Combining Constraint Programming and Simulated Annealing on University Exam Timetabling

Tuan-Anh Duong, Kim-Hoa Lam

Abstract—**In this paper, we present a solution method for examination timetabling, consisting of two phases: a constraint programming phase to provide an initial solution and a simulated annealing phase with Kempe chain neighborhood. The second phase is equipped with some refined mechanisms that help to determine crucial cooling schedule parameters. We perform preliminary experiments of the algorithm on real data sets.**

Index Terms - **automated timetabling, constraint programming, simulated annealing, cooling schedule.**

I. Introduction

THE difficulty of developing appropriate examination timetables for higher education institutions is increasing. Institutions are enrolling more students into a wider variety of courses in many different fields. For example, at the Ho Chi Minh City University of Technology, approximately 10000 students have to fitted into about 320 exams over two and a haft week period. Consequently examination timetabling is a difficult combinatorial optimization problem and in fact a NP-complete problem.

The examination timetabling problem regards the scheduling for the exams of a set of university courses, avoiding overlaps of exams of courses having common students, and spreading the exams for the students as much as possible.

This scheduling problem has been studied extensively and many approaches developed to solve it. Surveys of different methods ([7]) for exam timetabling classify the different approaches as sequential construction heuristics, constraint programming, and local search (genetic algorithms, simulated annealing and tabu search).

Heuristics used in solving graph coloring problem are examples of sequential construction heuristics. These heuristics have been widely applicable for automated exam timetabling since 1960's ([7]). Sequential construction heuristics order the exams in some way (for example, largest exam first), and try to allocate each exam to a session in order, while satisfying all the constraints.

One of the major approaches in exam timetabling over the years has been constraint programming approach. Examples of this approaches are provided by Boizmault, Delon and Peridy, 1996 ([3]), David, 1997 ([12]) and Reis, Teixeira and Oliveira, 2000 ([18]). Boizmault et al. used the constraint logic programming language CHIP to solve the exam timetabling problem. David developed a local repair technique to solve exam timetabling problem modeled as a constraint satisfaction problem. Reis et al. used the constraint logic programming language ECLIiPSe to solve the exam timetabling problem.

Local search approaches play an important role in the exam timetabling literature. White and Xie([20]) and Di Gaspero and Schaef ([13]) used tabu search method in exam timetabling. White and Xie kept two tabu lists, the usual short-term tabu list, and a long-term tabu list keeps track of the most moved exams. Di Gaspero and Schaef used a single tabu list, but when exams are added to this list, it is for a randomly determined number of iterations.

Over the last few years simulated annealing has been investigated for exam timetabling with some level of success. In 1991, Abramson ([1]) applied simulated annealing to construct high school timetables. In 1996 Thompson and Dowsland ([19]) considered an adaptive cooling technique where the temperature is automatically reduced or increased depending upon the success of the move. The two authors employed Kempe chain as neighborhood structure. In 2001, Burke et al. ([6]) proposed a time-predefined variant of simulated annealing.

Another local search approach is genetic algorithm, which is also widely used in automated timetabling. Examples of genetic algorithms for exam timetabling are provided by Corne, Ross and Fang, 1994 ([8]), Burke, Elliman and Weave, 1995, ([4]), Corne and Ross, 1996 ([9]), Burke, Newall and Weave, 1996 ([5]).

In this paper, we present a new solution method for examination timetabling, consisting of two phases: a constraint programming phase to provide an initial solution and a simulated annealing phase to improve the quality of solution. The simulated annealing applies Kempe chain neighborhood and includes a mechanism that allows the user to define a certain period of time in which the algorithm should run. We perform preliminary experiments of the algorithm on the real data set from the HoChiMinh City (HCMC) University of Technology.

The approach presented here is close in spirit to the one described in [16]. However, the main difference between two approaches is that our simulated annealing phase is equipped with more refined mechanisms that help to

Tuan-Anh Duong and Kim-Hoa Lam are with Department of Information Technology, HoChiMinh University of Technology (corresponding author to provide e-mail: dtanh@ dit.hcmut.edu.vn).

determine crucial cooling schedule parameters.

II. EXAMINATION TIMETABLING PROBLEM

The development of an examination timetable requires to schedule a number of examinations ('exam') in a given set of exam sessions ('time slot' or 'session') so as to satisfy a given set of constraints.

The most common forms of constraints in the exam timetabling at the HCMC University of Technology are:

C1. Exam Clashing: No student may have two exams in the same session.

C2. Room Clashing: Not more than two exams can be assigned to the same room.

C3. Student Restrictions: Each student's exams should be spread over the exam session.

C4. Room Restrictions: When there are several student subgroups taking the same exam, rooms for that exam with all student groups should be arranged near to one another.

C5. Release/Due Date Restrictions: Some exams have to start after a release date and to finish before a due date.

C6. Room Utilization: Rooms are assigned to exams in such a way that the room utilization can be maximized.

C7. Predefined Room Constraint: Some exams must be assigned special rooms.

Notice that in HCMC University of Technology, students enrolling in courses are divided into *subgroups*. These subgroups are the units used when assigning students to rooms. Besides, the exam sessions can be of variable length.

The first two kinds of constraints, C1 and C2, are hard constraints that must be satisfied, but the rest can be considered as soft constraints that should be satisfied to improve the solution.

III. A TWO-STAGE METHOD

The approach we take to the exam timetabling problem consists of two stages, providing a hybrid method:

1. Constraint Programming: to obtain an initial feasible timetable.

2. Simulated Annealing: to improve the quality of the timetable.

The first stage is used primarily to obtain an initial timetable satisfying all the hard constraints. The second stage aims to improve the quality of the timetable, taking the soft constraints into account. The method used in the second stage is optimization method, which will seek to optimize a given objective function.

The combination of the two different approaches in our exam timetabling problem aims at taking advantage of the strong points of each approach by using each method under circumstances where it performs best. This is a relatively new trend in automated timetabling (much of the work being pioneered by researchers in Nottingham). A similar sequential hybrid approach has been taken in work on another research: White and Zhang ([21]) use constraint programming to find a starting point for tabu search in solving course timetabling problems.

Notice that the examination timetabling problem can be seen as consisting of two subproblems: (1) assigning sessions to exams and (2) assigning student groups to rooms. For real-life situations, these two subproblems can not be solved separately. Due to limited space, the solving method for the room assignment will not be described in this paper.

A. *Constraint Programming*

It is well-known that the initialization strategy for the SA algorithm could have a crucial influence on the performance of the algorithm, especially the search space is disconnected, which is typical for exam timetabling problem. So we have to make the initial solution as good as possible in as little time as possible. Constraint programming is a good choice for this criterion.

The constraint programming algorithm used in the first stage is *backtracking with forward checking* (BC-FC). The algorithm is a combination of consistency technique and chronological backtracking ([12], [15]). It performs arc-consistency between pairs of not yet instantiated variable and instantiated variable, i.e., when a value is assigned to the current variable, any value in the domain of a "future" variable which conflicts with this assignment is removed from the domain.

One important advantage of BC-FC is that it is easy to include into the algorithm some variable ordering and value ordering heuristics that improve its performance ([12]). In our BC-FC algorithm, we employs a dynamic variable ordering, known as "fail first" which selects as the next variable the one with the smallest number of values in its current domain. As for performing a variable ordering heuristic along with BC-FC, we assign a *priority score* for each exam and use it to order the exams to be scheduled. The priority score of an exam bases on its remaining domain size, its number of students and some other relevant factors ([10]).

B. *Simulated Annealing*

Simulated Annealing (SA) is a global stochastic optimization technique that has been widely used in several types of combinatorial optimization problems ([1], [2], [17], [19]). It is a variant of local search which allows uphill moves to be accepted in a controlled manner. The basic algorithm given in ([12]) is described in the Fig. 3.1.

The typical SA algorithm accepts a new solution if its cost is lower than the cost of the current solution. Even if the cost of the new solution is greater, there is a probability of this solution to be accepted. With this acceptance criteria it is then possible to climb out of local optima.

```
Select an initial solution $s_0$
Select an initial temperature $t_0 > 0$
Select a temperature reduction function $\alpha$;
**repeat**
  **repeat**
     Randomly select $s \in N(s_0)$;  /* s is a neighbor
solution of $s_0$ */
    $\delta = f(s) - f(s_0)$; /* compute the change in cost function*/
    **if** $\delta < 0$ **then** $s_0 = s$
    **else**
        generate random $x \in [0,1]$;     /* x is a random
                     number in range 0 to 1 */
        **if** $x < \exp(-\delta/t)$ **then** $s_0 = s$
        **endif**
    **endif**
  until iteration_count = nrep;
  t = α(t)
until stopping condition = true.
  /* s_0 is the approximation to the optimal solution */
```

Fig. 3.1 Simulated Annealing Algorithm

As it can be seen there are several aspects of the SA algorithm that are problem-oriented. Design of a good annealing algorithm is nontrivial, it generally comprises three components: (1) neighborhood structure, (2) cost function and (3) cooling schedule.

1) Neighborhood Structure

In order to apply the SA algorithm we must have a neighborhood structure which defines for each solution a set of neighboring solutions. This is the key component of any simulated annealing method.

The *neighborhood* we use is a variant of the Kempe chains neighborhood. A Kempe chain is determined by an exam *i* currently allocated session *t*, and another session $t' \neq t$. Let G be the set of all exams allocated session *t*, and G' be the set of all exams allocated session *t'*. From our definition of a solution, both G and G' are conflict-free sets of exams. The Kempe chains can be thought of as the (unique) minimal pair of sets of exams $F \subseteq G$ and $F' \subseteq G'$, such that $i \in F$ and both $(G \setminus F) \cup F'$ and $(G' \setminus F') \cup F$ are conflict-free sets of exams. For a given exam *i* in session *t* and other session *t'*, the Kempe chain (F and F') can easily be constructed by a simple iterative procedure. We call the timetable obtained by reallocating session *t* to all exams in F' and reallocating session *t'* to all exams in F a neighbor of the solution. The neighborhood of a solution is defined to be the set of all such neighbors.

In our simulated annealing algorithm, a current solution is maintained and a neighbor of the current solution chosen at random.

2) Cost Function

For the case of exam timetabling problem, the cost function tries to reflect the influence of the soft constraints we have mentioned in section II. The *time distance* between the two exams for the same student is one of the important factors. The shorter time distance will yield higher penalty score.

To give penalty score for an exam, we apply the same method of computing penalty score proposed in Burke et al. ([6]). Let t_i, t_j be the sessions assigned to exam *i* and exam *j*. F_c, the *penalty score* for a given time table in term of time distance or proximity between each pair of exams, is given as follows:

$$F_c = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C_{ij} . \text{prox}(i,j)$$

where *n* is the number of exams, C_{ij} is the number of students that take the two exams, *i* and *j*, and

$$\text{prox}(i, j) = 2^{6 - |t_i - t_j|} \quad \text{if } 1 \leq |t_i - t_j| \leq 5$$
$$\text{prox}(i, j) = 0 \quad \text{otherwise.}$$

Besides, penalty scores might be given for the timetables in which students have to take two exams in the same day. The penalty score F_1 for a given timetable is given as follows:

$$F_1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} C_{ij} . \text{adjs}(i,j)$$

where adjs(i,j)=1 if $|t_i - t_j| = 1$ and t_i, t_j are in the same day,

adjs(i,j)=0 otherwise.

Thus the cost function F can be calculated as the sum of those soft constraints. It can be seen in the following formula and should be minimized:

$F = F_c + F_1$.

To speed up the cost function calculation, we apply a *delta evaluation* in which the cost difference between the current solution and the neighborhood solution is evaluated. Since a neighbor solution is a timetable resulted from a swap between two exam sessions in the current timetable, we can calculate the cost difference between the current timetable and the neighbor timetable quickly.

3) Cooling Schedule

Cooling schedule, the manner in which the control parameter, designated by temperature t, is lowered during annealing is crucial. Thompson and Dowsland ([19]) discuss different cooling schedules for simulated annealing processes applied to exam scheduling problem. They find that slow cooling schedules are generally more effective, but that no cooling schedule is markedly better than any other.

Here we use a *geometric cooling schedule*, in which at every *nrep* iterations, the temperature, *t*, is multiplied by α, where *nrep* and α are given parameters of the algorithm. In our algorithm *nrep* = 6 and α is determined according to the number of steps the user wants for the SA algorithm. To determine *nrep*, we had to experiment with several different values, namely, 1, 4, 6, 10, 50 (with the number of SA steps 5000). Finally, we choose *nrep* = 6 which returns the best solution cost within an acceptable run time.

To determine *starting temperature*, we use a rough start temperature t_0 = 10000, and try to derive the real start temperature T_0 basing on the functional dependence between the starting acceptance probability χ_0 (70% to 80%) and the starting temperature T_0.

As proposed by Poupaert and Deville in [17], the functional dependence between the starting acceptance probability χ_0 and the starting temperature T_0 is given as follows:

$$\chi_0 = \chi(\{\delta_1, \delta_n, \delta_{n+1}, \ldots, \delta_m\}, T_0)$$
$$= \frac{1}{m} \sum_{i=1}^{n} \exp(-\delta_i/T_0) + (m-n)/m \quad (3.1)$$

where $\delta_i = f(s_i) - f(s_0)$, s_0 is the initial solution, s_i is a neighbor solution of s_0, *f* is the cost function, *m* is the size of neighbor solution space.

(For a timetable with *n* exams, the size of a neighbor solution space is n*(n-1)/2).

We have to develop a small algorithm that can derive the starting temperature T_0 from the starting acceptance probability χ_0 (70% to 80%) using the equation (3.1). This algorithm has to be run only once for each execution of the SA algorithm. The algorithm is given in Fig. 3.2

```
Step 1: $m := n(n-1)/m$ ;   /* n is the number of exams  */
        compute $\delta_i$ , $1 \le i \le m$;
        $t_0 := 10000$;
        $t := t_0$; $j := 0$;
        **repeat**
          $j := j+1$; $t := t*j$;
          compute $\chi = \chi(t_0)$ using (3.1);
        **until** $\chi \ge 0.8$;
Step 2: $t_{end} := t$; exit := false;
        **repeat**
          $t = (t_0 + t_{end})/2$;
          compute $\chi = \chi(t)$ using (3.1);
          **if** $0.7 < \chi < 0.8$ **then** exit := true
          **else if** $\chi \le 0.7$ **then** $t_{end} := t$ **else** $t_0 := t$
        **until** exit;
        /* t is the desired starting temperature  */
```

Fig. 3.2 Algorithm to determine starting temperature.

As for *final temperature* T_f, since there are no accurate recommendations for the value in literature, we had to experiment with several final temperatures, namely, 0.5, 0.05, 0.005, 0.0005, 0.00005 (with the number of SA steps 5000). Finally, we choose a fixed value ($T_f = 0.005$) which returns the best solution cost.

To determine the *reduction parameter* α for geometric cooling, we apply the formula proposed by Burke et al. ([6]) that allows us to define a value for the parameter α based on the predefined time we want the simulated annealing to run for. The time we want the SA algorithm to run for is represented in the number of SA steps, N_{move}.

$$\alpha = 1 - (\ln(T_0) - \ln(T_f))/N_{move}. \qquad (3.2)$$

With the fixed values for T_0 and T_f, using (3.2) we can compute a value for the parameter α based on the predefined time (N_{move}) that the user wants the SA algorithm to run for. This mechanism is called by Burke et al. ([6]) *time-predefined simulated annealing*. The mechanism is very helpful for our research. It not only helps to increase the efficiency of the SA algorithm but also helps to make simulated annealing experiments easier.

C. Experimental Results

We implemented the exam timetabling program with Microsoft Visual C++ 6.0 and experimented on a Pentium II 450 MHz PC. Typical run time on the real data set from HCMC University of Technology (about 324 exams per semester in 30 exam sessions, based on 64000 enrollments) equates to less than 2 minutes for the constraint programming stage. The first stage always provides a good intial feasible timetable in a very little time.

As for the simulated annealing phase, run times on the same computer resources with the number of SA steps, N_{move}, changing from 500 to 70000 are given in the following table.

TABLE 1
RUNTIMES FOR SIMULATED ANNEALING STAGE

N_{move}	500	1000	5000	10000	20000	50000	70000
Run time (sec)	326	358	599	922	1664	3026	4150

So, the running time of the SA stage for all 324 exams with 50000 SA steps takes about 3026 sec ≈ 50 minutes. The effect of the number of the SA steps on cost is given in Fig. 3.3.

The figure shows the fact that with the number of SA steps high enough, i.e. the rate of cooling slow enough, the solution cost will improve a lot, i.e. a good quality solution comes out, but when the number of SA steps is already too high, the solution will not improve much.

It is quite clear that the longer the hybrid algorithm is allowed to run, the better the quality of the solution comes out.

Our experiments have shown the importance of the cooling parameters and the difficulties associated with setting them at the right levels. These parameter values are determined empirically by repeatedly running the algorithm with different values until the quality of the solutions produced by the algorithm ceases to improve. It is interesting to note that similar difficulties would be also encountered with tabu search method ([13]).

In comparison to the pure constraint programming approach ([10]), or the graph coloring method ([14]) for exam timetabling on the same data set have been experimented at

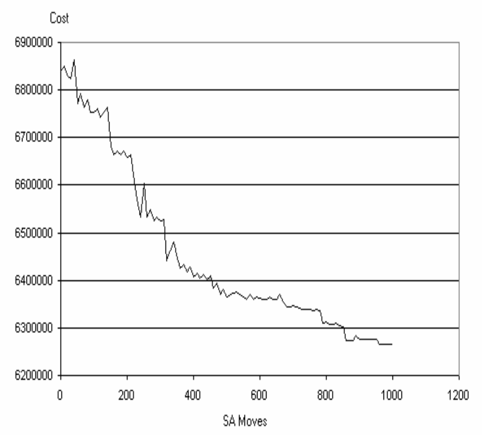

Fig. 3.3a Effect of the number of SA steps on solution cost (from 0 to 1200 SA steps).

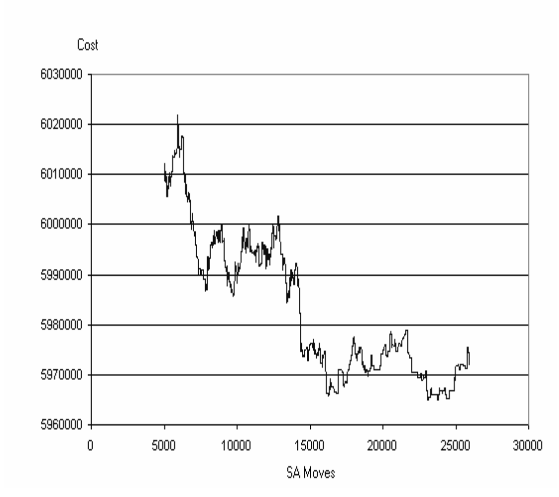

Fig. 3.3b Effect of the number of SA steps on solution cost (from 0 to 30000 SA steps).

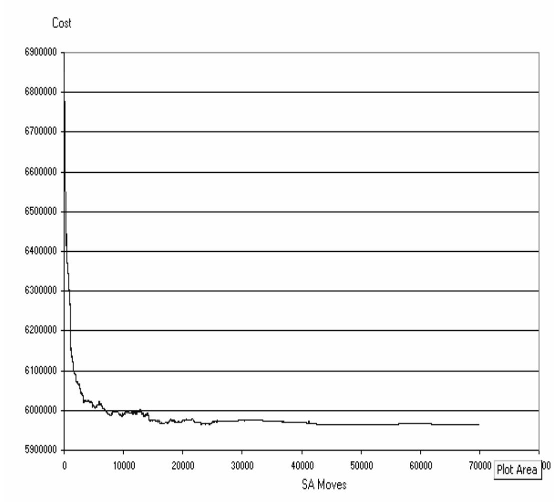

Fig. 3.3c Effect of the number of SA steps on solution cost (from 0 to 70000 SA steps).

HCMC University of Technology, the proposed hybrid algorithm takes much more run time, but yields a quite better solution for highly constrained problems.

However, we have not yet experimented some other hybrid approaches, for example, the memetic algorithm ([5]) which is a genetic algorithm that incorporates local search method or the combination of constraint programming and Tabu search as in [21], in order to compare the performance of the two different hybrid approaches.

IV. CONCLUSION

We have successfully combined constraint programming and simulated annealing for the problem of exam timetabling with real data sets. Our main conclusion from this work is that we can solve a very difficult scheduling problem with simulated annealing but we must be very careful in the way we choose to implement the SA components that are problem-dependent. We have applied Kempe chain as neighborhood structure, a special technique for determining starting temperature T_0 and a mechanism that allows the user to define a certain period of time in which the algorithm should run. The mentioned mechanism not only helps to increase the efficiency of the SA algorithm but also helps to make simulated annealing experiments easier.

As for cooling schedule, our results show that wherever possible, very slow cooling should be used. Typical run times for very slow cooling equate to no more than one hour and a half run on a Pentium II PC. The examination timetabling will be solved one or twice a semester and as the need for the timetable is not instant such the run time is acceptable.

If less solution time is available, the balance between solution quality and search time can be achieved through the predefined-time simulated annealing mechanism incorporated in our SA algorithm.

In spite of the shortcomings of the comparisons, the hybrid method still prove as a promising algorithm, among the best currently is used for examination timetabling. The constraint programming stage provides a fast way to the first feasible solution. This solution is improved by the simulated annealing stage.

In a future work we plan to try the hybrid two-stage method consisting of constraint programming and tabu search for exam timetabling problem, and to compare results between the two different hybrid methods on the same data set.

We conjecture that the dominant methods of the future for the examination timetabling problem will combine solution construction with local search. The stage of the hybrid approach may be integrated more fully, to yield a more powerful and robust algorithm.

REFERENCES

[1] D. Abramson, “Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms”, *Management Science*, vol. 37, no.1, 98-113, 1991.

[2] D. Abramson, M. Krishnamoorthy, and H. Dang, “Simulated Annealing Cooling Schedules for School Timetabling Problem”, 1997. Available: http://citeseer.nj.nec.com/104097.html

[3] P. Boizumault, Y. Delon and L. Peridy, “Constraint Logic Programming for Examination Timetabling”, *Journal of Logic Programming*, 1995, 1-17.

[4] E. K. Burke, D. G. Elliman and R. F. Weave. “A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems”, Proceedings of *the 6th Int. conf. on Genetic Algorithms* (Pittsburg, USA), 15-19 July 1995

[5] E. Burke, E., J. P. Newall, and R. F. Weare, “A Memetic Algorithm for University Exam Timetabling”, In: *Practice and Theory of Automated Timetabling, First International Conference, Selected Papers*, E. Burke & P. Ross (Eds.), Edinburgh, U.K., August/September 1995, Lecture Notes in Computer Science 1153, Springer-Verlag, 241-250, 1996.

[6] E. Burke, Y. Bykov, J. Newall and S. Petrovic, “A Time-Predefined Local Search Approach to Exam Timetabling Problems”, Computer Science Technical Report No. NOTTCS-TR-2001-6, Univ. of Nottingham, 2001.

[7] M. W. Carter, and G. Laporte, “Recent Developments in Practical Examination Timetabling”, In: *Practice and Theory of Automated Timetabling, First International Conference, Selected Papers*, E. Burke & P. Ross (Eds.), Edinburgh, U.K., August/September 1995, Lecture Notes in Computer Science 1153, Springer-Verlag, 3-21, 1996.

[8] D. Corne, P. Ross and H. L. Fang. “Fast Practical Evolutionary Timetabling”, In: *Evolutionary Computing*, T. Fogarty, Ed., 250-263, 1994.

[9] D. Corne, and P. Ross, “Peckish Initialization Strategies for Evolutionary Timetabling”, In: *Practice and Theory of Automated Timetabling, First International Conference*, *Selected Papers*, E. Burke & P. Ross (Eds.), Edinburgh, U.K., August/September 1995, Lecture Notes in Computer Science 1153, Springer-Verlag, 227-240, 1996.

[10] B. B. Dang, and N. T. Quang, “Constraint Satisfaction Approach to University Examination Timetabling”, B.Eng. Thesis, Dept. of Information Technology, HCMC University of Technology, 2003.

[11] P. David, “A Constraint-Based Approach for Examination Timetabling Using Local Repair Techniques”, *The Practice and Theory of Automated Timetabling: Selected Papers PATAT 1997*, E. Burke and M. Carter, Eds. , Lecture Notes in Computer Science 1408, Springer-Verlag, 168-186, 1998.

[12] R. Dechter, and D. Frost, “Backjump-based Backtracking for Constraint Satisfaction Problems”, *Artificial Intelligence*, vol. 136, 147-188, 2002.

[13] L. Di Gaspero, and A. Schaerf. “Tabu Search Techniques for Examination Timetabling”. *The Practice and Theory of Automated Timetabling PATAT 2000, Selected Papers,* E. Burke and W. Erben, Eds. ,. Lecture Notes in Computer Science 2079, Springer-Verlag, 104-117, 2000.

[14] N. H. Hai, D. T. Anh, “A Multi-Strategy Approach in University Examination Scheduling”, Proceedings of *School on Scientific Computing and Applications*, March, 2002, HCMC University of Technology, 141-150.

[15] V. Kumar, “Algorithms for Constraint Satisfaction Problems: A Survey”, *AI-Magazine*, 32-44, 1992.

[16] L. T. G. Merlot, N. Boland, B. D. Hughes, & P. J. Stuckey, “A Hybrid Algorithm for the Examination Timetabling Problem”, Proc. *Practice and Theory of Automated Timetabling* (PATAT’2002), 2002.

[17] E. Poupaert, Y. Deville, “Simulated Annealing with Estimated Temperature”, *AI Communication*, vol. 13, 2000, 19-26.

[18] Reis, L.P., P. Teixeira and E. Oliveira. Examination Timetabling using Constraint Logic Programming. Proceedings of *3rd Int. Conf. on the Practice and Theory of Automated Timetabling* PATAT 2000, Konstanz, Germany, 181-183.

[19] J. Thompson, and K. A. Dowsland, ”General Cooling Schedules for a Simulated Annealing Based Timetabling System”, In: *Practice and Theory of Automated Timetabling, First International Conference: Selected Papers*, (E. Burke & P. Ross , Eds.), Edinburgh, U.K., August/September 1995. Lecture Notes in Computer Science 1153, Springer-Verlag, 345-363, 1996.

[20] G. M. White and B. S. Xie, “Examination Timetables and Tabu Search with Long-term Memory” *The Practice and Theory of Automated Timetabling PATAT 2000, Selected Papers*. E. Burke, W. Erben, Eds., Lecture Notes in Computer Science 2079, Springer-Verlag, 85-103, 2000.

[21] G. M. White and J. Zhang, “Generating Complete University Timetables by Combining Tabu Search with Constraint Logic”, *The Practice and Theory of Automated Timetableing PATAT 1997, Selected Papers*. E. Burke, W. Erben, Eds., Lecture Notes in Computer Science 1408, Springer-Verlag, 187-198, 1998.