Parallel Metaheuristic Algorithm for Exam Timetabling

Nashat Mansour, Ghia Sleiman Haidar
Department of Computer Science and Mathematics,
Lebanese American University, Beirut, Lebanon
E-mails: nmansour@lau.edu.lb, ghia.sleimanhaidar@lau.edu.lb

***Abstract*. Exam timetabling is a computationally intractable problem, which requires heuristic techniques for producing good sub-optimal solutions within reasonable execution time. For large numbers of exams and students, sequential algorithms are likely to be time consuming. The purpose of this work is to present a parallel metaheuristic algorithm for producing good sub-optimal exam timetables in a reasonable time. Empirical results show that our proposed parallel scatter search algorithm yields good speed-up. Also, they show that the parallel algorithm improves solution quality in comparison with the sequential algorithm.**

Keywords: Metaheuristics; parallel algorithm; scatter search; exam timetabling.

I. INTRODUCTION

Exam timetabling refers to assigning exams to days, time periods and rooms, given a number of constraints, which can be hard or soft. An example of hard constraints is room capacity. Examples of soft constraints are number of students having consecutive exams. The timetabling problem is NP-hard. A number of heuristic algorithms have been developed for finding sub-optimal exam timetabling solutions. A recent survey of exam timetabling techniques appears in [1]. Examples of these techniques are: case-based reasoning technique in combination with a simulated annealing algorithm [2]; hybridization within a graph based hyperheuristic framework [3]; genetic and evolutionary algorithms [4]; tabu search [5]; scatter search algorithm [6]; hybrid heuristics [7].

For larger numbers of students and exams, sequential algorithms are likely to be slow. Hence, parallel algorithms should be explored. In this paper, we present a parallel scatter search (PSS) algorithm for exam timetabling to improve both execution time and solution quality. A small number of strategies to build parallel scatter search have been proposed [8, 9] . But, they are based on different parallelism models and none of these strategies has addressed the exam timetabling problem. Our purposed PSS is based on distributing computations over different processors with limited amount of tree-based communication.

This paper is organized as follows. Section II describes the exam timetabling problem. Section III presents the parallel metaheuristic algorithm for exam timetabling. In Section IV, we. Conclusions are given in Section V.

II. EXAM TIMETABLING PROBLEM DESCRIPTION

Given: A list of S students, a list of C course-exams, a pre-defined number of classrooms *R,* each with a maximum capacity Ψ_r, and a pre-defined number of Exam Periods $\prod$, exam timetabling consists of assigning time periods and rooms to the given set of exams of size E . These exams refer to courses in which students are enrolled based on choices allowed by their curricula. We consider exam timetabling with the following constraints:

- Eliminate / minimize the number of students having simultaneous exams (S_{SE}).
- Minimize the number of students having consecutive exams per day (S_{CE}).
- Minimize the number of students having 3 multiple exams per day (S_{3E}).
- Eliminate/minimize the number of students having 4 multiple exams per day ($S_{4E)}$.
- No exam is assigned to a room whose maximum capacity is less than the number of students taking the exam.

We define the following objective function (*OF)*, which determines the quality of timetabling solutions:

$$OF = \alpha * S_{SE} + \varphi * S_{CE} + \sigma * S_{3E} + \beta * S_{4E} + \gamma * (\Sigma_{1<=x<=\Pi} \Sigma_{1<=y<=R} \rho_{xy})$$

Where the last summation term gives the total number of rooms that violate the room capacity constraint.

III. PARALLEL ALGORITHM FOR EXAM TIMETABLING

A. Overview

Scatter search is a population-based meta-heuristic algorithm [10]. Scatter search starts with the generation of an

initial population of candidate exam timetables by the diversification generation method that involves diversification and randomization. Each timetable is represented by a list of scheduled course-exams, *CRS*. Each scheduled course-exam, $CRS_i(t, r)$, is associated with an exam period $t \in \{1, 2, \ldots, \Pi\}$ and an assigned room $r \in \{1, 2, \ldots, R\}$, for $i = 1, 2, \ldots, C$. From this initial population set, a smaller subset, called the reference set, is created by selecting the highest quality and diverse solutions by the reference set update method. Then, we select solutions subsets and apply, to every subset, both the combination and improvement methods to generate new solutions. The subset generation method generates subsets from the reference set solutions. The solution combination method combines subset solutions to yield new solutions; the improvement method attempts to improve the solution quality. The reference set update method maintains the reference set with high quality and diverse solutions. These methods are repeated until no new solutions are added to the reference set.

B. Parallel Scatter Search Algorithm

The Parallel Scatter Search (PSS) algorithm applies tree-based communication among all processors $Processor_p$, where $1 \le p \le n\_pr$, when unifying the RefSet in every iteration. Fig. 1 shows the PSS algorithm steps. First, $Processor_0$ *(*or *RootProcessor)* broadcasts the input data files to all other processors; the input data files are courses file, student enrolments file, conflict file and rooms file. Then, each processor determines its own population size, $ProcPopSize_p$, which is defined as a function of the total *PopSize* and the number of processors (*n_pr)*. If *PopSize* is divisble by n_pr, then $ProcPopSize_p$ will be equal for all p=0, 2,…, *n_pr-1*; otherwise, the reminder of the division is distributed evenly among the lower rank processors .

In Step 4 of Fig. 1, each processor, $Processor_p$, runs the diversification generation method to create its local population. This is followed by the improvement method locally (Step 5). After the creation of $ProcPopSize_p$, every $Processor_p$ generates its complete local reference set $ProcRefSet_p$. Once all reference sets are defined, PSS merges all $ProcRefSet_p$ in a bottom-up tree-based communication way starting from leaf processors and moving up toward *RootProcessor*. Each $Processor_p$ ($0 \le p \le n\_pr$) creates a local $ProcRefSet_p$ of size equal to *GlobalRefSet*. Then, every two processors unify their corresponding *GlobalRefSet* in a bottom-up tree-based communication toward the *RootProcessor*. Unification means that the two local RefSets are compared and the solutions of higher quality and higher diversity are selected from both. The tree-based communication system in PSS is implemented to reduce the communication time. After having a complete unified *GlobalRefSet* from all $ProcRefSet_p$, the *RootProcessor* broadcasts it to all other processors.

In Step 12, each $Processor_p$ ($0 \le p \le n\_pr$) creates it local *ProcSubsets* of size $SubsetSize_p$. The subset generation method creates 2-element subsets. After $SubsetSize_p$ *ProcSubsets* are determined, combination and improvement methods are applied to the subsets in order to yield improved solutions. The reference set update method is then applied on every generated solution to check whether this new candidate solution can be added to the processor reference set $ProcRefSet_p$.

This procedure is applied on every processor. If $ProcRefSet_p$ is changed at any processor, all processors re-unify all $ProcRefSet_p$ in bottom-up tree-based communication towards the *RootProcessor*, where the *RootProcessor* re-generates a *GlobalRefSet* and broadcasts it to all processors. Then, another iteration is initiated. PSS ends when no new solution is added to *GlobalRefSet.* The communication performance of tree-based PSS could be more promising for appropriate interconnection networks that support tree communication than the cluster results.

---

1. *Broadcast InputData (*from *RootProcessor* to *n_pr processors);*
2. *Determine* $ProcPopSize_p$*(n_pr processors);*
3. for each *processor* $p = 0, \ldots, n\_pr - 1$, do in parallel
4. *DiversificationGeneration(*$ProcPopSize_p$*);*
5. *Improvement(*$ProcPopSize_p$*);*
6. *RefSetUpdate(*$ProcRefSet_p$*);*
   end for;
7. *Gather* $GlobalRefSet(ProcRefSet_{p,}\ p = 0, \ldots, n\_pr - 1)$ *in Buttom-up Tree-based Communication;*
8. if *(RootProcessor)* then
9. *Broadcast GlobalRefSet;*
   end if;
10. repeat
11. for each processor $p = 0, \ldots, n\_pr - 1$, do in parallel
12. *SubSetGeneration(SubsetSize, Subsets);*
13. *Select ProcSubsets(*$SubsetSize_p$*, ProcSubsets);*
14. *while (ProcSubsets ≠ Ø) do*
15. *SelectSubset s from ProcSubsets;*
16. *SolutionCombination(s) to produce solution x;*
17. *Improvement (x) to produce x';*
18. *RefSetUpdate(*$ProcRefSet_p$*) with x';*
    end while;
    end for;
19. if (*at least 1* $ProcRefSet_p$ *is improved*) then
20. *Gather* $GlobalRefSet(ProcRefSet_{p,}\ p = 0 \ldots, n\_pr - 1)$ *in Bottom-up Tree-based Comm.;*
    end if;
21. if (*RootProcessor*) then
22. *Broadcast GlobalRefSet;*
    end if;
23. until (*no new solution in GlobalRefSet*);

Figure 1. Parallel Scatter Search Algorithm.

C. Diversification Generation Method

The diversification generation method in PSS is implemented in full parallelism over all processors. That is, the initial population size is divided equally or near-equally among the *n_pr* processors.

The diversification generation method combines diversification with randomization. For this purpose, we use controlled randomization and a frequency-based memory technique to generate an initial set of diverse solutions. In our implementation, we divide the range $\prod$ into 4 sub-ranges. To assign a period to a given exam, we first select the sub-range that is the least-frequently selected one so far, and then randomly select a period value within this sub-range. We also assign a randomly selected room. This method is replicated on all the *n_pr* processors, so that each processor is responsible for creating its $ProcPopSize_p$ initial solutions.

D. Improvement Method

The improvement method in PSS is also implemented in full parallelism for improving the quality of the local solutions generated on the individual processors.

Our improvement method sorts the exams in terms of decreasing number of enrolled students. Then, it considers every exam starting with the one that has the largest number of enrolments down to the exam with the lowest enrolments. For each exam, we apply a better-move heuristic procedure. In this procedure, we reassign an exam to other randomly selected period and randomly selected room. This reassignment is repeated a limited number of times, say 5, and the assignment that yields the greatest decrease in the value of OF is accepted. If none of these 5 moves leads to a decrease in OF, the original period and room assignments are kept.

E. Reference Set Update Method

The reference set update method is invoked in one of two ways. One way corresponds to generating the initial reference set and the second corresponds to updating a previous reference set. After updating the local reference set, inter-processor communication takes place in order to produce a global reference set for the next iteration.

The reference set is composed of two subsets which are equal in size, in our implementation. The high quality subset, *HQRefSet*, includes the highest quality solutions. The second subset is the subset of diverse solutions, *DivRefSet*. The initial reference set on a processor p is denoted as $ProcLocalRefSet_p$. It is selected from the improved initial population of solutions generated on processor p whose size is *ProcPopSize*. The local reference set, $ProcLocalRefSet_p$, has a size, $RefSetSize_p$, which is typically less than 20% of *ProcPopSize*.

The local reference set, $ProcLocalRefSet_p$, is the union of $HQRefSet_p$ and $DivRefSet_p$. $HQRefSet_p$ consists of the best solutions, selected from the local population, as determined by the objective function values. $DivRefSet_p$ items are, then, selected from the initial population based on diversity. To measure the diversity of a solution, we use a function $\partial(x', x'')$ that gives the number of different exam assignments (to periods) that appear in solution x' and solution x''. To select the *DivRefSet* solutions, we define $\partial_{min}(x)$ as the minimum distance between solution x and all the solutions x' in the $HQRefSet_p$. After ordering the solutions in the local population based on the minimum distance $\partial_{min}(x)$, we select the items that have the greatest distance values and add them to the $DivRefSet_p$.

When all *n_pr ProcLocalRefSet*s are created, the global reference set update method gathers all *n_pr ProcLocalRefSet*s on the root processor to generate the global reference set, *GlobalRefSet*, and broadcast it to all processors as a pre-requisite stage required by the generation method. The gather step is implemented by a tree-based interprocessor communication where the leaves of the tree correspond to the local processor reference sets. Adjacent pairs of processors compare their *ProcLocalRefSets* and unify the *HQRefSet* and *DivRefSet* parts. The resulting reference set is forwarded to one higher level in the tree so that the two adjacent unified reference sets will also be compared and unified. This tree traversal continues up the tree until the *GlobalRefSet* is produced at the root processor. Then, *GlobalRefSet* is broadcast to all processors.

F. Subset Generation Method

The Subset Generation method generates subsets of the reference set solutions on *n_pr* processor. We choose to generate subsets of size 2 (subset type 1) and subsets of size 3 (subset type 2) by augmenting 2-elements subset with the best solution outside this subset. Thus, for subset type 1, we pair all possible solutions (x, x') from the *GlobalRefSet* generated by the parallel reference set update method.

The subset generation method is a fast operation; we replicate this method on all *n_pr* processors.

G. Solution Combination Method

The Solution Combination Method combines the solutions grouped in subsets in the preceding step. Our combination method approach adapts a voting technique depending on the *OF* values and applies a random selection of Room and Exam Period in case we have the same scores. First, we initiate an empty solution *s* with all courses ordered in decreasing order according to the number of student enrolments. Then, for each course *c* in the ordered solution, we consider the partial *OF* values that result from the period-room assignments found in each of the solutions to be combined. Then, we select the room and the exam period that correspond to the lowest *OF* value.

The Solution Combination Method is applied in parallel by distributing the subsets generated (in the Subset Generation Method) over the *n_pr* processors. That is, each processor combines a different part of the generated subsets.

IV. EMPIRICAL RESULTS

A. Procedure

We apply our parallel scatter search algorithms to four real-world subject problem instances SP1-SP4, where E varies between 473 and 634, number of students varies between 3652 and 3794, and R = 38. The solution quality is evaluated by the objective function (*OF*) and its 5 components. The objective function user-defined weights used are: $\alpha = 200$, $\varphi = 1$, $\sigma = 10$, $\beta = 100$, and $\gamma = 100$. The performance of the parallel algorithm is measured by its parallel efficiency (*PEff*). We execute our parallel program on a cluster of PCs running Linux operating system and connected by an Ethernet network. Each PC has 2.33 GHz CPU and 2 GByte RAM memory. The programs are implemented in C++ and MPI-2 software.

B. Results and Discussions

Table 1 shows the results of executing PSS using *PopSize* = 1024 and *RefSize* = 32 for SP1-SP4 on *n-pr*=1,2,4,8,16. Table 2 gives both average *PEff* values and average *OF* values for the four subject problem instances and different numbers of processors. From these results, we infer the following:

(a) PSS does reduce the execution time with respect to the sequential SS algorithm. For example, Table 1 shows that sequential SS takes 1,929 min. for SP4 whereas PSS takes 134 min. on 16 processors for the same number of iterations, 10. This reduction in time allows including time demanding scatter search features that are likely to improve the solution quality such as combining subsets with more than 2 solutions, which explores more areas of the search space.

(b) PSS is necessary for handling/solving large timetabling problems in reasonable times by running them on larger number of processors.

(c) PSS is based on an adequate parallelization strategy since it yields high efficiency (*PEff*) values in the great majority of the results and good average *PEff* values. This is also demonstrated in the decrease in the time per iteration as the number of processors increase. Furthermore, PSS is expected to yield better efficiency on other parallel architectures that support tree-based inter-processor communication

(d) In most cases, the values of *OF* decrease as a result of parallel execution. However, the values of the terms that make up the *OF* expression show that the exam timetables produced are quite adequate. For example, for problem SP3, PSS yields an exam timetable with *OF*=536 on 8 processors; this corresponds to 1 simultaneous exam, 266 consecutive exams, and 7 students with 3 exams per day.

V. CONCLUSION

We have proposed a parallel scatter search algorithm for the exam timetabling problem, executed on a cluster of PCs. Parallelism allows us to handle larger exam timetabling problems within reasonable time. The experimental results showed that PSS produces good exam timetables with good parallel efficiency. Further work will compare PSS with other parallel scatter search models and extend empirical work to more cases such as varying population size and the reference set size for studying the quality of the exam timetables in comparison with sequential scatter search.

ACKNOWLEDGMENT

This work was partially supported by the Lebanese American University and the Lebanese National Council for Scientific Research.

REFERENCES

[1] 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," *Journal of Scheduling*, vol. 12, pp. 55-89. 2009.

[2] E.K. Burke, A.J. Eckersley, B. McCollum, S. Petrovic, and R. Qu, "Similarity measures for exam timetabling problems," *Proceedings of the 1st multidisciplinary conference on scheduling: theory and applications*, Nottingham, August 13-16, pp. 120-136, 2003.

[3] R. Qu and Burke, E.K., "Hybridisation within a graph based hyperheuristic framework for university timetabling problems," *Journal of Operational Research Society*, vol. 60, pp. 1273-1285, 2009.

[4] C.Y. Cheong, K.C. Tan, and B. Veeravalli, "A multi-objective evolutionary algorithm for examination timetabling," *Journal of Scheduling*, vol. 12, pp. 121-146, 2009.

[5] G. Kendall and N. Mohd Hussin, "A Tabu Search hyper-heuristic approach to the examination timetabling problem at the MARA University of Technology," In E. Burke. And M. Trick (Eds.) *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*. Lecture Notes in Computer Science 3616, pp. 270-293, 2005.

[6] N. Mansour, V. Isahakian, and I. Galayini, "Scatter search technique for exam scheduling," *Applied Intelligence*, in press.

[7] N. Pillay and W. Banzhaf, "A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem," *European Journal of Operational Research*, vol. 197, pp. 482-491, 2009.

[8] W. Bozejko and M. Wodecki, "Parallel Scatter Search algorithm for the flow shop sequencing problem," *Parallel Processing and Applied Mathematics*, vol. 4967, pp. 180-188, Heidelberg: Springer Berlin, 2008.

[9] F. Garcia-Lopez, M.G. Torres, B.M. Batista, J.A. Moreno-Perez, and J.M. Moreno-Vega, "Solving features subset selection problem by a parallel scatter search," *European Journal of Operational Research*, vol. 196, pp. 477-489, 2006.

[10] F. Glover, M. Laguna, and R. Marti, "Fundamentals of scatter search and path relinking," *Control and Cybernetics*, vol. 39, pp. 575-589, 2000.

TABLE 1. RESULTS FOR PSS, PopSize=1024, RefSize=32

Subject Problem	No. of Procs	S_{SE}	S_{CE}	S_{3E}	S_{4E}	R_V	*OF*	No. of Iterations	Execution Time (Mins)	Time/ Iteration	*PEff*
SP1	1	0	357	16	0	0	517	11	919	83.54	-
	2	1	405	11	0	0	715	17	704	41.41	65.25
	4	1	398	22	0	0	818	15	314	20.93	73.2
	8	0	352	10	0	0	452	13	139	10.69	82.65
	16	0	403	8	0	0	483	6	37	6.16	155.2
SP2	1	0	329	12	0	0	449	6	545	90.83	-
	2	0	314	6	0	0	374	13	562	43.23	48.5
	4	0	213	0	0	0	213	4	143	35.75	95.3
	8	0	203	1	0	0	213	3	56	18.66	121.65
	16	0	398	14	0	0	538	21	117	5.57	29.2
SP3	1	1	328	7	0	0	598	6	812	135.33	-
	2	1	224	6	0	0	484	23	1527	66.39	26.6
	4	1	268	3	0	0	498	15	476	31.73	42.6
	8	1	266	7	0	0	536	9	157	17.44	64.65
	16	1	219	6	0	0	479	16	145	9.06	35
SP4	1	0	276	2	0	0	296	10	1929	192.9	-
	2	0	272	1	0	0	282	10	990	99	97.4
	4	0	260	1	0	0	270	17	836	49.17	57.7
	8	0	260	2	0	0	280	31	754	24.32	32
	16	0	261	1	0	0	271	10	134	13.4	90

TABLE 2. AVERAGE *PEff* AND *OF* VALUES BASED ON TABLE 1.

2 Procs		4 Procs		8 Procs		16 Procs	
PEff	***OF***	***PEff***	***OF***	***PEff***	***OF***	***PEff***	***OF***
59.4	463.75	67.2	449.75	75.23	387.75	77.36	442.75