A TIME-PREDEFINED APPROACH TO COURSE TIMETABLING

***Edmund Burke*[1]*, Yuri Bykov*[1]*, James Newall*[2]*, Sanja Petrovic*[1]**

[1] *Automated Scheduling, Optimisation and Planning Group*

School of Computer Science & IT, University of Nottingham,

Wollaton Road, Nottingham NG8 1BB, UK

{ekb,yxb,sxp}@cs.nott.ac.uk

[2] *EventMAP Limited*

21 Stranmillis Road, Belfast BT9 5AF, N. Ireland

Jim.Newall@eventmaponline.com

ABSTRACT

A common weakness of local search metaheuristics, such as Simulated Annealing, in solving combinatorial optimisation problems, is the necessity of setting a certain number of parameters. This tends to make significantly increase the total amount of time required to solve the problem and often requires a high level of experience from the user. This paper is motivated by the goal of overcoming this drawback by employing "parameter-free" techniques in the context of automatically solving course timetabling problems.

We employ local search techniques with "straightforward" parameters, i.e. ones that an inexperienced user can easily understand. In particular, we present an extended variant of the "Great Deluge" algorithm, which requires only two parameters (which can be interpreted as search time and an estimation of the required level of solution quality). These parameters affect the performance of the algorithm so that a longer search provides a better result - as long as we can intelligently stop the approach from converging too early. Hence, a user can choose a balance between processing time and the quality of the solution. The proposed method has been tested on a range of university course timetabling problems and the results were evaluated within an International Timetabling Competition. The effectiveness of the proposed technique has been confirmed by a high level of quality of results. These results represented the third overall average rating among 21 participants and the best solutions on 8 of the 23 test problems.

Keywords: Combinatorial optimisation, Metaheuristic, Local search, Timetabling

1. INTRODUCTION

Local search metaheuristics have been among the most successful approaches to solving combinatorial optimisation problems over the last few years. Local search is the common name for a group of methods which (on the whole) iteratively repeat the replacement of a current solution $\boldsymbol{s}$ by a new one $\boldsymbol{s}$*, until some stopping condition has been satisfied. The new solution is selected from a neighbourhood $\boldsymbol{N(s)}$ **-** the set of candidate solutions into which the current one can be transformed, usually by a single move. The quality of the solution is characterised by its fitness (cost function) $\boldsymbol{f(s)}$. The goal of the search process is to minimise the cost function.

Variants of this typical basic local search approach differ by their mechanisms of accepting or rejecting the candidate solution from the neighbourhood, definitions of neighbourhood and stopping conditions. A description of the local search methods and their applications to different combinatorial optimisation problems can be found in [1].

1.1 Hill-Climbing

The simplest local search algorithm is Hill-Climbing. This method was used for the timetabling problem as early as 1960 by Appleby et al. [4]. A candidate solution is accepted only if it has better or equivalent fitness than the current one. Hill-Climbing does not require the definition of any parameters and its behaviour is quite stable. It aims to converge very fast but often has a final solution of relatively poor quality as it tends to get trapped in local optima.

1.2 Simulated Annealing

Different extensions of Hill-Climbing allow the acceptance of worse solutions in order to eventually get better ones. These approaches widen the search space and can improve the quality of the result. One of the most widely studied local search metaheuristics is Simulated Annealing. It was proposed as a general optimisation technique in 1983 by Kirkpatrick et al. [18] and has been repeatedly applied to solve a wide range of problems. An overview of its different applications is given in [19].

Simulated Annealing is similar to Hill-Climbing but it accepts worse solutions with a probability: $\boldsymbol{P=e}^{-\delta/T}$, where $\boldsymbol{\delta = f(s^*) - f(s)}$ and the parameter $\boldsymbol{T}$ denotes the temperature (which is analogous to the temperature in the process of annealing). Originally it was suggested to start the search from a high temperature and reduce it to the end of a process by progression formula: $\boldsymbol{T_{i+1} = T_i}$ -

$\boldsymbol{T_i * \beta}$ (geometric cooling schedule). However, the cooling rate $\boldsymbol{\beta}$ and initial value of $\boldsymbol{T}$ are usually different for different problems and are often selected empirically. This uncertainty causes problems with the practical use of Simulated Annealing. This has been indicated in an early application of Simulated Annealing to the timetabling problem by Davis and Ritter in 1987 [10]. They reported that the manual enumeration of parameters took two weeks and therefore they developed a genetic algorithm especially for determination of the best values for the parameters.

Different improvements of the basic Simulated Annealing algorithm have been suggested, such as: an adaptive cooling technique where the temperature is reduced or increased depending on the success of the move [11], running the algorithm several times starting from a random seed [2], cooling schedules with a variable cooling rate and reheating [3], making the temperature dependent on the absolute value of the cost function [21], and the "mean field annealing" technique, where the search space is approximated by a system of thermodynamical differential equations [14]. However, the question about the best values of the parameters still has no definitive answer. Moreover some of the proposed improvements introduce new parameters.

1.3 The Threshold Acceptance Algorithm

A deterministic variant of the Simulated Annealing, known as the Threshold Acceptance method, accepts every worse solution, when $\boldsymbol{\delta}$ does not exceed some threshold $\boldsymbol{T}$. It was introduced by Dueck and Scheuer in 1989 [12]. They applied the algorithm to a Travelling Salesman Problem and claimed that their algorithm is superior to classical Simulated Annealing. Originally the authors suggested that the threshold should be decreased when the algorithm does not improve the solution for a long time. However, it is not clear when to do this and how much to decrease it by. Although the acceptance procedure was refined, it still involves a few parameters whose values are deduced empirically. The adaptive cooling scheme was introduced for the Threshold Acceptance method [17], but as in the previous case, it did not yield sensible practical benefits and this technique is not widely applied.

1.4 Contribution

In this paper we demonstrate the advantages of a new variant of local search metaheuristic for solving course timetabling problems with parameters which are easily understandable to the user. The description of this technique is given in Section 2. In Section 3 we define a problem instance and present the investigation of the algorithm's properties and its comparison with other techniques. Section 4 includes a summary of our study and an outline of some future work.

2. THE GREAT DELUGE ALGORITHM

In [13] Dueck introduced an algorithm, which accepts every solution whose objective function is less than or equal to the upper limit (level) ***B***. This method was called the "Great Deluge algorithm". The value of ***B*** is monotonically decreased during the search and bounds the feasible region of the search space. Usually this algorithm converges when the level "outruns" a current solution. In order to prevent a premature convergence (encourage current solutions to return into the feasible region) and thus, to improve the performance of this method we propose to extend it by accepting all the candidate solutions which are better than the current one. The pseudocode of this extended algorithm is given in Figure 2.1.

```
Set the initial solution $s$
Calculate initial cost function $f(s)$
Initial level $B_0=f(s)$
Specify input parameter $\Delta B$ = ?
While not some stopping condition do
    Define neighbourhood $N(s)$
    Randomly select the candidate solution $s^* \in N(s)$
    If ( $f(s^*) \leq f(s)$ ) or ( $f(s^*) \leq B$ )
    Then accept $s^*$
    Lower the level $B = B - \Delta B$
```

Figure 2.1 The extended Great Deluge algorithm

The initial value of level B_0 is equal to the initial cost function. This forestalls sharp descents and idle steps in the beginning of the search. Hence, only one input parameter ΔB, the decay rate at each step has to be specified. Although this parameter is not clearly understandable (straightforward) we show below in Section 3.2 that it can be interpreted as a function of expected search time and expected solution quality, which are relatively easy to specify.

3. AN EVALUATION OF THE GREAT DELUGE ALGORITHM FOR COURSE TIMETABLING

University course timetabling problems are known to be difficult real world problems that have been studied in some depth over the last few decades or so. The interested reader can see a more detailed description of the various approaches that have appeared over the years in the following recent survey/review papers: [6], [24]. Some research directions and some new approaches are discussed in [5].

3.1 Course Timetabling Problems

University course timetabling involves the scheduling of lectures (courses) within a given number of timeslots (periods) and their allocation into available rooms (usually on a weekly basis) while satisfying certain constraints. Generally, the constraints are classified as either hard or soft. Satisfaction of the hard constraints is a strict requirement, i.e. in a feasible timetable they should not be violated under any circumstances. Soft constraints can be violated, but it is important to minimise those violations. Thus the cost function of every solution indicates the number of violated soft constraints under the assumption that all the hard ones are satisfied or alternatively it introduces a very high cost for the violation of a hard constraint.

The prime hard constraint is caused by an obvious requirement that no one person can attend two lectures simultaneously. Therefore any two courses which clash (have common students) must not be placed into the same timeslot. Another usual hard constraint reflects a situation where not all of the rooms are suitable for particular courses. Therefore in a feasible solution, courses should be allocated into appropriate rooms, i.e. the facilities required for certain courses have to be available and the size of the room has to be big enough to accommodate all the students registered for the course. Course timetabling problems are often solved by different kinds of heuristic constructive techniques (e.g. [9], [20]) or constraint logic programming methods (e.g. [7], [15]). A description of a number of approaches is presented in [6].

Soft constraints usually differ from university to university. In our study we address the set of soft constraints which are described in the rules of the International Timetabling Competition organized by the EU Metaheuristics Network and sponsored by the Practice and Theory of Automated Timetabling IV (PATAT IV) conference in 2003. These constraints generate a penalty when:

- a student has only one lecture in a day,
- a student has more than three consecutive lectures in a day,
- a student has a lecture in the last timeslot of a day.

The objective function is calculated as the sum of the number of violations of these constraints.

In addition to Simulated Annealing, various other metaheuristics have been applied to university course timetabling: tabu search (e.g. [16], [23]), genetic algorithms (e.g. [8], [25]) and their hybrids with Hill-Climbing (memetic algorithms) (e.g. [22]).

3.2 The Progress Diagram

As mentioned above, in our experiments we used the course timetabling data given in the International Timetabling Competition. It is located at [26], and comprises 23 problem instances. Each problem instance consists of 350-440 courses, 10-11 rooms and 200-350 students. The number of timeslots is the same for all the problems and is equal to 45. We have carried out experiments on all these problems and our method showed a similar behaviour on all of them.

The investigation of the properties of the Great Deluge algorithm was started by generating progress diagrams such as that presented in Figure 3.1. The algorithm was implemented in Delphi 7 and run on a PC Celeron 2.2 GHz with OS Windows 98. The decay rate $\Delta \boldsymbol{B}$ was defined as $5*10^{-5}$. Every 50 000 moves, the current cost $\boldsymbol{F_C}$ and the number of moves $\boldsymbol{N_{mov}}$ were depicted as a point in the "time-cost" space. An example of a resulting diagram for the 1st problem instance is presented in Figure 3.1.

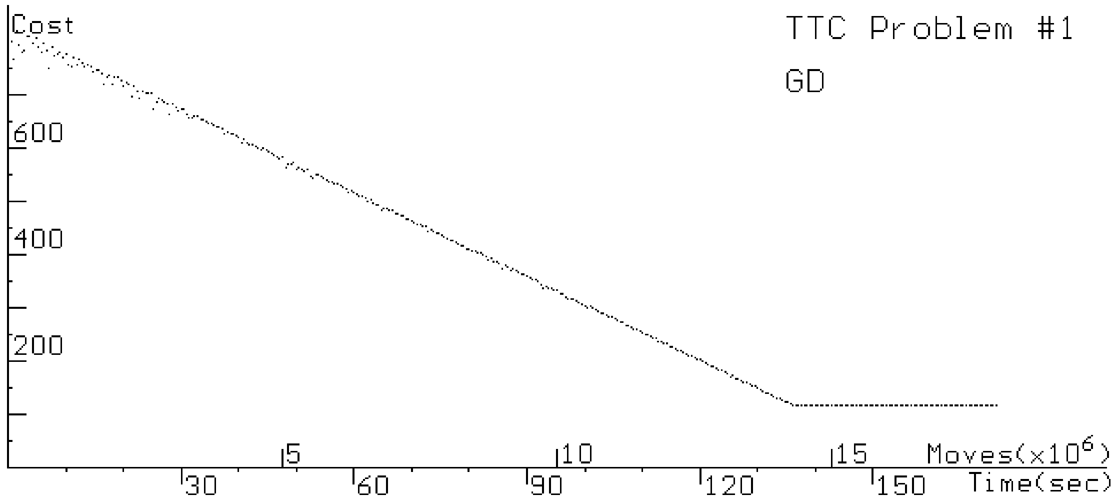

Figure 3.1 The progress of Great Deluge algorithm

This diagram shows two main properties of the Great Deluge algorithm:

1. The profile of the process is explicit. The search rigidly follows the decreasing of the level. Fluctuations are visible only at the beginning but later on, all intermediate solutions lie close to the line $\boldsymbol{F_C = B_0 - \Delta B * N_{mov}}$.

2. The point of convergence is quite recognisable. When a current solution reaches the value, where any further improvement is impossible, the search rapidly converges and the diagram becomes level. This moment can be easily detected in order to terminate the search procedure.

However, the point of convergence is uncertain and problem-dependent. Therefore if some information about the range of possible results is available, we suggest using it for reducing the number of idle steps. If we estimate the cost function of a desired result as ***f(s')*** we can calculate ***ΔB*** by formula (3.1).

$$\Delta B = \frac{B_0 - f(s')}{N_{mov}} \tag{3.1}$$

Usually such an approximation is quite possible. For example, in our following experiments we approximated ***f(s')*** by employing the results of a simple Hill-Climbing algorithm. Obviously, the correct specification of a processing time and an expected value of the cost function can be provided by a timetabling officer (in contrast to temperatures and a cooling rate in the Simulated Annealing).

3.3 Time-Cost Diagrams

The influence of processing time on the performance of the method was investigated while running the algorithm several times for a different predefined number of moves. The results are presented as "time-cost" diagrams where every point corresponds to the final cost function and the processing time of a separate solution. The example of such a diagram for the 2nd problem instance is shown in Figure 3.2.

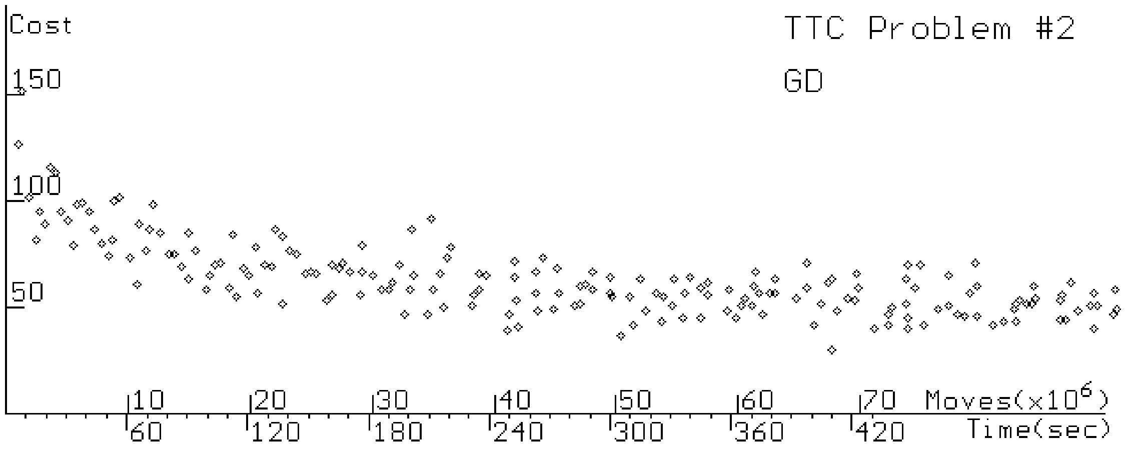

Figure 3.2 Time-cost diagram of Great Deluge algorithm

Even though the results are relatively scattered (which is not surprising), the clear tendency in this diagram can be observed: *a longer search produces better results*. The algorithm allows a user to improve the quality of the solution but he/she should pay a price for it with an increase in the amount of processing time. This is not valid for other metaheuristic approaches where the search can be stuck in a local optimum and no additional time can enable it to move out.

The reasonable balance between time and cost depends on the user's opportunities and preferences. In some cases the user requires results quickly but in other situations it is more preferable to spend much more time to search for a high quality solution. In the case of timetabling, very fast but relatively poor results cannot be considered as a best choice. In most situations the calculation of the solution is just part of the process, which includes the preparation of input data and the administering of the results of the software. Commonly it takes several days (if not weeks). The renewing of data is infrequent (because a course timetable is normally produced once or twice a year). However, the high quality of the solution is very important as the timetable affects a high number of people. In this environment a searching procedure, which can last several hours, seems to be quite acceptable.

3.4 Comparison with Other Techniques

A comparison of our method with Simulated Annealing for the 3rd problem instance is shown in Figure 3.3 where the diagram for Simulated Annealing is marked by "SA" and the diagram for Great Deluge is marked by "GD". The Simulated Annealing algorithm was run several times with variations of the initial temperature from 10^{-2} to $2*10^{4}$ (our results confirmed that this is an appropriate interval). In order to get approximately the same execution time in both algorithms, the cooling rate β was varied from $5*10^{-8}$ to $2*10^{-5}$.

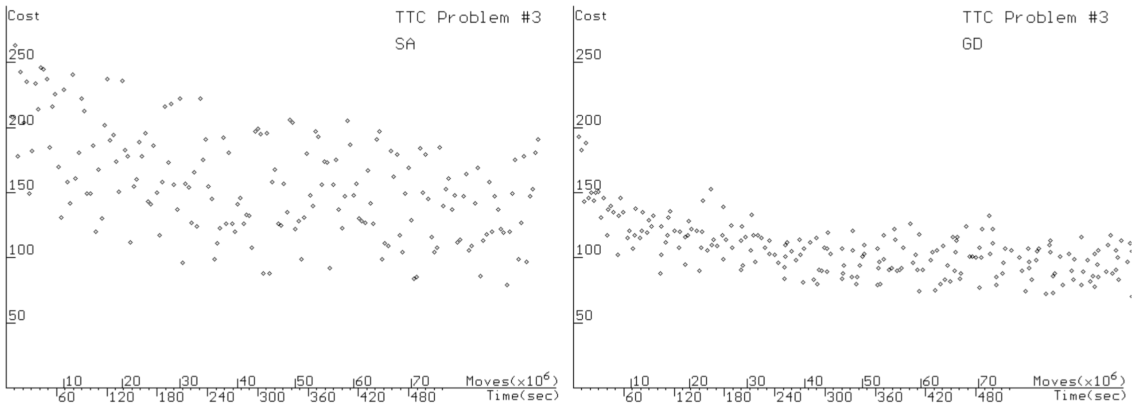

Figure 3.3 Comparison of Simulated Annealing and Great Deluge algorithms

The Simulated Annealing diagram shows a substantially higher scatter of results than the Great Deluge algorithm. A greater number of poor quality solutions are generated though the use of inappropriate parameter values. The superiority of the Great Deluge algorithm is obvious from these diagrams. Although both methods have approximately the same values of the cost function for the best results (in the given execution time), Simulated Annealing *can reach* it only with properly defined parameters, while Great Deluge *does* it always.

The employment of "straightforward" parameters significantly improves the effectiveness of the search. The deriving of the best cooling schedule requires several runs of the Simulated Annealing algorithm. Therefore its total processing time (from input to output) is several times longer than the processing time of a single run. With the Great Deluge approach all this time is spent in a single run (and it gets better results). Hence, with respect to the total time of the solving process, the performance of the Great Deluge is substantially better. Similar results are evident from the time-cost diagrams for the other problem instances.

The same comparison was carried out with the Threshold Acceptance algorithm. For the 8th problem instance the initial threshold was varied in the interval: 1-1000 and the rate of its decreasing was 10^{-8} - 10^{-3}. The resulting diagrams are presented on Figure 3.4, which shows the same behaviour as for Simulated Annealing (the diagram for Threshold Acceptance method is marked by "TA").

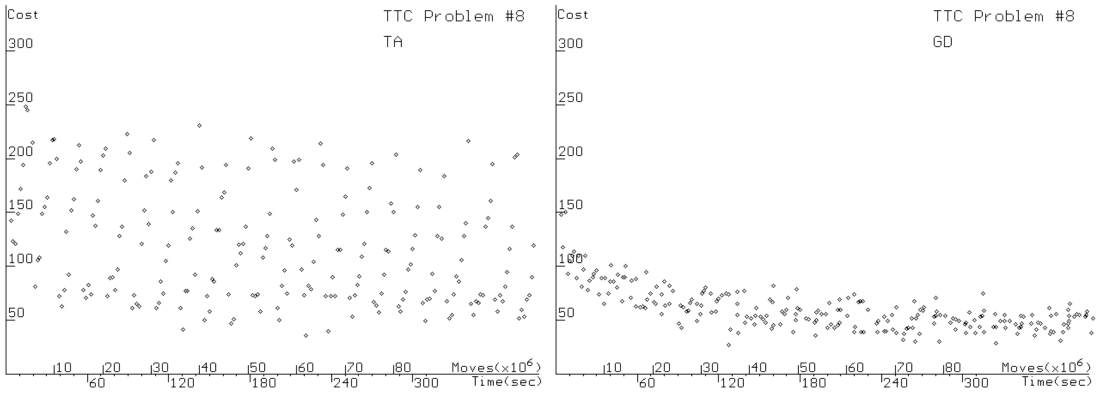

Figure 3.4 Comparison of Threshold Acceptance and Great Deluge algorithms

In the experiments on Hill-Climbing the time-cost diagrams were produced with a very short search time. The search time of Hill-Climbing depends on the stopping condition. We used the given number of idle steps as the stopping condition and it was varied in the range of 1-50000. The results for the 6th problem instance are presented in Figure 3.5 (the diagram for Hill-Climbing is marked by "HC").

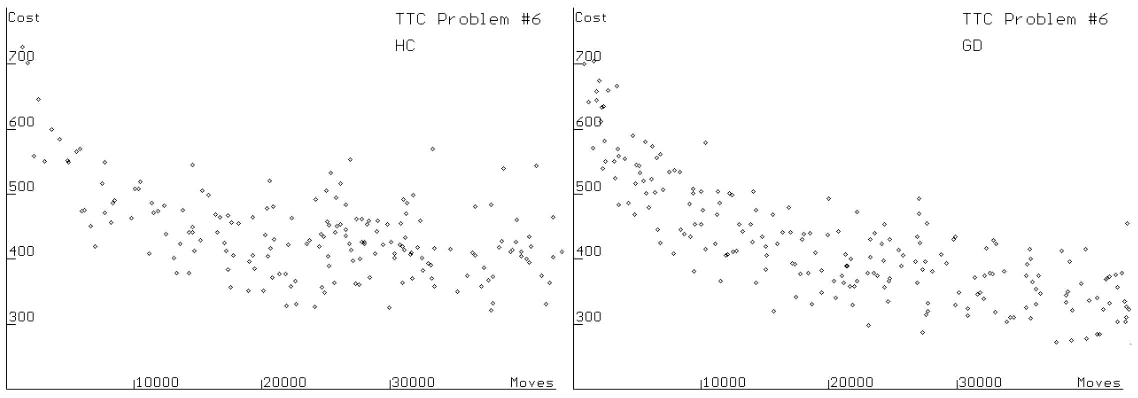

Figure 3.5 Comparison of Hill-Climbing and Great Deluge algorithms

Both diagrams have the same distribution of points at the beginning. However, the behaviour of those techniques in the right hand sides of the diagrams became different. If the chosen number of idle steps is too high – Hill-Climbing wastes this additional time, but Great Deluge uses it for improving the solution.

3.5 Evaluation of the Proposed Approach within an International Timetabling Competition

The participants in the competition submitted results without any information about the other participants. Also, all the solutions had to be obtained within the same time interval. In order to synchronise the time intervals on different hardware, a special test program was provided by the organising committee. In particular, on a PC Celeron 2.2GHz the processing time was limited to 726 seconds. Besides this, all submitted results were verified by the organising committee. In total, 21 participants (individual researchers and research teams) submitted solutions to all 20 problem instances. A comparison of the results of the 7 leading participants (including our results) is presented in Table 3.1. The best submitted results are shown in bold.

Table 3.1 The results, of the International Timetabling Competition

Place	1	2	3	4	5	6	7
Instance	P.Kostuch	B.Jaumard et al.	Our results	L.Di Gaspero and A.Shaerf	H.Arntzen and A.Lokketangen	A.Dubourg et al.	G.Toro and V.Parada
1	**45**	61	85	63	132	148	178
2	**25**	39	42	46	92	101	103
3	**65**	77	84	96	170	162	156
4	**115**	160	119	166	265	350	399
5	102	161	**77**	203	257	412	336
6	13	42	**6**	92	133	246	246
7	44	52	**12**	118	177	228	225
8	**29**	54	32	66	134	125	210
9	**17**	50	184	51	139	126	154
10	**61**	72	90	81	148	147	153
11	**44**	53	73	65	135	144	169
12	107	110	**79**	119	290	182	219
13	**78**	109	91	160	251	192	248
14	52	93	**36**	197	230	316	267
15	**24**	62	27	114	140	209	235
16	**22**	34	300	38	114	121	132
17	86	114	**79**	212	186	327	313
18	**31**	38	39	40	87	98	107
19	**44**	128	86	185	256	325	309
20	7	26	**0**	17	94	185	185

In addition to the submitted results, the organising committee checked the performance of participants' algorithms on three unseen problem instances. For one of these instances our algorithm produced the best solution among all the other participant's algorithms. The results on unseen instances are given in Table 3.2 where again the best ones are shown in bold.

Table 3.2 The results, produced on unseen instances

Unseen instance	P.Kostuch	B.Jaumard et al.	Our results	L.Di Gaspero and A.Shaerf	H.Arntzen and A.Lokketangen	A.Dubourg et al.	G.Toro and V.Parada
1	100	**86**	329	97	145	132	161
2	6	8	**3**	9	23	51	22
3	**72**	105	84	103	94	159	173

Among the given participants P.Kostuch used Simulated Annealing with a variation of the neighbourhood. All other participants used different variations of tabu search. The detailed descriptions of the applied techniques can be found in [26].

Although the processing time was relatively short (with longer available time our algorithm reached results with an even better cost value), our results were the best among all participants in 8 from the 23 problems. Moreover, among all the registered participants only our algorithm has provided a solution with a zero value of the objective function for problem instance 20 and hence has reached the global optimum of the problem. The competition procedures ranked the participants according to an average value. It is interesting to note that although our algorithm performed the best on 8 of the 23 problems it performed the worst on two of the problems (among the leading 7 participants). Indeed, it performed particularly poorly on problem 16. The reason why our algorithm fluctuated from the best to the worst is an area that is currently under investigation. We note that, in terms of the number of best solutions achieved, our algorithm comes second rather than third in the competition. Taking into account that our approach does not require additional time for tuning algorithmic parameters the results in Tables 3.1 and 3.2 confirm the high effectiveness of the presented technique.

4. CONCLUSIONS AND FUTURE WORK

This paper introduced an extended variant of the Great Deluge local search algorithm for course timetabling. The advantage of this method is that it requires the definition of only two parameters that correspond to search time and an estimation of desired solution quality. These parameters are problem-independent and have an obvious “real-world” meaning (thus they can be considered to be easily understood by university administration).

Our algorithm shows the clear trade-off between search time and the quality of the overall result, namely a longer search produces better solutions. This property allows the user to choose an acceptable processing time for each particular problem. The experiments with benchmark course timetabling problem instances confirm the effectiveness of the presented technique. For 8 out of 23 datasets, the Great Deluge algorithm achieved the best results among 21 compared algorithms.

Our future work will include evaluation of the algorithm in other domains. Additional issues will be investigated: how to choose good initial solutions, how to define non-linear level functions, hybridisation of the Great Deluge with other metaheuristics, etc. We should notice that the first and second place participants paid more attention to neighbourhood structures. Indeed, this is one of the most promising ways of improving the performance of timetabling algorithms and can be considered as an important direction of our future research.

BIBLIOGRAPHY

[1] Aarts, E. and Lenstra, J. K. (1997), "Local Search in Combinatorial Optimization (Wiley-Interscience Series in Discrete Mathematics and Optimization)", John Wiley & Sons, Chichester.

[2] Abramson, D. (1991), "Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms", Management Science, Vol. 37(1), 98-113.

[3] Abramson, D., Krishnamoorthy, M. and Dang, H. (1999), "Simulated Annealing Cooling Schedules for the School Timetabling Problem", Asia-Pacific Journal of Operational Research, Vol. 16, 1-22.

[4] Appleby, J. S., Blake, D. V. and Newman, E. A. (1960), "Techniques for Producing School Timetables on a Computer and their Application to other Scheduling Problems", The Computer Journal, Vol. 3, 237-245.

[5] Burke, E. K. and Petrovic, S. (2002), "Recent Research Directions in Automated Timetabling", European Journal of Operational Research –EJOR, Vol. 140(2), 266-280.

[6] Carter, M. W. and Laporte, G. (1998), "Recent Developments in Practical Course Timetabling", in Burke E., Carter M. (eds.): The Practice and Theory of Automated Timetabling II: Selected Papers (PATAT'97). Lecture Notes in Computer Science, Vol. 1408, Springer-Verlag, Berlin, Heidelberg, New York, 3-19.

[7] Cheng, C., Kang, L. Leung, N. and White, G.M. (1996), "Investigations of a Constraint Logic Programming Approach to University Timetabling", in Burke E., Ross P. (eds.): The Practice and Theory of Automated Timetabling: Selected Papers (ICPTAT '95). Lecture Notes in Computer Science, Vol. 1153, Springer-Verlag, Berlin Heidelberg, New York, 112-129.

[8] Colorni, A, M.Dorgio, V.Maniezzo, "Genetic Algorithms and Highly Constrained Problems: The Time-Table Case", in Goos, G., Hartmanis, J. (eds.): Parallel Problem Solving from Nature, Springer-Verlag, 1990, 55-59

[9] Cooper, T. B. and Kingston, J.H. (1993) “The Solution of Real Instances of the Timetabling Problem”, The Computer Journal, Vol. 36(7), 645-653.

[10] Davis, L. and Ritter, L. (1987), “Schedule Optimization with Probabilistic Search”, Proceedings of the 3rd IEEE Conference on Artificial Intelligence Applications (Orlando, Florida, USA, Feb. 1987), IEEE1987, 231-236.

[11] Dowsland, K. (1993), “Simulated Annealing”, in: Reeves C.R. (ed.) “Modern Heuristic Techniques for Combinatorial Problems”, Blackwell, 20-69.

[12] Dueck, G. and Scheuer, T. (1990), “Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing”, Journal of Computational Physics, Vol. 90, 161-175.

[13] Dueck, G. (1993) "New Optimization Heuristics. The Great Deluge Algorithm and the Record-to-Record Travel", Journal of Computational Physics, Vol 104, 86-92.

[14] Elmohamed, M. A. S., Coddington, P. and Fox, G. (1998) “A Comparison of Annealing Techniques for Academic Course Scheduling”, in Burke E., Carter M. (eds.): The Practice and Theory of Automated Timetabling: Selected Papers (PATAT '97). Lecture Notes in Computer Science, Vol. 1408, Springer-Verlag, Berlin Heidelberg, New York, 92-112.

[15] Gueret, G., Jussien, N., Boizumault, P. and Prins, C. (1996), “Building University Timetables Using Constraint Logic Programming”, in Burke E., Ross P. (eds.): The Practice and Theory of Automated Timetabling: Selected Papers (ICPTAT '95). Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg, New York, 130-145.

[16] Hertz, A. (1991), “Tabu Search for Large Scale Timetabling Problems”, European Journal of Operational Research, Vol. 54, 39-47.

[17] Hu, T. C., Kahng, A. B. and Tsao, C. W. (1995), “Old Bachelor Acceptance: A New Class of Non-Monotone Threshold Accepting Methods”, ORSA Journal on Computing Vol. 7(4), 417-425.

[18] Kirkpatrick, S, Gellat, J. C. D. and Vecci, M. P. (1983), “Optimization by Simulated Annealing”, Science, Vol. 220, 671-680.

[19] Koulmas, C., Antony, S. R. and Jaen, R. (1994) “A Survey of Simulated Annealing Applications to Operations Research Problems”, Omega International Journal of Management Science Vol. 22, 41-56.

[20] Laporte, G. and Desroches, S. (1986), “The Problem of Assigning Students to Course Sections in a Large Engineering School”, Computers and Operations Research, Vol. 13(4), 387-394.

[21] Melicio, F., Caldeira, P. and Rosa, A. (1999), “Solving the Timetabling Problem with Simulated Annealing”, Proceedings of the First International Conference on Enterprise Information Systems (ICEIS'99), 272-279.

[22] Paechter, B., Cumming, A., Norman, M. G. and Lucian, H. (1996), “Extensions to a Memetic Timetabling System”, in Burke E., Ross P. (eds.): The Practice and Theory of Automated Timetabling: Selected Papers (ICPTAT '95). Lecture Notes in Computer Science, Vol. 1153, Springer-Verlag, Berlin Heidelberg, New York, 251-265.

[23] Schaerf, A. (1996) “Tabu Search Techniques for Large High-School Timetabling Problems”, Proceedings of the 13th American Conference on Artificial Intelligence (AAAI-96) Portland, Oregon, USA August 1996, 363-368.

[24] Schaerf A. (1999), “A survey of automated timetabling”, Artificial Intelligent Review, Vol. 13, 87-127.

[25] Ueda, H., Ouchi, D., Takahashi, K. and Miyahara, T. (2001), “A Co-evolving Timeslot/Room Assignment Genetic Algorithm Technique for University Timetabling”, in Burke E., Erben W. (eds.): The Practice and Theory of Automated Timetabling III: Selected Papers (PATAT 2000). Lecture Notes in Computer Science, Vol. 2079, Springer-Verlag, Berlin Heidelberg, New York, 48-63.

[26] International Timetabling Competition home page: http://www.idsia.ch/Files/ttcomp2002/