecutively.

sc_3: Any student should not have a single class on any day.

The PATAT's criteria to determine the winner of the contest is based on the next formula [1]:

$$F_i = \left(x_i - b_i\right) \Big/ \left(w_i - b_i\right) \quad (1)$$

where i is the number of the instance ($1 \leq i \leq 20$), x is the number of soft constraints violated for the contestant; b and w are, respectively, the number of soft constraints violated by the best and the worst participant on this instance during the same contest.

The next variables are defined in PATAT for UTT:

- n Events: $E = \{e_1, e_2, \ldots, e_n\}$
- m Students: $U = \{u_1, u_2, \ldots, u_m\}$
- 45 Periods: $P = \{p1, p2, \ldots, p45\}$
- r Rooms: $A = \{a1, a2, \ldots, ar\}$
- r Room sizes: $C = \{c1, c2, \ldots, cr\}$
- t Features: $F = \{f1, f2, \ldots, ft\}$

Besides three binary matrixes are defined as follows:

- Matrix student/event: D_{nxm}

$$D_{nxm} = \begin{bmatrix} d_{11} & d_{12} & \ldots & d_{1n} \\ d_{21} & d_{22} & \ldots & d_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ d_{m1} & d_{m2} & \ldots & d_{mn} \end{bmatrix}$$

where $d_{il} = 1$ if the student l attends the event i, and 0 otherwise.

- Matrix room/feature: S_{txr}

$$S_{txr} = \begin{bmatrix} s_{11} & s_{12} & \ldots & s_{1t} \\ s_{21} & s_{22} & \ldots & s_{2t} \\ \ldots & \ldots & \ldots & \ldots \\ s_{r1} & s_{r2} & \ldots & s_{rt} \end{bmatrix}$$

where $s_{jf} = 1$ if the room j satisfies the feature f, and 0 otherwise.

- Matrix event/feature: Q_{txn}

$$Q_{txn} = \begin{bmatrix} q_{11} & q_{12} & \ldots & q_{1t} \\ q_{21} & q_{22} & \ldots & q_{2t} \\ \ldots & \ldots & \ldots & \ldots \\ q_{n1} & q_{n2} & \ldots & q_{nt} \end{bmatrix}$$

where $q_{if} = 1$ if the event i requires the feature f, and 0 otherwise.

Let be $x_{ijk} = 1$ if the event i, is assigned to the room j in the period k, and 0 otherwise.

Therefore, the problem can be formulated as follows:

$$Min\ z = \sum_{v=1}^{TotSoftConstraints} sc_v$$

$$subject\ to$$

$$Every\ \ hc\ \ is\ Fulfilled$$

Next the explanation and mathematical model of every constraint is presented:

Hard Constraints (hc):

hc_1 **(No student attends more than one event at the same time)**: This constraint establishes that in any period k, at most one event $i \in Q_e$ can be programmed. Q_e *is a* set defined for every event. Q_e contains its event e plus every event that have conflicts with e. Therefore this constraint is written as:

$$\sum_{i \in Q_e} x_{ijk} \leq 1 \quad j = 1,\ldots,r \quad k = 1,\ldots,45 \quad e = 1,\ldots,n \quad (2)$$

where i, j and k represent the number of events, rooms and periods respectively.

hc_2 (**The room is big enough for all the attending students and satisfies all the features required by the event):** This constraint is described by these two other constraints:

- A) The room is big enough for all the attending students, and

- B) The room satisfies all the features required by the event.

The constraint A can be also stated as follows: For every room *j* where the event *i* is programmed, the capacity c_j of the room *j* should be higher or equal to the number of students that attend the event *i*. That is equivalent to the next relation among the number b_i of participants of the event *i*:

$$b_i = \sum_{l=1}^{m} d_{li} \quad i = 1,...,n \tag{3}$$

where the student *l* attends the event *i*, d_{li} worths 1 if the student *l* attends the event *i*.

The constraint A can be written as follows:

$$\forall x_{ijk} = 1, \quad b_i \le c_j, \qquad i = 1,...,n \quad j = 1,...,r \quad k = 1,...,45 \tag{4}$$

The constraint B can be stated as follows: For each *k* period, the room *j* must satisfy all the features required by every event *i* programmed in this room. This constraint can be established as follows:

$$x_{ijk} = 1 \Rightarrow \forall f \; q_{if} \le s_{jf}, \qquad i = 1,...n \quad j = 1,...,r \quad k = 1,...,45 \tag{5}$$

where q_{if} represents the feature *f* associated to the event *i*, and s_{jf} represents the feature *f* satisfied by the room *j*.

hc_3: **(Only one event is hold in each room at any timeslot).** This constraint is established for every period *k*. For any period k and any room *j* at most one event must be held. This constraint can be written as follows:

$$\sum_{j=1}^{r} x_{ijk} \le 1 \quad i = 1,...,n \quad k = 1,...,45 \tag{6}$$

Besides, there is another very important hard constraint implicit in the problem's definition: "All the events must be programmed in some periods". Therefore, for all the 45 periods, every event *i* must be programmed exactly once. Therefore this constraint can be written as follows:

$$\sum_{k=1}^{45} \sum_{j=1}^{r} x_{ijk} = 1 \quad i = 1,...,n \tag{7}$$

Soft constraints

The soft constraints are the following:

1. **Any student should not have a class in the last slot of the day**

Let be a set *V* that contains the last periods of the days: *V = {k | k mod NUMPER = 0}*. Where *k* = 1, 2, ... , 45 and *NUMPER* = Number of periods for day (9, in this case). This constraint can be realized in this way: "some event *i* should not be programmed in any room *j*, in any period $k \in V$". Therefore, this constraint can be written as:

$$\forall k \in V \quad \sum_{i=1}^{n} \sum_{j=1}^{r} x_{ijk} = 0 \tag{8}$$

2. **Any student should not have more than two classes consecutively**

That means that any student *l* should not have 3 or more events programmed in a row. Let be S_l the set of events of the student *l* so this constraint is written as:

$$\forall \; v \in V \qquad \sum_{k=v-NUMPER+1}^{v-2} \sum_{i \in S_l} \sum_{j=1}^{r} x_{ij(k+1)} x_{ij(k+2)} x_{ij(k+3)} = 0$$

$$l = 1,2,\ldots m \tag{9}$$

3. **Any student should not have a single class on any day.**

This constraint establishes that all the students should have programmed zero or more that one event per day; that means:

$$\forall v \in V \qquad \sum_{k=v-NUMPER+1}^{v-2} \sum_{i \in S_l} \sum_{j=1}^{r} x_{ijk} \ne 1 \qquad l = 1,...,m \tag{10}$$

3 Problem's Instances

In [1], a generator to produce instances from different seeds is used. All the instances generated have at least one perfect solution, which means, zero violated constraints (hard and soft). In table 1, the features of PATAT's instances are presented, and table 2 contains the criteria to classify them.

In PATAT, small cases are called S1, S2, S3, S4 and S5, while medium instances are called M1, M2, M3, M4 and M5, and finally, large instances are called L1 and L2. Besides, in PATAT some UTT instances are commonly named Competitionx or Cx, where x is the number of the UTT instance (e.g., Competition01 is C01, and so on).

4 Algorithm Description

Simulated Annealing (SA) is one of the metaheuristics used with more success to solve UTT. SA allows some "wrongs" movements in order to escape of local optimum with the goal of reach the global optimum whether its parameters are well tuned.

Table 1 PATAT´s Benchmark

Instance	Events	Rooms	Features	Students
C01	400	10	10	200
C02	400	10	10	200
C03	400	10	10	200
C04	400	10	5	300
C05	350	10	10	300
C06	350	10	5	300
C07	350	10	5	350
C08	400	10	5	250
C09	440	11	6	220
C10	400	10	5	200
C11	400	10	6	220
C12	400	10	5	200
C13	400	10	6	250
C14	350	10	5	350
C15	350	10	10	300
C16	440	11	6	220
C17	350	10	10	300
C18	400	10	10	200
C19	400	10	5	300
C20	350	10	5	300

Table 2 Socha's classification

Instance	Events	Rooms	Features	Students
small	100	5	5	80
medium	400	10	5	200
Large	400	10	10	400

The most common cooling scheme of SA and used in this paper is a geometric scheme: $T(k+1)=\alpha T(k)$, where k is the temperature number and $0<\alpha<1$. The algorithm's parameters are: the initial temperature T_0, the final temperature T_F , the parameter alpha α and the length of Markov chain L. Besides, in SA a feasible initial solution S_0 is required; to find this initial solution for the PATAT's benchmark, two different methods are commonly used:

- The first method uses a heuristic that provides a feasible solution and then the classical SA improves it. The heuristic that finds this initial solution uses the concept of "more constrained event" [8].
- The second method uses SA itself to find the initial feasible solution. In the past this strategy was not successful to find feasible initial solutions but [5], now it is more efficient. This method disables the infeasible movements in such a way that, they do not introduce any new hard violation.

The algorithm is the following: Firstly, all the events are initialized to the first hour in the first room. Then, a random event is chosen. Its feasible neighborhood is calculated and a random feasible neighbor is selected. In most of the cases, the procedure reaches a feasible solution and when it is achieved, the algorithm continues improving its objective function until the system is frozen. The next implementations of SA were developed in this work:

Pseudocode of the implementation SA01

```
Begin
x = initial_solution();
BestCost = costIni = f(x);
T = 470 * num_students + num_events;
L_MAX = 10000;
END_TEMP = 0.01;
While (T > END_TEMP)
   Iter=0;
   While (Iter < L_MAX)
      Do
         timeslot_1 = rand() MOD 45;
         timeslot_2 = rand() MOD 45;
         room_1 = rand() MOD num_rooms;
         room_2 = rand() MOD num_rooms;
         if the interchange of the event
         placed in timeslot_1 and room_1
         with the event placed in
         timeslot_2 and room_2 is feasible
         then
             ban = true
         else
             ban = false;
         endif
      While ( ban == false)
      costNew = f(x_new);
      costDif = costNew – costIni;
      r = rand()
      if (costNew <= 0) then
         costIni = costNew;
         x = x_new;
      else
         r = rand()
         if (r < exp(-costDif/T)) then
            costIni = costNew;
            x= x_new;
         End_if
      End_if
      if (BestCost > costoIni) then
         x* = xl;
         BestCost = costoIni;
      End_if
      iter = iter + 1
   End_While
   T = T * ALPHA
End_While
End
```

4.1 First Neighborhood

A neighborhood called first neighborhood or SA01) was established with the next parameters:

- Markov chain length: 10000 (as in[8]).
- Cooling scheme: Geometric defined as $T_n = \alpha T_{n-1}$. Experimentation was done with different alpha values: 0.70, 0.75, 0.85, 0.90 and 0.95
- Initial temperature ci=470m + n, where m=number of students; n=number of events.

This SA implementation starts from a feasible solution obtained for the first strategy previously described in [8].

This first neighborhood is used to generate new solutions with a procedure as follows.

- First, two random periods are selected and after that,
- Two random classrooms are chosen. If an interchange of events is feasible, this interchange is accepted. Otherwise, two news random periods and two new random rooms are selected, and this process goes on.

4.2 Second Neighborhood

A second neighborhood (SA02) was also defined; is similar to SA01 except by the form of the procedure used to generate new solutions:

- Firstly, a random event is chosen;
- Secondly, if this event is feasible several interchanges are calculated (over the period and/or over the room or even over other event) taking care that any hard constraint is not violated.
- Finally, a random change is chosen.

This new way to calculate the neighborhood allows to obtains better solutions that the previous one.

4.3 SA without feasible initial solution

SA without feasible initial solution was label SA03. Initially, any feasible solution is given to the algorithm, all the events are set in the timeslot 1 and the room is set to 1. Using this solution, the algorithm starts working until it reaches a feasible solution. The percentage of not feasible solutions obtained with this approach is less to 2%. The neighborhood using by this implementation is the same as the Second Neighborhood.

Pseudocode of the implementation SA02

```
Begin
x = initial_solution();
BestCost = costIni = f(x);
T = 470 * num_students + num_events;
L_MAX = 10000;
END_TEMP = 0.01;
While (T > END_TEMP)
   Iter=0;
   While (Iter < L_MAX)
      ban = false;
      Do
         e1 = rand() MOD num_events;
         num = neighborhood(e1);
         if num != 0 then
             move = rand() MOD num;
             ban = true;
             do the the random move of
             the event
         endif
      While ( ban == false)
      costNew = f(x_new);
      costDif = costNew – costIni;
      r = rand()
      if (costNew <= 0) then
         costIni = costNew;
         x = x_new;
      else
         r = rand()
         if (r < exp(-costDif/T)) then
            costIni = costNew;
            x = x_new;
         End_if
      End_if
      if (BestCost > costoIni) then
         x* = xl;
         BestCost = costoIni;
      End_if
      iter = iter + 1
   End_While
   T = T * ALPHA
End_While
End
```

5 Analytically tuned parameters of Simulated Annealing

The scheme of tuned parameters published in [10] was used. This scheme was used successfully in other problems [11] [12]. In this scheme, the initial temperature T_0 is obtained as a function of the maximum possible deterioration of the objective function that can be accepted in a current solution. In other hand, the final temperature T_f is obtained as a function of the minimum possible deterioration of the objective function.

The maximum possible deterioration, with the proposed neighborhood is: K*(maximum number of students per event). Where is easy to note that in this case K is equals to eight

T_0 and T_f are calculated with the next formulas:

$$T_0 = \frac{-\Delta Z_{V\max}}{\ln\left(P^A(\Delta Z_{V\max})\right)} \quad (11)$$

$$T_f = \frac{-\Delta Z_{V\min}}{\ln\left(P^A(\Delta Z_{V\min})\right)} \quad (12)$$

Pseudocode of the implementation SA04

```
Begin
x = initial_solution();
BestCost = costIni = f(x)
T_analytical = DZmax / log
(Pacceptation)
β = exp((log(Lmax)-log(L1))/n);
END_TEMP = 0.01;
L=L1;
While (T > END_TEMP)
   Iter=0;
   While (Iter < L)
      Do
         timeslot_1 = rand() MOD 45;
         timeslot_2 = rand() MOD 45;
         room_1 = rand() MOD num_rooms;
         room_2 = rand() MOD num_rooms;
         if the interchange of the event
         placed in timeslot_1 and room_1
         with the event placed in
         timeslot_2 and room_2 is
         feasible then
             ban = true
         else
             ban = false;
         endif
      While ( ban == false)
      costNew = f(x_new);
      costDif = costNew – costIni;
      r = rand()
      if (costNew <= 0) then
         costIni = costNew;
         x = x_new;
      else
         r = rand()
         if (r < exp(-costDif/T)) then
            costIni = costNew;
            x= x_new;
         End_if
      End_if
      if (BestCost > costoIni) then
         x* = xl;
         BestCost = costoIni;
      End_if
      iter = iter + 1
      End_if
   End_While
   T = T * ALPHA
   L = β * L
End_While
End
```

Pseudocode of the implementation SA05

```
Begin
x = initial_solution();
BestCost = costIni = f(x)
T_analytical = DZmax / log (Pacceptation)
β = exp((log(Lmax)-log(L1))/n);
END_TEMP = 0.01;
L=L1;
While (T > END_TEMP)
   Iter=0;
   While (Iter < L)
      ban = false;
      Do
         e1 = rand() MOD num_events;
         num = neighborhood(e1);
         if num != 0 then
             move = rand() MOD num;
             ban = true;
             do the the random move of
             the event
         endif
      While ( ban == false)
      costNew = f(x_new);
      costDif = costNew – costIni;
      r = rand()
      if (costNew <= 0) then
         costIni = costNew;
         x = x_new;
      else
         r = rand()
         if (r < exp(-costDif/T)) then
            costIni = costNew;
            x= x_new;
         End_if
      End_if
      if (BestCost > costoIni) then
         x* = xl;
         BestCost = costoIni;
      End_if
      iter = iter + 1
      End_if
   End_While
   T = T * ALPHA
   L = β * L
End_While
End
```

The Markov chains length L of the metropolis loop (the inner one) are also tuned dynamically. In [10], is presented how to determine the length L_i for any iteration *i*. In this method, L_i is determined establishing the relationship between the cooling function and the Markov chain length. In the first Metropolis cycle the *L-parameter is* 1 and it is increased according to a β-parameter, until, in the last Metropolis cycle of Metropolis, the maximum L value or L_{max},:is reached; Lmax follows the Markov model (13)

$$L_{max} = \beta^n L_1 \quad (13)$$

Where

$$n = \frac{\ln L_{\max} - \ln L_1}{\ln \alpha} \quad (14)$$

$$\beta = \exp \frac{\ln L_{\max} - \ln L_1}{n} \quad (15)$$

With this analytical method, T_f will be 0; however the stochastic equilibrium is verified here since 0.01. L_{max} takes the same value for all implementations (10000 iterations). Implementations with these tuned parameters are next described: The implementation of the analytical tuned parameters of SA first neighborhood (SA01) was labeled SA04.
The implementation of the analytical tuned parameters of SA second neighborhood (SA02) was called SA05

Usually, using the analytical tuned approach is possible to obtain a SA algorithm which has a similar quality of the experimental tuned version of the same algorithm, but the former may save until fifty percent of computational time [11], [12]. The next paragraphs presents an efficiency analysis of the algorithm analytically tuned is presented:

Theorem 1:
The number of iterations of the cycle of Metropolis of Tuned Simulated Annealing (ITSA) is smaller than the number of iterations of the Simulated Annealing (ISA), when the number of cycles of Metropolis is greater or equal to two:

$$ITSA < ISA,\ n \geq 2 \quad (16)$$

using:

$$\beta = \exp \frac{\ln L_{\max} - \ln L_1}{n} \quad (17)$$

Where n is the number of cycles of Metropolis, L_{max} is the number of iterations of the last cycle of Metropolis, and L_1 is the number of iterations of the first cycle of Metropolis.

$$\beta = \exp\left(\frac{\ln \frac{L_{MAX}}{L_1}}{n} \right) = \exp\left(\ln \frac{L_{MAX}}{L_1} \right)^{\frac{1}{n}} = \left(\frac{L_{MAX}}{L_1} \right)^{\frac{1}{n}} \quad (18)$$

When n = 1: $L_1 < L_{MAX}$, therefore *ITSA* < *ISA* where n = 1
When n = 2

$$L_1 \left(\frac{L_{MAX}}{L_1} \right)^{\frac{1}{n}} < L_{MAX} \quad (19)$$

Therefore

$$L_1 + L_1 \left(\frac{L_{MAX}}{L_1} \right)^{\frac{1}{n}} < 2L_{MAX} \quad (20)$$

and *ITSA* < *ISA* where n = 2. In general, it is trivial to demonstrate that:

$$L_1 \left(\frac{L_{MAX}}{L_1} \right)^{\frac{i}{n}} < L_{MAX} \quad (21)$$

where $0 \leq i < n$, then:

$$\sum_{i=0}^{n-1} L_1 \left(\frac{L_{MAX}}{L_1} \right)^{\frac{i}{n}} < (n-1) L_{MAX} \quad (22)$$

And, adding L_{MAX} to both members of the inequality (in the iteration number n), it is obtained:

$$\sum_{i=0}^{n} L_1 \left(\frac{L_{MAX}}{L_1} \right)^{\frac{i}{n}} < n L_{MAX} \quad (23)$$

Thus *ITSA* < *ISA*

Although an interesting saving of time was obtained with the analytical tuned parameters of SA, the quality of its solutions is lightly smaller to those obtained without formulae (11) to (15). The explanation is the following. The experimental method has obtained a very big T_0 value, which was bigger than the T_0 value obtained with the analytical method (formulas 11-15); in fact the latter was relatively too small. The later result was obtained because both the acceptance probability and the exploratory capacity of the experimental implementation were very high. To solve this problem, some actions can be taken:
a) The initial temperature T_0 is set equal to that obtained with the experimental method, and its Markov chain length is set equal to 1 (L_1=1), as is established by the analytical method.
b) After each cycle of Metropolis, the temperature is decreased according to the geometrical cooling scheme, and the Markov chain length is increased by one, until is reach the T_0 value obtained with the analytical method.
c) Starting from the T_0 value obtained by the analytical method, the length Markov chain is increased according to the β parameter, until it reaches L_{max} and then, it stays constant until arriving to T_f.
This implementation is called, SATUNED and the pseudocode is the following one:

Pseudocode of the implementation SATUNED

```
Begin
x = initial_solution();
BestCost = costIni = f(x)
T = 470 * num_students + num_events;
T_analytical = DZmax / log
(Pacceptation)
β = exp((log(L_max)-log(L_1))/n);
END_TEMP = 0.01;
L=L_1;
While (T > END_TEMP)
   Iter=0;
   While (Iter < L)
            ban = false;
      Do
         e1 = rand() MOD num_events;
         num = neighborhood(e1);
         if num != 0
             then
             move = rand() MOD num;
             ban = true;
             do the the random move of
             the event
         endif
      While ( ban == false)
      costNew = f(x_new);
      costDif = costNew – costIni;
      r = rand()
      if (costNew <= 0)
         then
         costIni = costNew;
         x = x_new;
         else
         r = rand()
         if (r < exp(-costDif/T))
             then
             costIni = costNew;
             x= x_new;
         End_if
       End_if
      if (BestCost > costoIni)
         then
         x* = xl;
         BestCost = costoIni;
      End_if
      iter = iter + 1
      End_if
   End_While
   T = T * ALPHA
   if(T<T_analytical)
      then
      L = L + 1
   else
      if(L<Lmax)
         then
         L = β * L
      Endif
   endif
End_While
End
```

6 Results

In table 3 to 4, and figures 1 to 4 are presented the results of quality and execution time of the different implementations of SA proposed in this paper.

Graphical results for alpha 0.90 and 0.95 used in the geometrical cooling scheme are shown. In most of the cases, the best results were obtained by a SA using the second neighborhood and SA without feasible initial solution or SATUNED (SA02, SA03 and SA06, respectively).

Figure 4 shows the execution time of the implementations with alpha equals to 0.95; as can be noticed, the faster implementation is SA05, but, its quality is not the best one.

Regarding the execution time, the best result was obtained with implementations SA02, SA03 and SA06. For simplicity, only some alpha values in this table and figures are presented.

The results are shown in two categories: quality and time. The quality solution is measured considering the number of soft constraints violated.

The execution time unit used is seconds and every instance was executed ten times for every alpha value:

Table 3 Quality results with alpha = 0.70

Instance	SA01	SA02	SA03	SA05	SA06
s1	1	3.7	2.1	3.5	2.4
s2	22	5.7	2.8	6.4	4.1
s3	1	6.2	3.2	6.3	4.1
s4	1	5.4	2.6	6.7	4.6
s5	80	2.3	0.5	0.8	1.1
M1	150	112.3	114.3	112.8	114.8
M2	170	111.5	113.1	109.2	113.4
M3	199	154.5	153.8	155.5	151.8
M4	122	102.4	105.9	104.8	102.5
M5	123	72.2	74.2	86.1	85.7
H1	*	*	550.5	*	*
H2	*	*	482.4	*	*

* It was not possible to obtain enough feasible solutions to make an average.

Table 4 Quality results with alpha = 0.80

Instance	SA01	SA02	SA03	SA04	SA05	SA06
s1		0.9	2.6		3.6	1.5
s2		2.4	2		3.2	3.1
s3		2.4	2.2		3.8	2
s4		2.6	3.4		4.5	4.1
s5		0.9	1.1		2	0.3
M1		99	104.1		99.1	104.7
M2		109.8	104.3		99.6	95.1
M3		137.1	141.4		137.7	140
M4		95.7	93.9		91.2	90.1
M5		61	71.5		77.7	68.9
H1		*	478.1		*	*
H2		*	434.7		*	*

Table 5 Quality results with alpha = 0.90

Instance	SA01	SA02	SA03	SA04	SA05	SA06
s1	1	1.1	1.4		1.4	0.8
s2	10	1.5	1.4		1.7	1.5
s3	1	2.1	1.1		3.6	2.4
s4	1	2.8	1.6		3.2	2.7
s5	82	0.3	0.2		1.5	0.5
M1	126	93.9	85.5		92.6	87.1
M2	161	80	84		81.1	79.7
M3	149	118.1	117.9		120.2	118.4
M4	105	74.4	74.8		73	75
M5	72	47.5	48.1		52.3	49.2
H1	753	*	425.4		*	*
H2	870	*	380.8		*	*

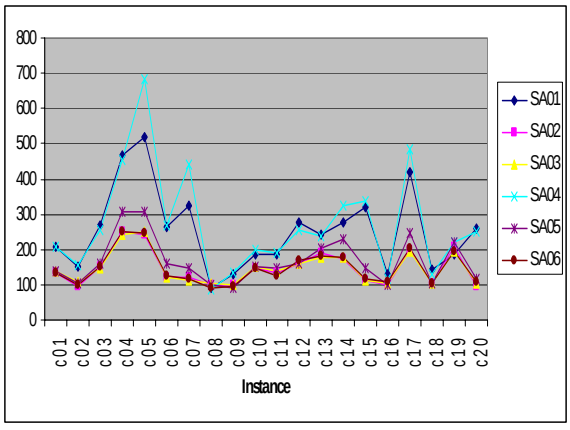

Fig. 1 Quality results using alpha = 0.85, The axe *y* is the number of soft constraints violate and the axe *x* is the instance.

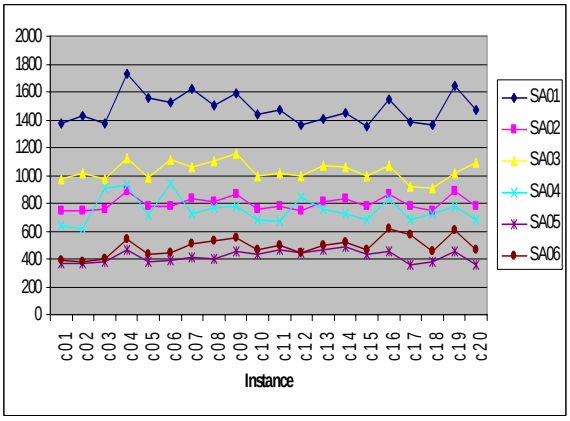

Fig. 2 Time results using alpha = 0.85, the axe *y* are the seconds and the axe *x* the instance

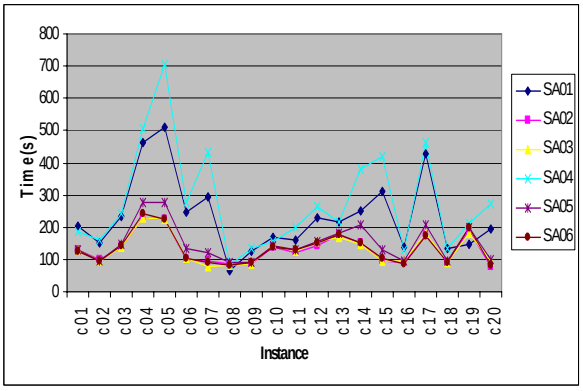

Fig. 3 Quality results using alpha = 0.90, The axe *y* is the number of soft constraints violate and the axe *x* is the instance.

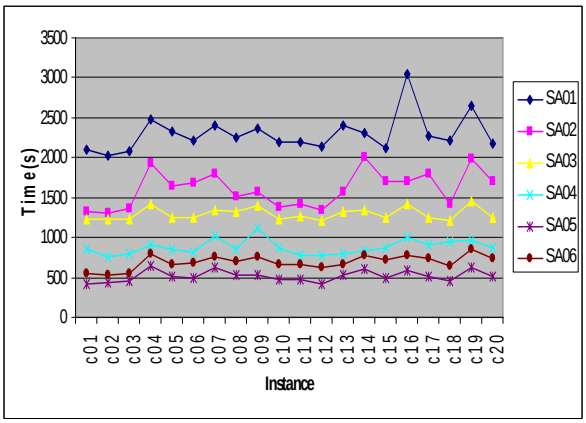

Fig. 4 Time results using alpha = 0.90, the axe *y* are the seconds and the axe *x* the instance

* It was not possible to obtain enough feasible solutions to make an average.

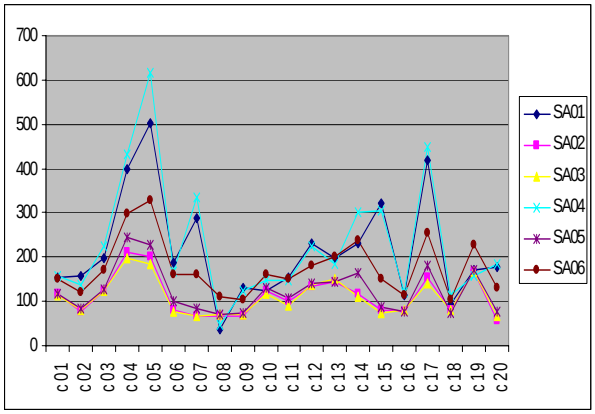

Fig. 5 Quality results using alpha = 0.95, The axe *y* is the number of soft constraints violate and the axe *x* is the instance.

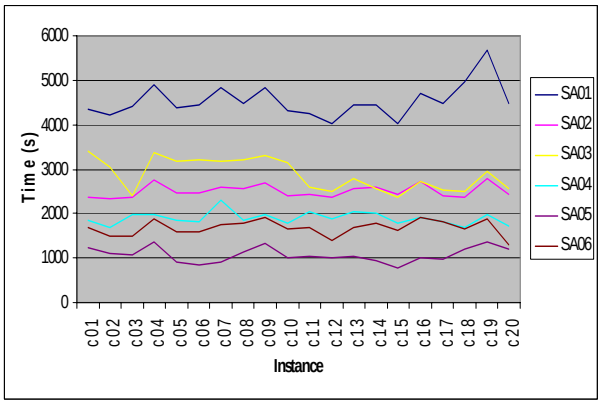

Fig. 6 Time results using alpha = 0.95, the axe *y* are the seconds ant the axe *x* the instance

7 Conclusions

In this paper, several implementations of SA for UTT are presented. These implementations are able to find feasible solutions for hard instances. It represents an advance in relation with previous results [5]. The best results were obtained with SA02, SA03 and SA06; the fastest was SA06. SATUNED implementation (SA06) saves around 32% of the execution time wasted by SA02 or saves around 40% of the time used by SA03. Besides SA06 has a similar quality than other implementations. Therefore, SA with the analytical tuned method presented in the paper had a good performance and is relatively very simple to be implemented.

References:

[1] International Timetabling Competition, URL: http://www.idsia.ch/Files/ttcomp2002/ Consultant date: November 28 of 2007

[2] Cerny, V. Minimization of Continuous Functions by Simulated Annealing. *Research Institute for Theoretical Physics, University of Helsinki*, preprint HU-TFT-84-51, 1984.

[3] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. Optimization by Simulated Annealing, *Science*, Vol 220, No. 4598, pp 671-680, (1983).

[4] Kostuch, P.A. The University Course Timetabling Problem with 3-Phase approach. *Third Int Conf on Practice and Theory of Automated Timetabling, USA*,LNCS 3616 Springer 2005, ISBN 3-540-30705-2. pp 251-266

[5] Rossi-Doria, O., Sampels, M., Biratrari, M., Chiarandini, M., Dorigo, M., Gambardella, L. M., Knowles, J., Manfrin, M., Mastrolilli, L., Paetcher, B., Paquete L., Stützle, T. A comparison of the performance of different metaheuristic on the timetabling problem. *Napier University, Université Libre de Bruxelles, Technische Universitaet Darmstadt*. (2002).

[6] Luca Di Gaspero, Andrea Schaerf. Timetabling Competition TTComp 2002: Solver Description, *Conf on Practice and Theory of Automated Timetabling, Pittsburgh,PA,USA*,Aug 2004.

[7] Socha, K.; Knowles, J.; Sampels, M. "A MAX-MIN Ant System for the University Timetabling Problem".*In Proceedings of the 3rd International Workshop on Ant Algorithms*, ANTS 2002, LNCS, Vol. 2463, Springer, (2002), pp. 1-13.

[8] Bykov, Y., The Description of the Algorithm for International Timetabling Competition. *Timetabling Competition of PATAT*, University of Nottingham, School of Computer Science & IT, Wollaton Road, August, 2004.

[9] Halvard Arntzen, Arne Lokketangen. A local search heuristic for a university timetabling problem, *PATAT time tabling Com*. Aug, 2004.

[10] Sanvicente-Sanchez, H., Frausto-Solis, J., A Method to Establish the Cooling Scheme in Simulated Annealing Like Algorithms. Assis, Italy. *ICCSA'2004. LNCS Vol. 3095*. 755-763.

[11] Sanvicente-Sanchez, H.; Frausto-Solís, J., Imperial-Valenzuela, F., Solving SAT Problems with TA Algorithms Using Constant and Dynamic Markov Chains Length, *Algorithmic Applications in Management, Springer* LNCS, June (2005)

[12] Sanvicente-Sánchez, H., A Methodology to parallelize the Temperature Cycle of Simulated Annealing Like Algorithms, (in Spanish), PhD Thesis in Computer Science, *ITESM Campus Cuernavaca*, Octuber (2003).