International Journal of Computational Intelligence and Applications
Vol. 15, No. 4 (2016) 1650023 (22 pages)

DOI: 10.1142/S1469026816500231

World Scientific
www.worldscientific.com

A Multi-Phase Hybrid Metaheuristics Approach for the Exam Timetabling

Ali Hmer and Malek Mouhoub*

Department of Computer Science
University of Regina
Regina S4S 0A2, Canada
**Malek.Mouhoub@uregina.ca*

Received 16 February 2016
Revised 25 October 2016
Accepted 2 November 2016
Published 29 November 2016

We propose a Multi-Phase Hybrid Metaheuristics approach for solving the Exam Timetabling Problem (ETP). This approach is defined with three phases: pre-processing phase, construction phase and enhancement phase. The pre-processing phase relies on our variable ordering heuristic as well as a form of transitive closure for discovering implicit constraints. 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 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 Problem (ETP),[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. In this regard, many ETP solving approaches have been proposed and discussed[1–13] 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 consisting of the following three stages: preprocessing, construction, and enhancement. 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[14] and implicit constraints are discovered using a form of transitive

*Corresponding author.

closure based on our Dynamic Path Consistency (DPC) algorithm for temporal constraints.[15–17] 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 is used. The following metaheuristics are considered: Hill Climbing (HC),[18,19] Simulated Annealing (SA)[20] and our updated version of the extended "Great Deluge" solving algorithm.[3] This latter is an improvement of the one proposed in Ref. 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.[21] The results are very promising and are competitive with the well-known ETP solvers.

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 Sec. 4. Finally, concluding remarks and future works are listed in Sec. 5.

2. Problem Description

We model the ETP as a constraint optimization problem (COP) including the variables, hard and soft constraints listed in the following two subsections. Solving this problem consists in finding a complete assignment of values to all the variables satisfying all the hard constraints and minimizing the violations of the soft ones. In other words, this corresponds to finding a schedule that would be fair to all the students. Minimizing soft constraints is done by minimizing a penalty or cost function defined as shown in Sec. 2.2.

2.1. *Variables and constraints*

Following the common formulations to the Examination Timetabling,[22,23] variables and constraints are defined as follows.

- **Variable.** Each exam is modeled as a problem variable defined over a finite 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 temporal constraint prevents a student from being scheduled for more than one exam during a given time period.
- **Precedence Constraint.** This temporal constraint imposes an ordering (precedence) between two or more exams.
- **Same Time Constraint.** This temporal constraint restricts two or more exams to take place during the same time slot. This is the case of exams containing similar material.

- **Different Time Constraint.** This temporal constraint restricts two or more exams to take place during different time slots.
- **Same Room Constraint.** This constraint restricts two or more exams to take place in the same room.
- **Different Room Constraint.** This constraint restricts two or more exams to take place in different rooms.

2.2. *Soft constraints and penalty functions*

The penalty function is a measure 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 corresponding ETP. Penalties correspond to violating soft constraints including the following.

(1) Students taking two exams in a row.
(2) Students taking 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.

The goal of the above soft constraints is to maximize students' satisfaction (case of the first two soft constraints), to reduce University resources and cost (case of soft constraints 3 and 4) or both (case of the last soft constraint).

3. Proposed ETP Solving Approach

As described in the introduction section, our proposed solving approach consists of the following three main phases. A pre-processing phase followed by a construction and an enhancement phases. The following describes the details of each stage.

3.1. *Pre-processing phase*

The pre-processing phase consists of two stages described as follows.

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 Refs. 24 and 25, 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 students, large exams, 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. In addition and following the idea of most constrained variables first based on conflict driven heuristics for weighting constraints,[14] exams with most scheduling difficulty are scheduled first. The goal here is to prevent later failure earlier which will decrease the size of the search space. Conflict driven heuristics are those that gather information about constraint violations during the search process, in the form of constraint weights. The heuristics we use for this purpose are respectively based on HC and Ant Colony Optimization (ACO) techniques.[14] More precisely, these two approximation techniques are run for a specific amount of time or cycles, during which, the constraints gain weight every time they are violated. At the end of this process, each variable gets a weighted degree, which is the sum of the weights of the constraints that the variable is involved in. Variables are then sorted based on their weights and those with larger weight get more priority in the ordering. More details about this process can be found in Ref. 14.

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 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 pre-processing stage starts by creating a temporal constraint graph where nodes represent the exams and edges are the hard temporal constraints between exams. We then apply our DPC algorithm[16,17] to discover new temporal constraints between other exams in the same graph.

Figure 1 lists the pseudo-code of DPC[17] we used for discovering new temporal constraints. This algorithm is based of Allen's Algebra for representing qualitative temporal information.[15] In this representation, each temporal constraint is expressed as a disjunction of Allen primitives (possible relations between a pair of temporal intervals). Figure 2 lists all the possible Allen primitives. For instance the following represents the fact that $Exam_1$ and $Exam_2$ should be scheduled at different times (mutually exclusive events): $Exam_1\ B \vee Bi\ Exam_2$. $Exam_1\ Bi\ Exam_2$ corresponds to $Exam_2\ B\ Exam_1$ and the same applies for all the other inverse primitives. For the sake of notation simplicity, a temporal constraint is denoted as a set of Allen primitives rather than a disjunction of these basic relations. For instance, the above example will be represented as $Exam_1\ \{B, Bi\}\ Exam_2$. We will adopt this notation in the remaining of the paper.

Function $Restrict(i, j, new\_constraint)$
C_{ij}: current constraint (disjunction of Allen's primitives) between events i and j

1. $t \leftarrow new\_constraint \bigcap C_{ij}$
2. $updated\_list \leftarrow \{(i, j)\}$
3. **if** $(t = \emptyset)$ **then**
4. return "Constraint cannot be added"
5. **else**
6. $C_{ij} \leftarrow t$
7. **if** $\neg DPC(updated\_list)$ **then**
8. return "Constraint cannot be added"

Function $DPC(updated\_list)$
C_{xk}: current constraint (disjunction of Allen's primitives) between events x and k
$INVERSE(R)$: returns the disjunction of the inverse of each Allen primitive within R
$\bigotimes$: composition operator between two relations using Allen's composition table

1. $L \leftarrow updated\_list$
2. **while** $(L \neq \emptyset)$ **do**
3. select and delete an (x, y) from L
4. **for** $k \leftarrow 1$ to n, $k \neq x$ and $k \neq y$ **do**
5. $t \leftarrow C_{xk} \bigcap C_{xy} \bigotimes C_{yk}$
6. **if** $(t \neq C_{xk})$ **then**
7. **if** $(t = \emptyset)$ **then** return false
8. $C_{xk} \leftarrow t$
9. $C_{kx} \leftarrow INVERSE(t)$
10. $L \leftarrow L \cup \{(x, k)\}$
11. $updated\_list \leftarrow updated\_list \cup \{(x, k)\}$
12. $t \leftarrow C_{ky} \bigcap C_{kx} \bigotimes C_{xy}$
13. **if** $(t \neq C_{ky})$ **then**
14. **if** $(t = \emptyset)$ **then** return false
15. $C_{yk} \leftarrow INVERSE(t)$
16. $L \leftarrow L \cup \{(k, y)\}$
17. $updated\_list \leftarrow updated\_list \cup \{(y, k)\}$
18. return true

Fig. 1. Dynamic path consistency algorithm.

Our DPC algorithm has the ability to process temporal constraints in an incremental way. In this regard, each new constraint (expressed as a disjunction of some Allen primitives) between two events (exams) i and j is first processed by the $Restrict(i, j)$ function. This latter function will compute the intersection between this new constraint with the current one (if any). This will update the relation between the two events (this can be the case where the user is submitting a more restrictive constraint) or rejects the new constraint (if it conflicts with the current one). Note that the initial constraints between each pair of events are set to the universal relation (disjunction of all the Allen primitives) which corresponds to completely unknown relations. The DPC algorithm is then applied on the list of new constraints in order to check the consistency of these latter and deduce new temporal constraints. This is done by enforcing the path consistency (equivalent to 3 consistency) on each subset of

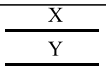
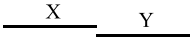
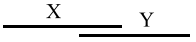
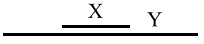
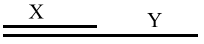
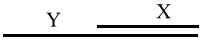
Relation	Symbol	Inverse	Meaning
X before Y	B	Bi	
X equals Y	E	E	
X meets Y	M	Mi	
X overlaps Y	O	Oi	
X during Y	D	Di	
X starts Y	S	Si	
X finishes Y	F	Fi	

Fig. 2. Allen primitives.

3 events, using both the composition and the intersection operations. The composition is computed according to Table 1. The $INVERSE(t)$, invoked in DPC procedure, returns the disjunction of the inverse of each Allen primitive within t. For instance, if $t = \{M, Oi, D\}$ then $INVERSE(t) = \{Mi, O, Di\}$.

Table 1. Allen's composition table.

	E	B	Bi	D	Di	O	Oi	M	Mi	S	Si	F	Fi
E	E	B	Bi	D	Di	O	Oi	M	m	S	s	F	Fi
B	B	B	I	u	B	B	u	B	u	B	B	u	B
Bi	p	I	Bi	vi	Bi	vi	Bi	vi	Bi	vi	Bi	Bi	Bi
D	D	P	Bi	D	I	u	vi	B	Bi	D	vi	D	u
Di	Di	v	ui	n	Di	zi	yi	zi	yi	zi	Di	yi	Di
O	O	B	ui	y	v	x	n	B	yi	O	zi	y	x
Oi	Oi	v	Bi	z	ui	n	xi	zi	Bi	z	xi	Oi	yi
M	M	B	ui	y	B	B	y	B	a	M	M	y	B
Mi	Mi	v	Bi	z	Bi	z	Bi	b	Bi	z	Bi	Mi	Mi
S	S	P	Bi	D	v	x	z	P	Mi	S	b	D	x
Si	s	v	Bi	z	Di	zi	Oi	zi	m	b	s	Oi	Di
F	F	P	Bi	D	ui	y	xi	M	Bi	D	xi	F	a
Fi	Fi	P	ui	y	Di	O	yi	M	yi	O	Di	a	Fi

$x = \{B, O, M\}$
$y = \{D, O, S\}$
$z = \{D, Oi, F\}$
$a = \{E, F, Fi\}$
$b = \{E, S, Si\}$
$u = \{B, O, M, D, S\}$
$v = \{B, O, M, Di, Fi\}$
$n = \{E, F, D, O, S, Fi, Di, Oi, Si\}$

Let us see how we can discover a new constraint using DPC. Assume we have 3 exams; $Exam_1$, $Exam_2$, and $Exam_3$, sharing the following two temporal constraints:

$C_{12} = Exam_1\ \{B\}\ Exam_2$
$C_{23} = Exam_2\ \{Di\}\ Exam_3$

The first constraint above states that $Exam_1$ should happen before $Exam_2$ while the second constraint expresses the fact that $Exam_2$ contains $Exam_3$ ($Exam_3$ happens during $Exam_2$).

In order to enforce path consistency on the above three events (and discover the new constraint between $Exam_1$ and $Exam_3$), we first have to set the temporal constraint between $Exam_1$ and $Exam_3$ to the universal relation (disjunction of the 13 Allen primitives) as this constraint is initially unknown:

$$C_{13} = Exam_1\{B, Bi, D, Di, S, Si, F, Fi, O, Oi, M, Mi, E\}Exam_3$$

DPC will then enforce path consistency as follows:

$$C_{13} \leftarrow C_{13} \bigcap C_{12} \otimes C_{23}$$

According to the composition table (see Table 1), $C_{12} \otimes C_{23}$ will return $\{B\}$. C_{13} will then be set to $\{B\}$ (the intersection takes the primitives that are common to both constraints).

We refer the reader to Refs. 17 and 26 for more details on DPC and temporal constraints.

3.2. *Construction phase*

In the construction phase a complete feasible solution is found using Tabu Search metaheuristics. Tabu Search iteratively moves from one potential solution to an improved one in the neighborhood of the current solution until the stopping criterion has been satisfied. The search is stopped after either a complete feasible solution is found or maximum time is reached. The overall approach is to avoid cycles by preventing or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited and that is why it is called "Tabu".

Our Tabu Search is used along with conflicts dictionary to reduce cycling. A Conflicts dictionary essentially is a dictionary data structure consisting of a key and a 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. 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.

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.

Note that the main difference between the traditional Tabu algorithm and our method is that in the former redundant moves are rejected in order to avoid cycling. In our method however we keep these redundant moves that will help us for our variable assignment decisions. More precisely, 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. More precisely, we applied an iteration distance mechanism that records at which iteration an assignment move along with its number of conflicts occurred. Then, during later search, if the variable is selected again for assignment, the stored information in conflicts dictionary (accumulated potential conflicts for each move) helps guiding the decision on which value should be assigned to the variable. In other words, all moves that involve this variable will be retrieved from Conflicts Dictionary (CD) and a min-conflict value selection heuristics is applied, which selects the entry with the least number of accumulated conflicts and the dictionary entry key value is assigned to that variable. We do not however keep all the past moves but only those that do not go beyond a given iteration number determined by the Iteration distance.

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 HC,[18,19] SA[20] 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) solving method[3] which in turn is based on the original Great Deluge (GD). GD was introduced by Dueck[27] 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.

EGD has a construction phase followed by an improvement phase. The construction phase is applied using the existing adaptive ordering heuristic search method.[28] 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" constraints. 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 Ref. 28. 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 neighbors 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 $(108\%)^{\text{iter}_{\text{idle}}}$ of the initial solution. $\text{iter}_{\text{idle}}$ 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 neighbor 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 3 presents the pseudo code of our MEGD. Neighborhood selection variation is by far the most influential technique that affects rapid local search. Using more than

Procedure Modified Extended Great Deluge (MEGD)
Input: $s \leftarrow$ initial feasible solution
termination criteria: # of maximum iterations
$f(s)$: Cost function
$t^* \leftarrow$ Decay Rate (should be $< 100\%$, default=99.99995%)
$t_{upper} \leftarrow$ Upper tolerance bound Rate (default=108%)
$t_{lower} \leftarrow$ Lower tolerance bound Rate (default=92%)
$\overline{n} \leftarrow$ number of last solutions values (default=40)
Output: an enhanced complete feasible solution
Begin
$t \leftarrow f(s)$ (initial tolerance boundary level)
$f \leftarrow f(s)$ (initial solution value)
$iter_{idle} \leftarrow 1$ (number of idle iterations)
$sol_{array}[\overline{n}] \leftarrow 0$ (array of last $\overline{n}$ solutions)
$n \leftarrow 0$
while (termination criteria not met) **do**
$s^* \leftarrow selectNeighbour(s)$
$f^* \leftarrow f(s^*)$ (calculate current solution value)
if $(f^* \leq f \ or \ f^* \leq t)$
$s \leftarrow s^*$
$sol_{array}[n] \leftarrow s^*$
$n \leftarrow n + 1$
$f \leftarrow f^*$ (current solution value)
endif
$t \leftarrow t \times t^*$ (decrease boundary)
if sol_{array} has $\overline{n}$ values and all are the same (local optimum)
$iter_{idle} \leftarrow iter_{idle} + 0.5$
$t \leftarrow (t_{upper})^{iter_{idle}} \times s$(new Tolerance Boundary Level)
$sol_{array}[\overline{n}] \leftarrow 0$ (reset last solutions values array)
$n \leftarrow 0$
else
$t_{level} \leftarrow (t_{lower})^{iter_{idle}+1} \times s$ (Tolerance Boundary Level)
if $(t \leq t_{level})$
$iter_{idle} \leftarrow iter_{idle} + 1$
$t \leftarrow (t_{upper})^{iter_{idle}} \times s$ (New Tolerance Boundary Level)
endif
endif
end while
return s
end

Fig. 3. MEGD algorithm.

one neighborhood 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 neighborhood, it might escape the local optimum, if assigned a different neighborhood and can consequently be more improved using a good feasible approach. In exam timetabling, the neighborhoods 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 **select-Neighbour()** in Fig. 3) uses the seven neighborhoods listed in Fig. 4.

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

Fig. 4. List of the seven neighborhoods.

4. Experimentation

This section reports the experiments conducted on the well-known timetabling benchmarking datasets of the International Timetabling Competition (ITC 2007).[21,29] This benchmarking datasets consists of 12 basic real world examination timetabling problems obtained from different anonymous universities around the world.

The detailed properties of the 12 benchmark instances are summarized in Table 2. The constraint (or conflict) density value is calculated using the following formula taken from Ref. 30:

$$\text{Constraint density} = \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (\frac{\beta_{ij}}{N})}{N}.$$

Table 2. ITC 2007 exam track benchmarking datasets.

Inst #	# of Students	# of Exams	# of Rooms	Hard Cons	Cons Density (%)	# of Time Slots
1	7891	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	5045	273	1	40	14.94	21
5	9253	1018	3	27	0.87	42
6	7909	242	8	23	6.13	16
7	14,676	1096	15	28	1.93	80
8	7718	598	8	21	4.54	80
9	655	169	3	10	7.79	25
10	1577	214	48	58	4.95	32
11	16,439	934	40	185	2.62	26
12	1653	78	50	16	18.21	12

Table 3. ITC 2007 exam timetabling data sets soft constraints violation penalties.

Instance #	Two in a Row	Two in a Day	Period Spread	Mixed Durations	Front Load
1	7	5	5	10	100, 30, 05
2	15	5	1	25	250, 30, 05
3	15	10	4	20	200, 20, 10
4	9	5	2	10	050, 10, 05
5	40	15	5	0	250, 30, 10
6	20	5	20	25	025, 30, 15
7	25	5	10	15	025, 30, 10
8	150	0	15	25	250, 30, 05
9	25	10	5	25	100, 10, 05
10	50	0	20	25	100, 10, 05
11	10	50	4	35	400, 20, 10
12	35	10	5	5	025, 05, 10

Table 3 shows the different penalties for violating soft constraints for the 12 datasets. These penalties are defined as follows.

- **Two Exams in a Row.** This penalty applies to exam assignments occurrences where two examinations are taken by a student, one straight after another, also known as back to back exams. It is calculated by totaling the number of students that violate the constraint and multiplied by the number provided in the two in a row weighting settings.
- **Two Exams in a Day.** This penalty applies to exam assignments occurrences where two examinations are taken by students in the same day but are not directly back to back. This is obviously conditioned by the fact that there must be three periods or more in a day. The total number is consequently multiplied by the two in a day weighting settings.
- **Period spread (of examinations).** The number of times when students have to sit more than one exam in a time period specified by the institution. This is usually used as an indication of fairness principle to all students taking exams.
- **Mixed duration (of examinations within individual periods).** The number of occurrences of exams timetabled in rooms along with other exams of differing time duration.
- **Front Load.** Most institutions desire that examinations with the largest numbers of students are scheduled at the beginning of the examination session so that markers would be under no stress and would take their time in marking exams. The penalty for this concept, in ITC 2007, is called Front Load and is defined as a sequence of three parameters n, m, t. The idea behind this penalty is to allow the institution to try to schedule larger exams earlier in the examination session. The first parameter defines how the largest exam is defined in terms of the number of students. If the number is, for example, 200 then any exam that has enrolled students of 200 or more is considered to be one of the largest exams. The second parameter is the number of last periods that these larger exams should be avoided

to be scheduled in. The third parameter is the penalty or weighting that should be added each time the constraint is violated. For example, if Front Load = (200, 20, 15), it means that any exam with 200 students or more that is scheduled in the last 20 periods will be penalized with 15.

As we will see, our proposed approach is successful in competing with benchmarking results published in literature so far. We measure the general behavior 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 (finalists of the examination track). All the experiments are performed on an PC Intel Core 2-Duo 2.4 GHz processor with 8 GB of RAM.

4.1. *Discovering constraints in the pre-processing phase*

Discovering constraints that were not stated in problem description would be beneficial for variables that share one or more constraints as it will reduce the size of the search space which will improve the search process. Table 4 shows the number of unspecified constraints, per instance, revealed during this stage. Dataset 3 and 11 have the most unspecified constraints while dataset 12 has no undiscovered constraints.

4.2. *Construction phase testing and analysis*

Our construction approach is based on Tabu Search with CD. 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. For the purpose of the construction phase testing, we selected dataset 4 as it has a high

Table 4. Unspecified constraints discovered during the pre-processing phase.

Instance #	Number of Constraints Discovered
Instance 1	2
Instance 2	6
Instance 3	19
Instance 4	0
Instance 5	11
Instance 6	16
Instance 7	6
Instance 8	3
Instance 9	2
Instance 10	13
Instance 11	19
Instance 12	0

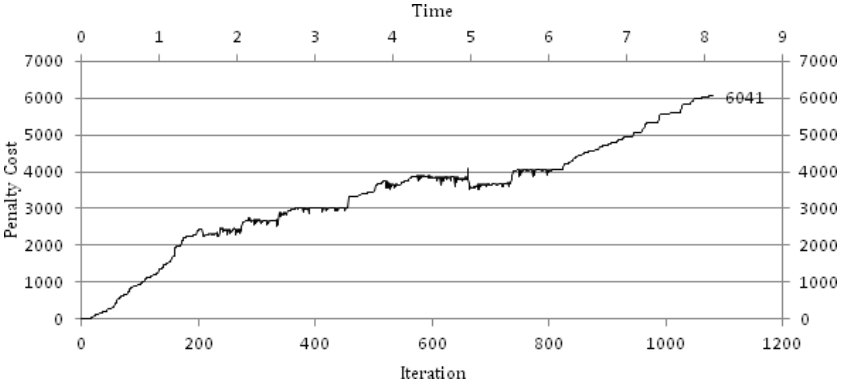


Fig. 5. Dataset 1 construction phase using standard TS.

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 10 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 5 illustrates the full snapshot of the best trial for standard Tabu Search (TS) on dataset 1 while Fig. 6 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 s 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 less fluctuations.

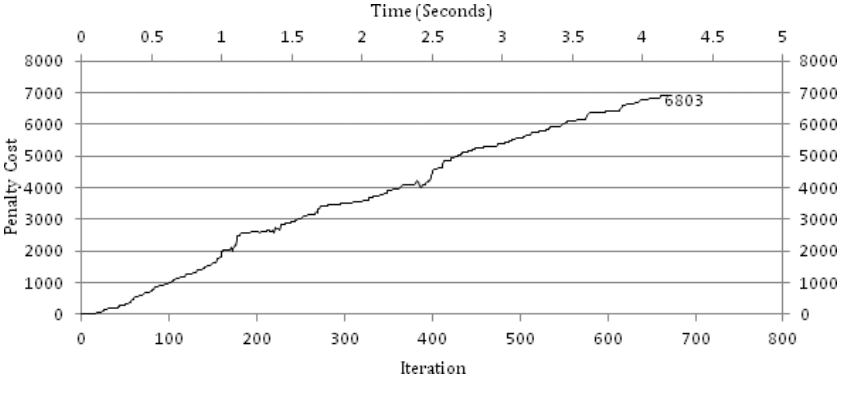


Fig. 6. Dataset 1 construction phase using TS with CD.

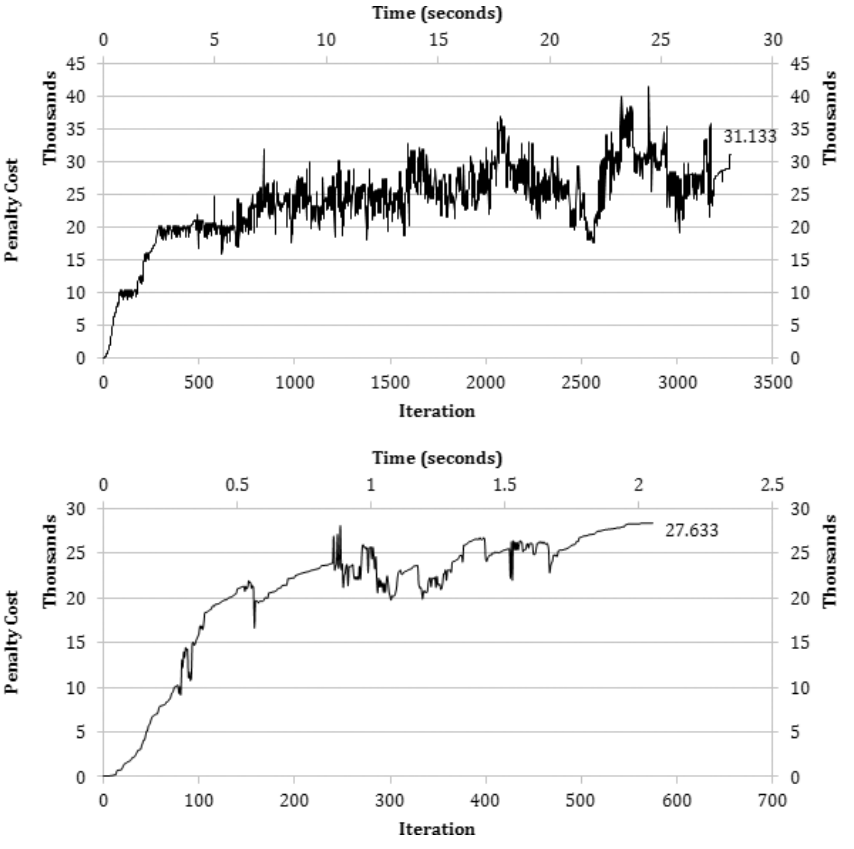


Fig. 7. Performance of the construction phase on dataset 4 with Tabu and Tabu together with conflicts dictionary.

However, dataset 4 has shown a different pattern. Dataset 4 is one of the most constrained problems. The top chart of Fig. 7 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 s (3281 iterations) to find a best complete solution, amongst 10 trials, with a penalty cost of 31,133 while TS with CD took only 2.03 s (567 iterations) to find one with a penalty of 27,633. That is a performance improvement of around 93% with solution value improvement of 11.2%.

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) 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 Fig. 8. Nonetheless, TS with CD algorithm performs slightly better than standard TS even though the problem itself tends to be easy to solve. In 10 trials,

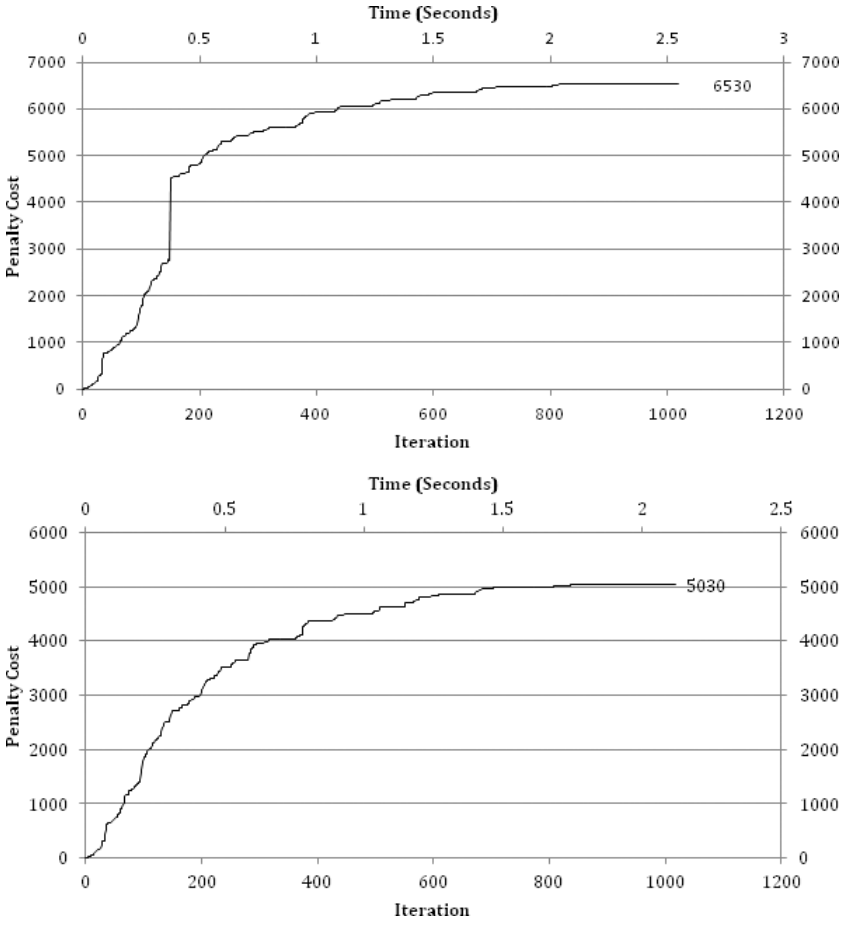


Fig. 8. Dataset 5 using standard TS (top chart) and TS with CD (bottom chart).

standard TS obtained 6530, as a best solution value, in 2.58 s (1020 iterations) while TS with CD achieved 5030, as a best solution value, in 2.21 s (1050 iterations).

4.3. *Enhancement phase testing and analysis*

We compare 4 methods labeled method 1, method 2, method 3, and method 4, 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 9–11, show 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

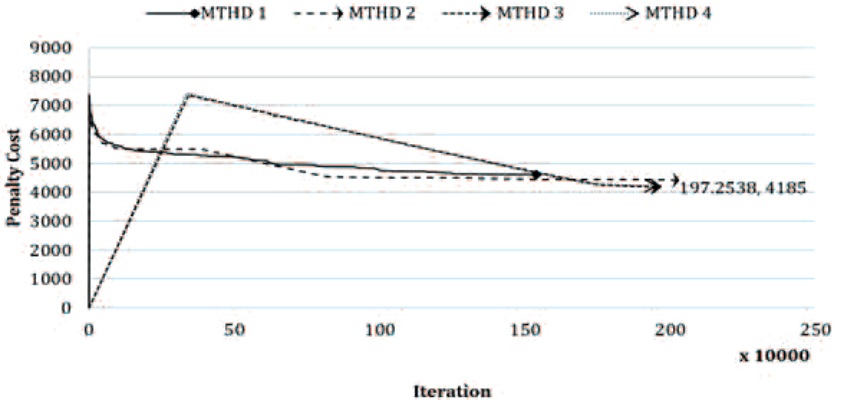


Fig. 9. Performance on dataset 1 of the enhancement phase.

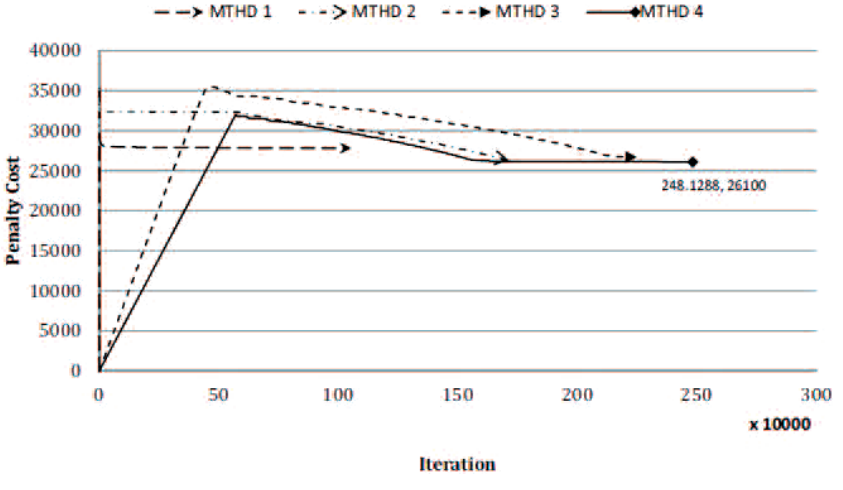


Fig. 10. Performance on dataset 6 of the enhancement phase.

suggests that it employs these iterations cycles either in backtracking or accessing nonefficient 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 flavors 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 GD 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) 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

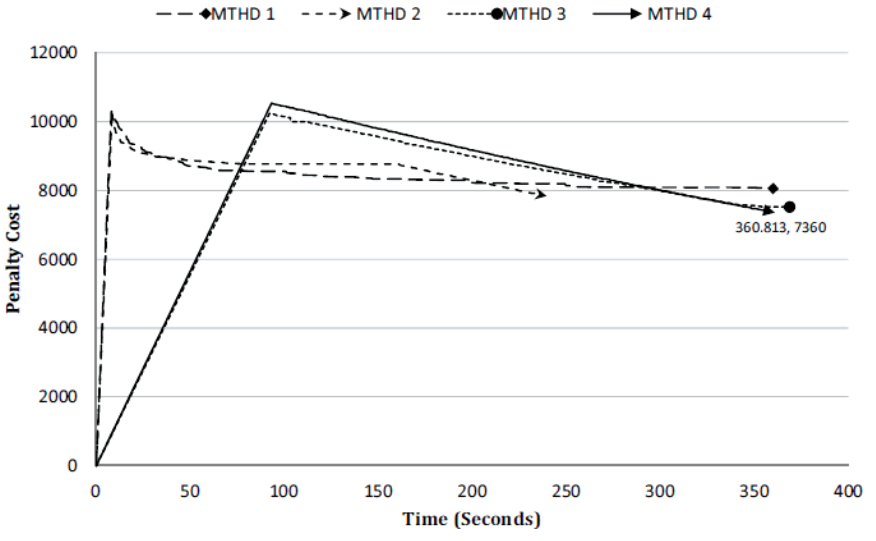


Fig. 11. Performance on dataset 8 of the enhancement phase.

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 9 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 Figs. 10 and 11 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.

4.4. *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 ETP 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[31] implemented a constraint-based solver, which constructs a complete solution, followed by a HC and a GD approach for improving the solution.
(2) Gogos *et al.*[32,33] used a Greedy Randomized Adaptive Search Procedure, followed by SA, and integer programming.
(3) Atsuta *et al.*[34] developed a constraint satisfaction solver combined with Tabu Search and iterated local search.
(4) De Smet[35] 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[36] 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 s. 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.

Figure 12 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 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

Instance #	1	2	3	4	5	6	7	8	9	10	11	12
T. Muller	4370	**400**	10049	18141	2988	26585	4213	7742	**1030**	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	-	**5264**
G. Smet	6670	623	-	-	3847	27815	5420	-	1288	**14778**	-	-
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	**4185**	415	10002	**17662**	**2693**	26705	**4149**	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	**9335**	18658	2718	**26100**	4181	**7360**	1050	14918	**31177**	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

Fig. 12. Comparative results.

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.

5. Conclusion and Future Work

We presented our proposed approach to solve the exam timetabling problem using four different metaheuristics search method. 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 MEGD 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 preknowledge of the exam timetabling problem. In the near future we will investigate advanced variables ordering heuristics[14] as well as evolutionary techniques using a parallel architecture.[37,38]

References

1. A. Schaerf, A survey of automated timetabling, *Artif. Intell. Rev.* **13**(2) (2009) 86–127.
2. R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot and S. Y. Lee, A survey of search methodologies and automated system development for examination timetabling, *J. Scheduling* **12**(1) (2009) 55–90.
3. B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke and S. Abdullah, An extended great deluge approach to the examination timetabling problem, *MISTA 2009, Multidisciplinary International Scheduling Conf. Theory and Applications* (2009), pp. 20–69.
4. C. Gogos, P. Alefragis and E. Housos, An improved multi-staged algorithmic process for the solution of the examination timetabling problem, *Ann. Oper. Res.* **194**(1) (2012) 203–221.
5. S. Abdullah and H. Turabieh, On the use of multi neighbourhood structures within a tabu-based memetic approach to university timetabling problems, *Inform. Sci.* **191**(0) (2012) 146–168. Data Mining for Software Trustworthiness.
6. N. R. Sabar, M. Ayob, R. Qu and G. Kendall, A graph coloring constructive hyper-heuristic for examination timetabling problems, *Appl. Intell.* **37**(1) (2012) 1–11.
7. A. Abuhamdah, M. Ayob, G. Kendall and N. R. Sabar, Population based local search for university course timetabling problems, *Appl. Intell.* **40**(1) (2014) 44–53.
8. R. Qu, N. Pham, R. Bai and G. Kendall, Hybridising heuristics within an estimation distribution algorithm for examination timetabling, *Appl. Intell.* **42**(4) (2015) 679.
9. N. Mansour, V. Isahakian and I. Ghalayini, Scatter search technique for exam timetabling, *Appl. Intell.* **34**(2) (2011) 299–310.
10. N. R. Sabar, M. Ayob, G. Kendall and R. Qu, Grammatical evolution hyper-heuristic for combinatorial optimization problems, *IEEE Trans. Evol. Comput.* **17**(6) (2013) 840–861.
11. S. A. Rahman, A. Bargiela, E. K. Burke, E. Özcan, B. McCollum and P. McMullan, Adaptive linear combination of heuristic orderings in constructing examination timetables, *Eur. J. Oper. Res.* **232**(2) (2014) 287–297.
12. S. Abdul-Rahman, E. K. Burke, A. Bargiela, B. McCollum and E. Özcan, A constructive approach to examination timetabling based on adaptive decomposition and ordering, *Ann. Oper. Res.* **218**(1) (2014) 3–21.
13. Y. Bykov and S. Petrovic, An initial study of a novel step counting hill climbing heuristic applied to timetabling problems, in *Proc. Multi-disciplinary International Scheduling Conference: Theory and Applications (MISTA 2013)* (2013), pp. 691–693.
14. M. Mouhoub and B. J. Jashmi, Heuristic techniques for variable and value ordering in csps, in *13th Annual Genetic and Evolutionary Computation Cont., GECCO 2011, Proc.*, pp. 457–464.
15. J. F. Allen, Maintaining knowledge about temporal intervals, *CACM* **26**(11) (1983) 832–843.
16. M. Mouhoub, Dynamic path consistency for interval-based temporal reasoning, *21st International Conference on Artificial Intelligence and Applications(AIA '2003)* (ACTA Press, 2003), pp. 393–398.
17. M. Mouhoub, Systematic versus non systematic techniques for solving temporal constraints in a dynamic environment, *AI Commun.* **17**(4) (2004) 201–211.
18. G. Kendall and N. Mohd Hussin, A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology, *Pract. Theory Autom. Timetabling* **3616** (2005) 270–293.
19. B. D. HughesL. T. G. Merlot, N. Boland and P. J. Stuckey, A hybrid algorithm for the examination timetabling problem, *Pract. Theory Autom. Timetabling* **2740** (2003) 207–231.
20. K. A. Dowsland, Simulate annealing, in *Modern Heuristics Techniques for Combinatorial Problems*, Chapter 2 (1995), pp. 20–69.

21. B. McCollum and P. McMullan, The second international timetabling competition: Examination timetabling track, University of Nottingham, Queen's University, Notingham, Belfast, Technical Report: QUB/IEEE/Tech/ITC2007/Exam/v4.0/17 2007 (2009).
22. J. P. Newall, E. K. Burke and R. F. Weare, A memetic algorithm for university exam timetabling, *Pract. Theory Autom. Timetabling* **1153** (1996) 241–250.
23. C. K. Chan, H. B. Gooi and M. H. Lim, Co-evolutionary algorithm approach to a university timetable system, in *The 2002 Congress on Evolutionary Computation*, Vol. 2, Honolulu (2002), pp. 1946–1951.
24. S. Petrovic, E. K Burke, B. MacCathy and Qu, Knowledge discovery in a hyperheuristic for course timetabling using case-based reasoning, *Pract. Theory Autom. Timetabling (PATAT'02)* (2002).
25. G. Laporte, M. W. Carter and S. Y. Lee, Examination timetabling: Algorithmic strategies and applications, *J. Oper. Res. Soc.* **47** (1996) 373–383.
26. M. Mouhoub, Analysis of approximation algorithms for maximal temporal constraint satisfaction problems, in *The 2001 International Conference on Artificial Intelligence (ICAI-2001)* (2001), pp. 165–171.
27. G. Dueck, New optimisation heuristics for the great deluge algorithm and the record-torecord travel, *J. Comput. Phys.* **104** (1993) 86–92.
28. E. K. Burke and J. P. Newall, Solving examination timetabling problems through adaption of heuristic orderings, *Ann. Oper. Res.* **129**(1–4) (2004) 107–134.
29. B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke and R. Qu, A new model for automated examination timetabling, *Ann. Oper. Res.* **194**(1) (2012) 291–315.
30. M. W. Carter and G. Laporte, Recent developments in practical examination timetabling, *Pract. Theory of Autom. Timetabling* **1153** (1996) 3–19.
31. T. Müller, Itc2007 solver description: A hybrid approach, in *Proceedings of the 7th International Conf. on the Practice and Theory of Automated Timetabling* (2008).
32. C. Alefragis, E. Gogos and P. Housos, Multi-staged algorithmic process for the solution of the examination timetabling problem, *Pract. Theory Automat. Timetabling (PATAT), 2008* (2008).
33. C. Gogos, G. Goulas, P. Alefragis, V. Kolonias and E. Housos, Distributed scatter search for the examination timetabling problem, *PATAT 2010* (2012), p. 211.
34. M. Nonobe, T. Atsuta and Ibaraki, An approach using a general csp solver, Technical Report (2008).
35. G. De Smet, ITC2007 examination track: Practice and theory of automated timetabling, Technical Report (2008).
36. N. Pillay, A developmental approach to the examination timetabling problem, Technical Report (2008).
37. R. Abbasian and M. Mouhoub, A new parallel ga-based method for constraint satisfaction problems, *Int. J. Comput. Intell. Appl.* **15**(03) (2016).
38. R. Abbasian and M. Mouhoub, A hierarchical parallel genetic approach for the graph coloring problem, *Appl. Intell.* **39**(3) (2013) 510–528.