10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

A Multi-Phase Hybrid Metaheuristics Approach for the Exam Timetabling

Ali Hmer and Malek Mouhoub

Department of Computer Science
University of Regina
Regina, Canada
{hmer200a,mouhoubm}@cs.uregina.ca

Abstract. We propose a Multi-Phase Hybrid Metaheuristics approach for solving the Exam Timetabling Problem (ETP). This approach includes a pre-processing phase, a construction phase and an enhancement phase. The pre-processing phase involves two stages: the propagation of ordering constraints and implicit constraints discovery stages. The construction phase uses a variant of the Tabu search with conflicts dictionary. The enhancement phase includes Hill Climbing (HC), Simulated Annealing (SA) and our updated version of the extended ”Great Deluge” algorithm. In order to evaluate the performance of the different phases of our proposed approach, we conducted several experiments on instances taken from the ITC 2007 benchmarking datasets. The results are very promising and competitive with the well known ETP solvers.

Keywords: Timetabling, Constraint Optimization, Metaheuristics.

1 Introduction

The examination timetabling [1, 2], is an annual or semi-annual problem for educational institutions. Due to its complexity and practicality, it is extensively studied by researchers in operational research and artificial intelligence. Many approaches have been proposed and discussed for solving the problem [1–7] using one or a combination of some of the following methods: graph-based, sequential techniques, clustering-based techniques, constraint-based techniques, metaheuristics, hyper-heuristics, multi-criteria techniques, and case-based reasoning techniques. In this paper we propose a Multi-Phase Hybrid Metaheuristics approach for solving the Exam Timetabling Problem (ETP). This approach consists of the preprocessing, construction and enhancement stages; and includes Tabu Search, Hill Climbing, Simulated Annealing and a modified version of Extended Great Deluge algorithms [8, 3] using metaheuristics techniques. The preprocessing phase is needed to prepare the work for the remaining two stages. During this phase, exams are sorted following the most constrained variables first heuristic [9] and implicit constraints are discovered using a form of transitive closure. During the construction stage a complete feasible solution is found using a variant of Tabu search along with conflicts dictionary to reduce cycling. In the enhancement phase a chosen metaheuristic is used. Once a solution can no longer be improved or reaches an idle state, another metaheuristic kicks in and used. The following metaheuristics are considered: Hill Climbing (HC) [10, 11], Simulated Annealing (SA) [12] and our updated version of the extended "Great Deluge" algorithm [8] which improves on the one proposed in [3].

In order to evaluate the performance of the different phases of our proposed approach we conducted several experiments on instances taken from the ITC 2007 benchmarking datasets [13]. The results are very promising and competitive with the well known ETP solvers. The rest of the paper is structured as follows. In the next section we will introduce the problem we are tackling. Section 3 presents our proposed solving approach. Experimental tests evaluating our solving method are then reported in Section 4. Finally, concluding remarks and future works are listed in Section 5.

2 Problem Description

2.1 Problem Formulation

Following the common formulations to the exam timetabling [14, 15] we model this problem as a constraint problem including the following.

Variable. Each exam is modeled as a problem variable defined over a domain of all possible assignments to that exam. An assignment is composed of a time period and a room.
Room Constraint. Exams are constrained by rooms seating capacity.
Student Constraint. This constraint prevents a student from being scheduled for more than one exam during a given time period.
Order constraint. This constraint is about exam ordering and precedence between two or more exams.
Same Duration Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in the same time slot.
Different Duration Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in different time slots.
Same Room Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in the same room.
Different Room Constraint. This constraint is about two or more exams that should/can (hard/soft) take place in different rooms.

2.2 Penalty Function

The penalty function is a problem dependent generic function to calculate the total cost/value of a given solution. Each soft constraint involves a single or multiple resources and violating it has its own penalty value that should be set in the problem description. The total penalty value of any solution is the sum of penalties of all violated soft constraints in the whole exam problem. Penalties correspond to violating soft constraints including the following.

1. Two exams in a row.
2. Two exams in the same day.
3. Mixed durations where two or more exams are taking place in the same room but have different durations.
4. Room penalty where using certain rooms implies specific penalty to discourage scheduling exam to them.
5. Period penalty where assigning exam to certain periods implies specific penalty.

3 Proposed Solving Approach

Our proposed solving approach consists of the following three main phases. A pre-processing phase followed by a construction and an enhancement phases.

3.1 Pre-processing phase

The difficulty of any exam timetabling depends on three factors; the number of students that enroll in it, direct student conflicts in that exam and its scheduling priority constraint, if any, among all exams scheduling. Following the idea of most constrained variables first [9], exams with most scheduling difficulty are scheduled first. The reason for this is that if scheduled late, they would most likely increase the potential of the un-assignment process of other exams that violate some constraints which eventually causes a backtracking. The pre-processing phase consists of following two stages.

1. Exam timetabling problem collections ordering.
2. The discovery of un-specified (implicit) hard constraints.

In the following we provide the details for each stage.

3.1.1 Problem collections ordering

In this stage a process takes place for the different collections that the exam problem consists of. These collections are exams, rooms, periods and students. Exams and students are usually large collections and pre-ordering those leads to a better performance and efficient results during search. In [16, 17] two of the well-known common techniques have been proposed to describe the ordering of exams based on difficulty criteria preceding their assignment to time slots. Our approach is slightly different from these techniques. It depends on a different concept revolving around our knowledge that large exam timetabling problems contain large exams, students and resources collections, and enhancing the way that we retrieve and lookup any element in these collections is a key in any efficient search algorithm. Indeed, the time complexity of looking up or retrieving an element from unsorted large collection is $O(n)$ whereas the time complexity of the same process in a sorted collection is $O(\log n)$.

Our approach of collections ordering pre-processing stage involves the creation of four ordered collections at the time of building problem variables, values and facts collections based on the efficient Microsoft .NET framework which implements the quicksort algorithm. It is worth mentioning that quicksort makes $O(n \log n)$ comparisons in average to sort n items. Prior to our decision on whether to perform this stage or not, we thought of two issues; the time needed to build large sorted collections and the time required to lookup or retrieve any element in these collections. After examining the different types of collections that Microsoft .NET framework provides, we made the decision to use Sorted Generic Lists for all variables, values and constraints collections. There are two reasons for that. First, we only need to build them once at the beginning of the problem modeling and hence we produce them in a sorted manner. This is the only time we spent to sort them. They also do not consume a lot of memory as other types of collections. In fact, they are the least memory consuming collection. The second reason is that after building any of the problem collections, we only need to do lookups which a generic list is good at and one of the fastest collections for that matter and its time complexity is $O(\log n)$. Table 1 shows the time complexity for adding an element and for looking up or retrieving an element from both unordered and ordered collections. Although, we include the cost of removing an element as in all large collections, as we only build any collection once and lookup or retrieve afterward and there is no need for removals. In the case of collections that need items removals we use hash sets as the cost of removing an item is considerably small.

Table 1. Time Complexity for Ordered and Unordered Collections

	Adding elt	Lookup/Retrieval	Removing elt
Unordered Collection	$O(1)$	$O(n)$	$O(n)$
Ordered Collection	$O(n)$	$O(\log n)$	$O(n)$

3.1.2 Discovery of implicit hard constraints

In this stage we have developed a technique to discover all hard constraints that were not explicitly defined in the problem. In any large COP problem that contains a large collection of variables, values and constraints, there is always the possibility of missing some of the hard constraints that depend on some of the declared ones. Our approach is to provide a pre-processing stage that discovers these unspecified constraints and add them to the problem constraints collection. In fact our goal is to add other constraints that should be known before assigning a value to a variable which in essence might eliminate some of the variables domain values and hence preventing a backtracking process, which would occur later on, if these additional constraints were not specified.

The exam timetabling problem usually contains the following three types of exam based constraints [14, 15].

Exam Ordering. Two or more exams scheduling should appear in a particular order. For example: exam 1 should take place after exam 2.
Exam Coincidence. Two or more exams should take place at the same time.
Exam Exclusion. Two or more exams should take place at different times.

This pre-processing stage is based on creating three full graphs for each type of these constraints where nodes represent the exams and edges are the hard constraints between exams. Then by traversing each graph, we try to discover same or different type of constraints between other exams in the same graph. The following are the four steps we use to achieve such discovery.

1. **Propagating ordering constraints that belong to concurrent exams.** For example, if a coincidence constraint declares that exam 1 must take place at the same time as exam 2 and another ordering constraint states that exam 2 must take place after exam 3. In this case, we need to add a new ordering constraint stating that exam 1 must take place after exam 3.
2. **Propagating ordering constraints by transitive closure.** For example, if exam 1 must be scheduled after exam 2 and exam 2 must be scheduled after exam 3 then this implies adding a new ordering constraint which states that exam 1 must be scheduled after exam 3.
3. **Propagating coincidence and exclusion constraints.** For example, if there is a coincidence constraint stating that exam 1 must take place in the same period as exam 2 and another distinct constraint stating that exam 2 must take place in a different period than exam 3 then a new exclusion constraint must be added to state that exam 1 and exam 3 must take place in different periods.
4. **Propagating the largest exam period of the same exam set to all other exams.** This happens when all exams are involved in the same coincidence constraint. For example, if there is a coincidence constraint involving three exams, exam 1, exam 2, and exam3, with a respective periods of 3 hours, 2 and half hours and 2 hours. Then all three exams anticipated periods should be updated to be equal to the largest (3 hours). It should be mentioned that any penalty cost that involves periods, when calculated, uses the original period and not the updated one.

3.2 Construction phase

In the construction phase a complete feasible solution is found using Tabu search metaheuristic along with conflicts dictionary to reduce cycling. Conflicts dictionary essentially is a dictionary data structure based consisting of a key and value and is used for its performance capability. Each entry in the conflicts dictionary represents a count for the number of conflicts that an assignment causes during search. In future search iterations, the entry with the highest counts are avoided and regarded as tabu. Utilizing Tabu search metaheuristics with conflicts dictionary can be further detailed as follows. As the search is only considered by variable and value selection criteria, the algorithm initially tries to find those variables that are most problematic to assign. Usually, a variable is randomly selected from unassigned variables that have the smallest domain size and less number of hard constraints. It then attempts to select the best value to assign to the selected variable using conflicts dictionary. A best value is one where its assignment improves the overall value of the solution. Also, any value that violates a fewer number of hard constraints is considered. In other words, when assigning a value to a given variable, the algorithm is looking to minimize the number of conflicting variables that need to be unassigned in order to reach or keep a solution feasible after assignment. A value is selected randomly if there is more than one value with such conditions. Soft constraints violations are totally ignored in this phase as they might affect the algorithm performance when searching for complete feasible solutions.

3.3 Enhancement phase

In the enhancement phase, a combination of three metaheuristics is employed and we can select just one, two or three out of theses metaheuristics. Whatever a metaheuristic is used, a local optimum is found. Once a solution can no longer be improved or reaches an idle state, another metaheuristic technique kicks in and is used. In our algorithm we used three of the well-known metaheuristics. These are Hill Climbing (HC) [10, 11], Simulated Annealing (SA) [12] and our Modified Extended Great Deluge (MEGD). MEGD is altered to allow some alternations of the bound that is imposed on the overall solution value. The search ends after a predetermined time limit has been reached. The best solution found within that limit is returned. Our MEGD is based on the Extended Great Deluge (EGD) [8] method which in turn is based on the original Great Deluge (GD). GD was introduced by Dueck [18] as a cure to SA requirement to find a cooling schedule for a particular instance of a given problem. GD algorithm starts with a ″water level″ equal to the initial solution value, and a preconfigured rate usually named ″tolerance rate″ to decrease that water level. The predetermined rate is the only parameter for this algorithm and this is one of this algorithm's advantages. GD accepts worsening solutions if the penalty cost is less than the water level. This latter is decreased by the pre-determined rate set for every iteration. Due to the advantage of using less parameters, GD has been used in several other implementations of metaheuristics.

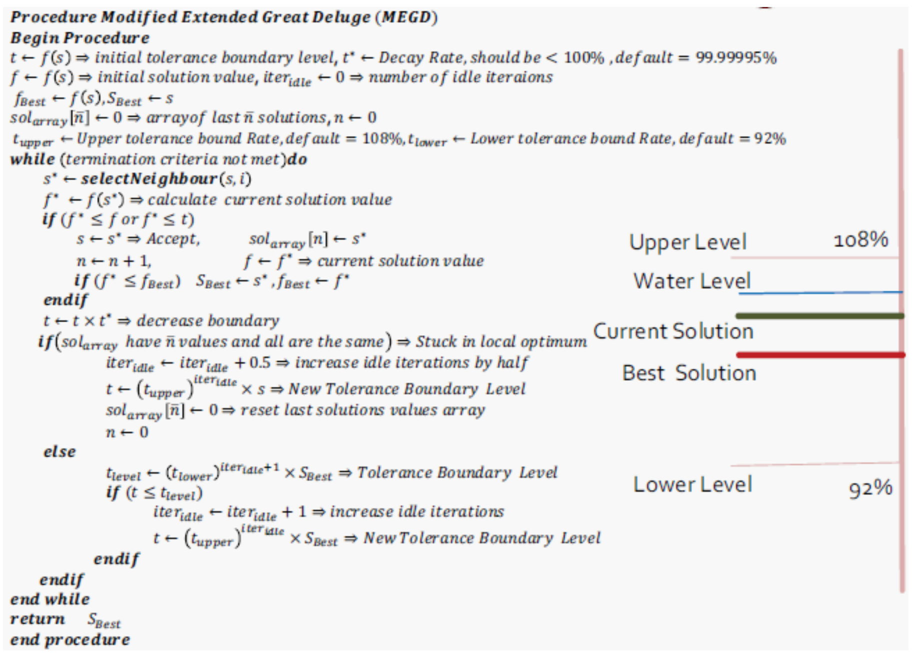
The Extended Great Deluge (EGD) [8] has a construction phase followed by an improvement phase. The construction phase is applied using the existing adaptive ordering heuristic search method [19]. This latter ordering uses a weighted order list of the examinations which is to be scheduled based on soft constraints as well as the ″difficulty to schedule″ constraint. Once an exam is scheduled, its weight is increased based on the localized penalties it came across. The unscheduled examinations are given a considerably larger increase, based on a formulation that is based on the maximum general penalty encountered from [19]. The improvement phase starts when feasibility is achieved in the construction phase and tries to provide an improved solution. Unlike EGD, our approach is only concerned with the enhancement phase and it only tries to improve the overall value of the current feasible complete solution. Our approach is different from EGD as follows.

1. In the original GD, the tolerance value starts with the initial solution's value and decreases by a preconfigured rate. It tries to range within all neighbours of the current solution in each iteration. However, in our approach, tolerance rate ranges between values that are percentage of the current solution value; one above and one below. In our approach, we use two preconfigured values, namely tolerance lower bound and tolerance upper bound. Tolerance upper bound is a preconfigured value that defaults to $(108\%)^{iter_{idle}}$ of the initial solution. $iter_{idle}$ is a counter that starts with 1 and is incremented by 1 each time the tolerance rate is reset. Tolerance lower bound is also a preconfigured value that defaults to 92% of the initial solution. The tolerance decay rate is a predetermined rate that defaults to 99.99995%. At the beginning a tolerance rate t is assigned to a value of the initial solution. It is decreased by tolerance decay rate in each iteration. Likewise, in every iteration, a new neighbour is selected and tested against the current t and the best solution value. If it is better than either one of them, the current solution becomes the best solution and t is decayed by tolerance rate.
2. The second difference occurs at the time of taking the decision to reset the tolerance value t. Tolerance value t is reset as follows. t reaches the tolerance lower bound which as we discussed is equal to 92% (or predetermined value) of the best solution so far. We can as well reset t based on the last n (defaulted to 40) solutions if they happen to be consistent and carry the same value. This means that we are stuck in a local optimum and there is no need to complete the full cycle and reach the lower bound. Rather, we decrease the current tolerance decay rate by half the rate and restart.

Figure 1 presents the pseudo code of MEGD.

Neighbourhood selection variation is by far the most influential technique that affects rapid local search. Using more than one neighbourhood within a search provides a very effective technique of escaping from a local optimum. It is notable that if the current solution is in a local optimum in one neighbourhood, it might escape the local optimum, if assigned a different neighbourhood and can consequently be more improved using a good feasible approach. In exam timetabling, the neighbourhoods used in local search techniques largely involve moving some exams from their current time slot and/or rooms to a new time slot and/or rooms. Based on that, our implementation (corresponding to the function **selectNeighbour()** in Figure 1) uses the following seven neighbourhoods.

1. Exam Duration Move: selects a single exam randomly and move it to a different feasible time slot randomly.



```
Procedure Modified Extended Great Deluge (MEGD)
Begin Procedure
$t \leftarrow f(s) \Rightarrow$ initial tolerance boundary level, $t^* \leftarrow$ Decay Rate, should be $< 100\%$, default $= 99.99995\%$
$f \leftarrow f(s) \Rightarrow$ initial solution value, $iter_{idle} \leftarrow 0 \Rightarrow$ number of idle iteraions
$f_{Best} \leftarrow f(s), S_{Best} \leftarrow s$
$sol_{array}[\bar{n}] \leftarrow 0 \Rightarrow$ arrayof last $\bar{n}$ solutions, $n \leftarrow 0$
$t_{upper} \leftarrow$ Upper tolerance bound Rate, default $= 108\%$, $t_{lower} \leftarrow$ Lower tolerance bound Rate, default $= 92\%$
while (termination criteria not met) do
    $s^* \leftarrow$ selectNeighbour$(s, i)$
    $f^* \leftarrow f(s^*) \Rightarrow$ calculate current solution value
    if ($f^* \leq f$ or $f^* \leq t$)
        $s \leftarrow s^* \Rightarrow$ Accept,          $sol_{array}[n] \leftarrow s^*$
        $n \leftarrow n + 1$,                    $f \leftarrow f^* \Rightarrow$ current solution value
        if ($f^* \leq f_{Best}$)   $S_{Best} \leftarrow s^*, f_{Best} \leftarrow f^*$
    endif
    $t \leftarrow t \times t^* \Rightarrow$ decrease boundary
    if($sol_{array}$ have $\bar{n}$ values and all are the same) $\Rightarrow$ Stuck in local optimum
            $iter_{idle} \leftarrow iter_{idle} + 0.5 \Rightarrow$ increase idle iterations by half
            $t \leftarrow (t_{upper})^{iter_{idle}} \times s \Rightarrow$ New Tolerance Boundary Level
            $sol_{array}[\bar{n}] \leftarrow 0 \Rightarrow$ reset last solutions values array
            $n \leftarrow 0$
    else
            $t_{level} \leftarrow (t_{lower})^{iter_{idle}+1} \times S_{Best} \Rightarrow$ Tolerance Boundary Level
            if ($t \leq t_{level}$)
                $iter_{idle} \leftarrow iter_{idle} + 1 \Rightarrow$ increase idle iterations
                $t \leftarrow (t_{upper})^{iter_{idle}} \times S_{Best} \Rightarrow$ New Tolerance Boundary Level
          endif
    endif
end while
return   $S_{Best}$
end procedure
```

Fig. 1. Procedure Modified Extended Great Deluge (MEGD).

2. Exam Duration Swap Move: selects two exams randomly and swaps their assigned time slots.
3. Non Conflicting Assignment Move: selects an exam randomly and assigns it to a non-conflicting assignment (time slot and room) randomly.
4. Room Move: selects a single exam randomly and moves it to a different feasible room randomly.
5. Room Swap Move: selects two exams randomly and swaps their assigned rooms.
6. Exam Swap Move: selects two exams randomly and swaps their assignments (i.e. time slots and rooms).
7. Random Move: selects an exam randomly and assigns a new assignment to it randomly. The assignment consists of a room and time slot and might cause conflicts.

4 Experimentation

This section reports the experiments conducted on the well-known benchmarking datasets of the International Timetabling Competition (ITC 2007) [13]. This benchmarking datasets consists of 12 basic real world examination time tabling problems obtained from different anonymous universities around the world. The detailed properties of the 12 benchmark instances are summarized in Table 2.

Table 2. ITC 2007 Exam Track Benchmarking Datasets.

Instance #	# of students students	# of exams	# of rooms	Period & Room Hard Constraints	Constraints density	# of time slots
1	7,891	607	7	12	5.04%	54
2	12,743	870	49	14	1.17%	40
3	16,439	934	48	184	2.62%	36
4	5,045	273	1	40	14.94%	21
5	9,253	1,018	3	27	0.87%	42
6	7,909	242	8	23	6.13%	16
7	14,676	1,096	15	28	1.93%	80
8	7,718	598	8	21	4.54%	80
9	655	169	3	10	7.79%	25
10	1,577	214	48	58	4.95%	32
11	16,439	934	40	185	2.62%	26
12	1,653	78	50	16	18.21%	12

As we will see, our proposed approach is successful in competing with benchmarking results published in literature so far. We measure the general behaviour and performance of our implementation in the two different phases to solve the exam timetabling problem; construction phase and enhancement phases. We also compare our approach to the well known exam timetabling problem solvers. All the experiments are performed on an PC Intel Core 2-Duo 2.4 GHz processor with 8 GB of RAM.

4.1 Construction Phase Testing and Analysis

Our construction approach is based on Tabu Search with Dictionary Conflicts. We set our goal to get a complete feasible solution as fast as possible so that the enhancement phase can kick in and improve the overall solution value gradually. In order to measure the performance of Tabu with CD, we tested it against standard Tabu search and in both cases preprocessing phase is done prior to constructing complete solution. As known, standard Tabu algorithm prevents cycling by using a tabu list, which determines the forbidden moves. This list stores the most recently accepted moves. The inverses of the moves in the list are forbidden. Our approach differs in that we sum all the accumulated number of conflicts that a move caused rather than just moves which are considered as forbidden. We also implemented "Iteration Distance" which excludes entries that are far away from the current iteration based on configured setting for iteration distance. For the purpose of the construction phase testing, we selected dataset 4 as it has a high conflict density (14.94%) along with high number of students and exams which makes it as one of the toughest problem to solve in our benchmarking datasets.

During the process of building a complete feasible solution, we record solution value in every iteration along with its time. This testing is only concerned with the construction phase and so we set our testing to run for 10 times for each method of the selected dataset. Then we select the trial with the best solution value from the ten trials. We represent each point in the graph with the corresponding penalty cost monitored after every iteration along with its time. The last penalty cost is the cost of the first complete feasible solution and that is where the construction phases stops.

Figure 2 illustrates the full snapshot of the best trial for standard TS on dataset 1 while figure 3 shows the same pattern for TS with CD. Among 10 trials, using best run's solution value, although standard TS shows better complete solution (6041), it took 8.03 seconds and 1081 iterations to get it while TS with CD with 6803, took 4.21 and 672 iterations. Also, standard TS algorithm shows relatively higher number of fluctuations between lower penalty cost and higher ones where TS with CD seems to have gradually been building the complete solution with a minimum number of instability.

However, dataset 4 has shown a different pattern. Dataset 4 is one of the most constrained problems. The top chart of Figure 4 illustrates the full snapshot of the best trial for standard TS on dataset 4 while the bottom chart shows the same pattern for TS with CD. The top chart articulates how standard TS struggled with finding the less penalty cost solutions in contrary to TS with CD (bottom chart). Standard TS spent a total of 28.54 seconds (3281 iterations) to find a best complete solution, amongst 10 trials, with a penalty cost of 31133 while TS with CD took only 2.03 seconds (567 iterations) to find one with a penalty of 27633. That is a performance improvement of around 1400% with solution value improvement of 112%.

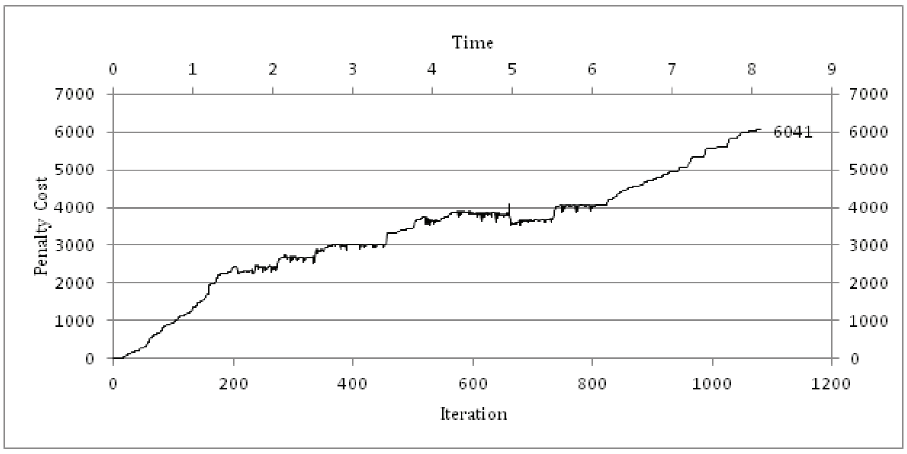


Fig. 2. Dataset 1 Construction Phase using Standard TS.

Dataset 5 is the least constrained problem with only 0.87% but with relatively high number of students and exams (9253 students and 1018 exams)

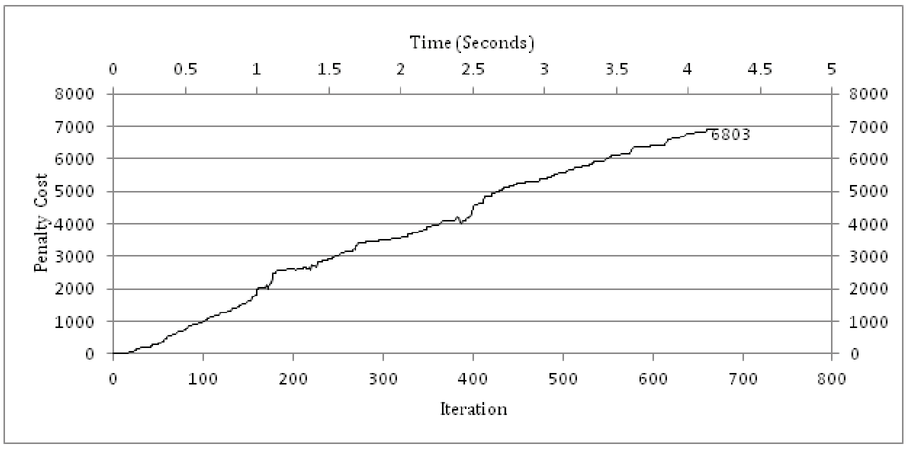


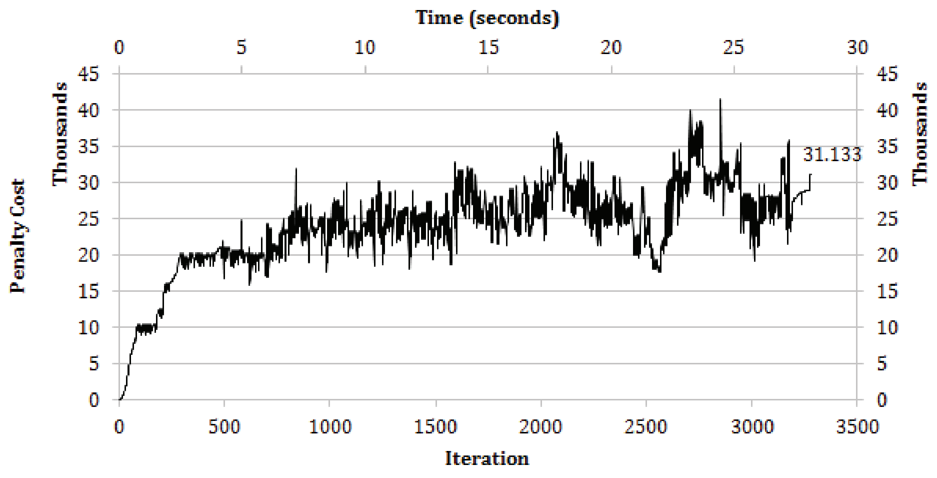
Fig. 3. Dataset 1 Construction Phase using TS with CD.

which leads us to think that it should be one of the easiest problems to solve. We can see that in the lack of any fluctuations between worse and better solutions in the graphs for the datasets in figure 5. Nonetheless, TS with CD algorithm performs slightly better than standard TS even though the problem itself tends to be easy to solve. In ten trials, standard TS obtained 6530, as a best solution value, in 2.58 seconds (1020 iterations) while TS with CD achieved 5030, as a best solution value, in 2.21 seconds (1050 iterations).

4.2 Enhancement Phase Testing and Analysis

We compare 4 methods labeled method 1, method 2, method 3 and method 4 and respectively corresponding to HC+SA, SA, EGD and our MEGD. All these methods use Tabu Search with Dictionary Conflicts in the construction phase. In addition, only methods 2, 3 and 4 have a preprocessing phase.

Figures 6, 7 and 8 shows the enhancement phase best solution distribution history for four methods against iteration in datasets 1, 6 and 8. From these figures, we can clearly notice that without preprocessing the first method tends to improve solutions values within a relatively short time and keeps improving almost very slowly. Another visible notice is that method 1 seems to use less number of iterations which suggests that it employs these iterations cycles either in backtracking or accessing non efficient collections. Method 2 which also uses SA starts enhancing a complete solution very early but then ends with slightly outperforming method 1. On the other hand, the last two methods, using GD flavours with preprocessing in place, spend some time to find the first improving solution after the first complete solution which also tends to be of, relatively, worse value than methods 1 and 2. This is due to the nature of great deluge algorithm which only accepts an improving solution. Also, a bad solution is accepted if its quality is less than (for the case of a minimization problem)



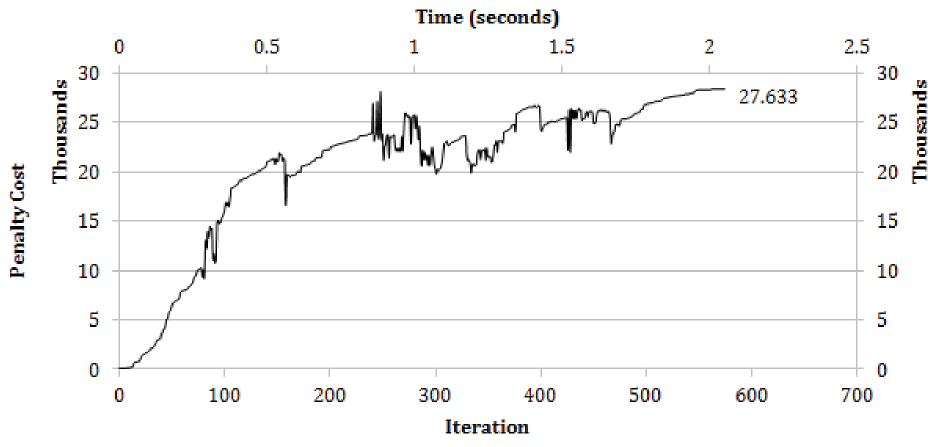
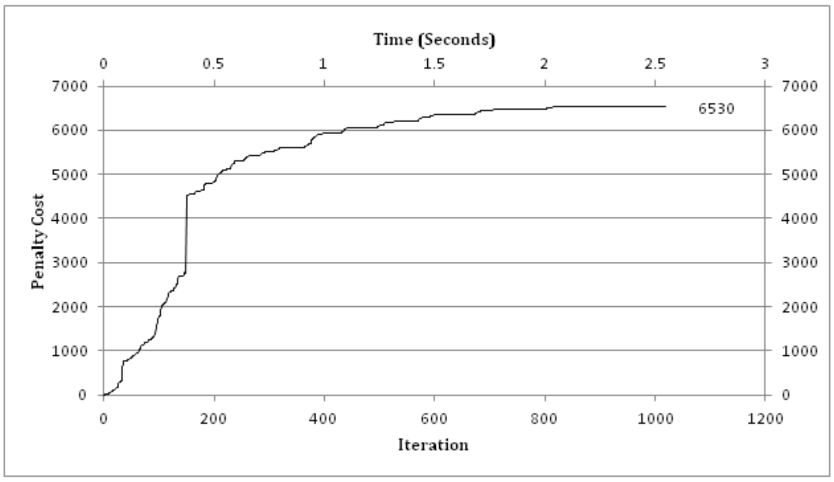


Fig. 4. Performance of the construction phase on dataset 4 with Tabu and Tabu together with conflicts dictionary.



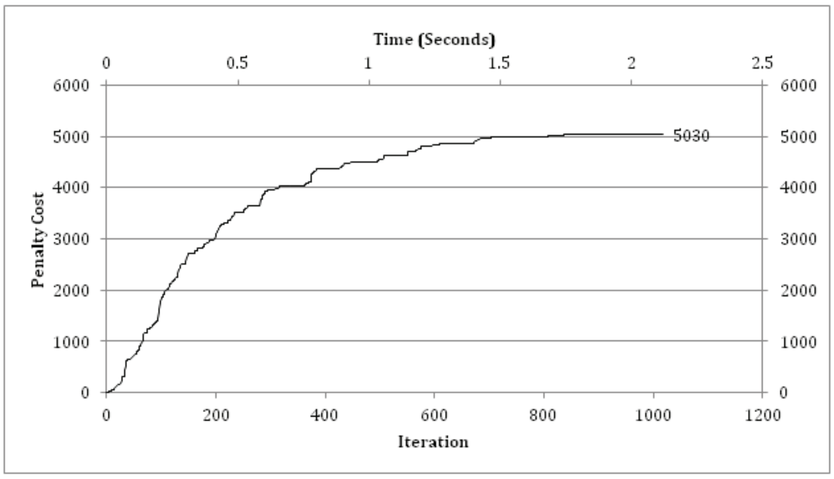


Fig. 5. Dataset 5 using Standard TS (top chart) and TS with CD (bottom chart).

or equal to an upper bound or ″level″ in which during the search process, the ″level″ is iteratively updated by a constant decreasing rate. It also means that, with the preprocessing phase in place, there will be more features. This means that there are more effort to satisfy more constraints but also gaining better performance when looking up the different collections in particular area as well as a more careful exploration. For the inclusion of preprocessing phase, our proposed search algorithm diversification of search to gather the whole search space proved the importance of finding the global minimum quickly. We also note that MEGD performs slightly better than EGD in 8 out of 12 of the datasets. Figure 6 illustrates that methods 3 and 4 were close in terms of results in achieving the best solution. This is also the case for method 1 and 2 although method 2 outperformed to some extent method 1. Method 3 reached a best solution value of 4185. The same pattern also appears in figures 7 and 8 where they show results for dataset 6 and dataset 8 where method 4 is marginally the winner in finding the best solution.

Generally, the algorithms might behave differently due to the different measurements enforced during the search process. However, the difference between SA, GD, EGD and MEGD algorithms lies in the acceptance criteria functionality that would make a difference on the limited solving time that was imposed on our benchmarking datasets. This might not be the case if we have relatively longer times for several hours or days as all these algorithms are based on the stochastic local search and there will always be the possibility of achieving good results.

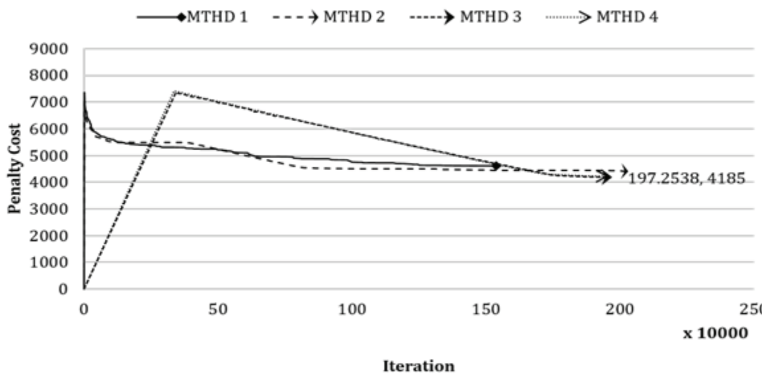


Fig. 6. Performance on dataset 1 of the enhancement phase.

4.3 Comparative Tests Results

On the basis of results obtained by both construction and enhancement phases, we decided to compare our four methods to the five well known ET solvers. Each of the datasets used in our testing phase has a previously discovered best known solution announced by ITC 2007. The five known solvers are the following finalists of the examination track of the competition.

1. Müller [20] implemented a constraint-based solver, which constructs a complete solution, followed by a hill climbing and a great deluge approach for improving the solution.
2. Gogos et al. [21] used a Greedy Randomized Adaptive Search Procedure, followed by simulated annealing, and integer programming.
3. Atsuta et al. [22] developed a constraint satisfaction solver combined with Tabu search and iterated local search.

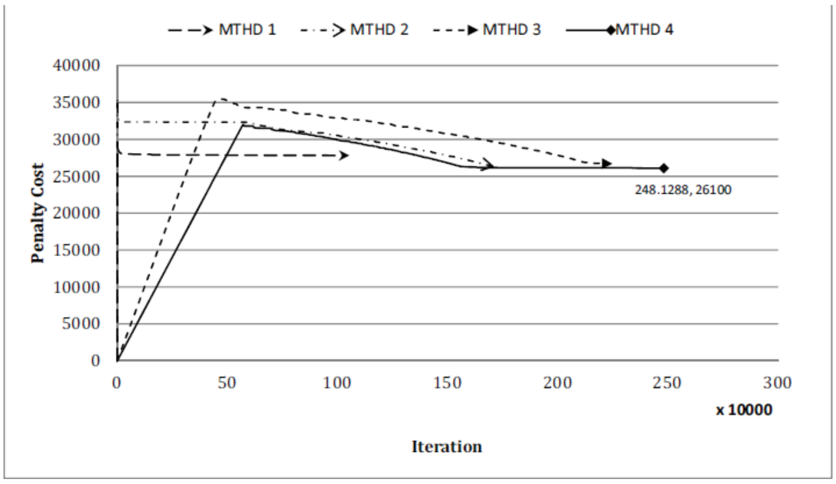


Fig. 7. Performance on dataset 6 of the enhancement phase.

4. De Smet [23] has his own solver, namely Drools Solver, which is a combination of Tabu search and the JBoss Drools rules engine used for the evaluation.
5. Finally, Pillay [24] used a heuristic method that is inspired by cell biology.

During experiment runs, we managed to achieve an outstanding 98% success in reaching complete feasible solution on all instances in all attempts. The remaining 2% were only in dataset 4 and 12.

For each method trials we performed 11 individual runs on each of the 12 competition instances, using the time limit specified by the competition benchmarking program as our stopping criteria, which equated to 362 seconds. The same timeout value on each machine is used for all of the 12 datasets. In all cases, we logged out all best solution values history along with times and iterations where these best solution values are discovered.

The settings of the algorithms have remained the same throughout the experiment for the purpose of going in line with ITC 2007 rules. One of our objectives in testing phase is to represent different algorithm variations that are composed of different algorithms and compare them to the performance of ITC 2007 results. The expectation was also set for the results to be reasonably comparable if not better than ITC 2007 exam track results.

Table 3 reports the comparative results including the best solution value (lowest penalty cost) and average of best solution values for each variant. When searching without preprocessing, performance degrades relatively to when using preprocessing phase. Only the first method did not use preprocessing and if we look first at the performance of its algorithms in comparison to ITC 2007

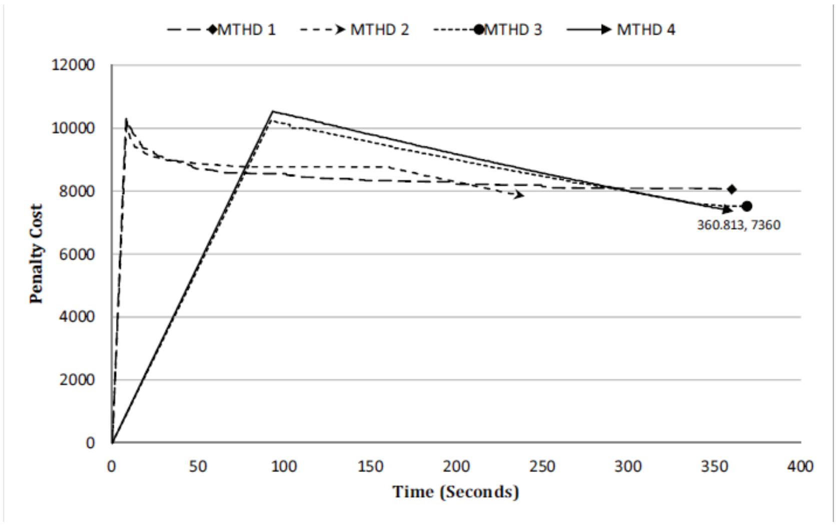
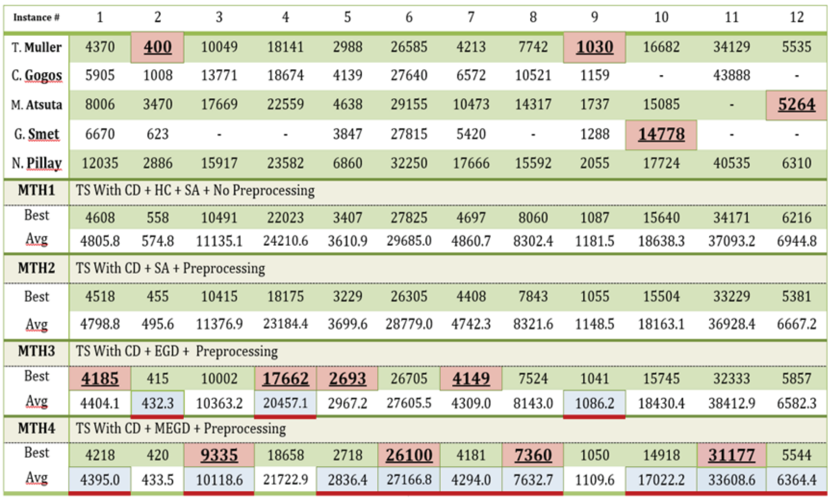


Fig. 8. Performance on dataset 8 of the enhancement phase.

results we will notice that it comes in the second place in 10 out of 12. This is the case for all datasets except datasets 10 and 12. TS with CD + HC+ SA with no preprocessing is the worst algorithm variant in our testing and it comes in the second place in most datasets in comparison to ITC 2007 results.

The other three algorithms variants performed better. Only when we used GD algorithm extensions (EGD and MEGD), we started to see results that overtake ITC 2007 results. Our approach gets 8 out of 12 datasets as best results. These results are split between EGD and MEGD evenly with 4 best results each.

In order to obtain a fair comparison, it is worth noticing that the performance loss is on average about 7% between SA with preprocessing and MEGD with preprocessing, whereas it is about 10% if the preprocessing phase is not implemented with SA. Moreover, one can also notice that the gap between the two methods becomes smaller with higher conflicts density problems, and that the behavior of the methods with pre-processing phase implemented is more stable with respect to SA with no preprocessing phase. All in all, EGD and MEGD performed much better than SA with preprocessing phase not to mention SA with no preprocessing. Finally, all of our methods performed well in comparison to ITC 2007 results in best solution values and in best average values.



Instance #	1	2	3	4	5	6	7	8	9	10	11	12
T. Muller	4370	**<u>400</u>**	10049	18141	2988	26585	4213	7742	**<u>1030</u>**	16682	34129	5535
C. **Gogos**	5905	1008	13771	18674	4139	27640	6572	10521	1159	-	43888	-
M. **Atsuta**	8006	3470	17669	22559	4638	29155	10473	14317	1737	15085	-	**<u>5264</u>**
G. **Smet**	6670	623	-	-	3847	27815	5420	-	1288	**<u>14778</u>**	-	-
N. **Pillay**	12035	2886	15917	23582	6860	32250	17666	15592	2055	17724	40535	6310
MTH1	TS With CD + HC + SA + No Preprocessing											
Best	4608	558	10491	22023	3407	27825	4697	8060	1087	15640	34171	6216
Avg	4805.8	574.8	11135.1	24210.6	3610.9	29685.0	4860.7	8302.4	1181.5	18638.3	37093.2	6944.8
MTH2	TS With CD + SA + Preprocessing											
Best	4518	455	10415	18175	3229	26305	4408	7843	1055	15504	33229	5381
Avg	4798.8	495.6	11376.9	23184.4	3699.6	28779.0	4742.3	8321.6	1148.5	18163.1	36928.4	6667.2
MTH3	TS With CD + EGD + Preprocessing											
Best	**<u>4185</u>**	415	10002	**<u>17662</u>**	**<u>2693</u>**	26705	**<u>4149</u>**	7524	1041	15745	32333	5857
Avg	4404.1	432.3	10363.2	20457.1	2967.2	27605.5	4309.0	8143.0	1086.2	18430.4	38412.9	6582.3
MTH4	TS With CD + MEGD + Preprocessing											
Best	4218	420	**<u>9335</u>**	18658	2718	**<u>26100</u>**	4181	**<u>7360</u>**	1050	14918	**<u>31177</u>**	5544
Avg	4395.0	433.5	10118.6	21722.9	2836.4	27166.8	4294.0	7632.7	1109.6	17022.2	33608.6	6364.4

Table 3. Comparative results.

5 Conclusion and Future Work

We presented our proposed approach to solve exam timetabling problem using four different metaheuristics search methods; tabu search, hill climbing, simulated annealing, and different flavours of great deluge. We also introduced a pre-processing phase to enhance the overall search process. A Tabu metaheuristic search method with conflict dictionary is proposed as a construction phase to achieve a partial or complete initial feasible solution. The tabu list does not contain operators or moves that are problem specific. It only needs to store the conflicted moves along with the accumulated number of conflicts it caused.

A modified extended great deluge heuristic search method is used during search to eliminate some of the time wasted in local optimum based on certain conditions. The selected heuristics perform in sequence to produce a good solution for the current state of the problem. The whole hybrid heuristics approach is configurable and able to manage and control its heuristics without having a domain pre-knowledge of the exam timetabling problem.

In the near future we will investigate advanced variables ordering heuristics [9] as weill as evolutionary techniques using a parallel architecture [25, 26]. We will also intend to explore path consistency algorithms [27] to discover temporal constraints in the preprocessing phase.

References

1. Qu, R., Burke, E., McCollum, B., Merlot, L., Lee, S.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling **12**(1) (2009) 55–90
2. Schaerf, A.: A survey of automated timetabling. Artificial Intelligence Review **13**(2) (2009) 86–127
3. McCollum, B., McMullan, P., Parkes, A., Burke, E., Abdullah, S.: An extended great deluge approach to the examination timetabling problem. MISTA 2009, Multidisciplinary International Scheduling Conference: Theory and Applications (2009) 20–69
4. Gogos, C., Alefragis, P., Housos, E.: An improved multi-staged algorithmic process for the solution of the examination timetabling problem. Annals of Operations Research **194**(1) (2012) 203–221
5. Abdullah, S., Turabieh, H.: On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems. Information Sciences **191**(0) (2012) 146 – 168 Data Mining for Software Trustworthiness.
6. Sabar, N., Ayob, M., Qu, R., Kendall, G.: A graph coloring constructive hyper-heuristic for examination timetabling problems. Applied Intelligence **37**(1) (2012) 1–11
7. Sabar, N., Ayob, M., Kendall, G., Qu, R.: Grammatical evolution hyper-heuristic for combinatorial optimization problems. Evolutionary Computation, IEEE Transactions on **17**(6) (Dec 2013) 840–861
8. McCollum, B., McMullan, P., Parkes, A.J., Burke, E., Abdullah, S.: An extended great deluge approach to the examination timetabling problem. in MISTA 2009, Multidisciplinary International Scheduling Conference: Theory and Applications, Dublin (2010) 20–69
9. Mouhoub, M., Jashmi, B.J.: Heuristic techniques for variable and value ordering in csps. [29] 457–464
10. Kendall, G., Hussin, N.M.: A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. Practice and Theory of Automated Timetabling **3616** (2005) 270–293
11. L. T. G. Merlot, N. Boland, B.D.H., Stuckey, P.J.: A hybrid algorithm for the examination timetabling problem. Practice and Theory of Automated Timetabling **2740** (2003) 207–231
12. Dowsland, K.A.: Simulate annealing. Modern heuristics techniques for combinatorial problems, ch. 2 (1995) 20–69
13. McCollum, B., McMullan, P.: The second international timetabling competition: Examination timetabling track. University of Nottingham, Queens University, Notingham, Belfast, Technical Report: QUB/IEEE/Tech/ITC2007/Exam/v4.0/17 2007 (2009)
14. E.K. Burke, J.N., Weare, R.: A memetic algorithm for university exam timetabling. Practice and Theory of Automated Timetabling **1153** (1996) 241–250
15. Chan, C.K., Gooi, H.B., Lim, M.H.: Co-evolutionary algorithm approach to a university timetable system. in The 2002 congress on evolutionary computation, Honolulu **2** (2002) 19461951
16. E. K Burke, B. MacCathy, S.P., Qu: Knowledge discovery in a hyperheuristic for course timetabling using case-based reasoning. Practice and theory of automated timetabling (PATAT'02) (2002)
17. M. W. Carter, G.L., Lee, S.Y.: Examination timetabling: algorithmic strategies and applications. Journal of the Operational Research Society **47** (1996) 373–383

18. Dueck, G.: New optimisation heuristics for the great deluge algorithm and the record-torecord travel. Journal of Computational Physics **104** (1993) 86–92
19. Burke, E.K., Newall, J.P.: Solving examination timetabling problems through adaption of heuristic orderings. Annals of Operations Research **129**(1-4) (2004) 107–134
20. Müller, T.: Itc2007 solver description: A hybrid approach. in Proceedings of the 7th international conference on the practice and theory of automated timetabling (2008)
21. E. Gogos, C.A., Housos, P.: Multi-staged algorithmic process for the solution of the examination timetabling problem. Practice and theory of automated timetabling (PATAT), 2008 (2008)
22. T. Atsuta, M.N., Ibaraki: An approach using a general csp solver. Technical report (2008)
23. Smet, G.D.: ITC2007 examination track: Practice and theory of automated timetabling. Technical report (2008)
24. Pillay, N.: A developmental approach to the examination timetabling problem. Technical report (2008)
25. Abbasian, R., Mouhoub, M.: An efficient hierarchical parallel genetic algorithm for graph coloring problem. [29] 521–528
26. Abbasian, R., Mouhoub, M.: A hierarchical parallel genetic approach for the graph coloring problem. Applied Intelligence **39**(3) (2013) 510–528
27. Mouhoub, M.: Analysis of Approximation Algorithms for Maximal Temporal Constraint Satisfaction Problems. In: The 2001 International Conference on Artificial Intelligence (IC-AI'2001), Las Vegas (2001) 165–171
28. Mouhoub, M.: Dynamic Path Consistency for Interval-based Temporal Reasoning. In: 21st International Conference on Artificial Intelligence and Applications(AIA '2003), ACTA Press (2003) 393–398
29. Krasnogor, N., Lanzi, P.L., eds.: 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011. In Krasnogor, N., Lanzi, P.L., eds.: GECCO, ACM (2011)