Solving Exam Timetabling Using Distributed Evolutionary Computation

Mihej Komar, Dorde Grbic, Marko Cupic
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
Unska 3, 10000 Zagreb, Croatia
E-mail: {mihej.komar, dorde.grbic, marko.cupic}@fer.hr

Abstract. *In this paper we describe specific exam timetabling problem encountered at our institution. We enumerate and briefly explain population based evolutionary computation algorithms that we implemented and then focus on coarse-grained algorithm parallelization. Parallelization is accomplished by employing a computer network in which separate populations exchange best solutions. Solution exchange is guided by various migration parameters such as network exchange topology. Proposed topologies are tested and experimental results are discussed. We conclude that usage of a computer network, exchange topologies, and large amount of processor time enabled us to find a good-quality exam timetable.*

Keywords. Exam timetabling, evolutionary computation, distributed algorithms

1. Introduction

Exam-timetabling is one of several timetabling problems present at each university. This problem is a combinatorial optimization problem and it is considered to be NP-complete [8]. There are many variations of this problem. In this paper we focus on a problem in which a set of courses should be scheduled in a predetermined set of timeslots. More than one course can be assigned to a single timeslot, up to timeslot capacity. A single course can be assigned only to a single timeslot. In order to produce a good-quality timetable, a number of additional constraints should also be satisfied. The goal is to make the exam timetable at least as good as experienced human would make it. Early attempts to solve this problem used heuristic algorithms such as graph coloring heuristic methods [12] in which courses were represented by nodes and timeslots were represented by node colors. Although those algorithms are efficient in solving small timetabling problems, they are mostly not efficient for larger problem instances [1].

Later attempts used meta-heuristic algorithms in attempt to solve timetabling problems. Meta-heuristic algorithms are a group of generic algorithms capable of solving large variety of problems that cannot be solved in a polynomial time using exact search methods [11]. Meta-heuristic algorithms can be divided in two groups: *single-solution based* algorithms and *population-based* algorithms. The former start with a single solution that is then iteratively modified in an attempt to satisfy given constraints, until globally optimal solution is reached. Drawback of these methods is that they usually get stuck in the local optimal solution and are unable to find better solutions in other parts of the search space. Population-based algorithms use multiple solutions at the same time (called a *population*). These algorithms try to cover a larger area of the search space in parallel thus increasing the possibility of finding optimal or nearly optimal solution. The drawback of these algorithms is that they are mostly concentrated on exploration and are often unable to thoroughly exploit the population neighborhood in the search space.

Finding optimal parameters for such algorithms is an additional problem. Inappropriate choice of the parameters can either lead to premature convergence of population towards locally optimal solution or make the algorithm spend too much time in finding optimal solution. The problem of parameter selection, although important, will not be discussed in this paper. To amend this, we complemented the developed population based algorithms with a simple local-search procedure. Population-based algorithms used in this paper are: Steady-state Genetic Algorithm, Generation Genetic Algorithm, Harmony Search, Simple Immune Algorithm, and $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System.

Parallelization is another approach often used when dealing with hard optimization problems. Because of the out-of-the-box multi-core computer processors and large computer networks are available in university environment, it is possible to deploy a large set of parallel meta-heuristic algorithms. The idea is that algorithms could do better if every algorithm instance creates its own population, tries to improve it and exchanges best solutions with other algorithms in the network. Solution exchange policy is determined by a topology specified before the algorithms started.

To deal with exam timetabling problem present at out institution, we have developed a Java-based framework. It offers a support for a distributed execution of multiple heterogeneous evolutionary computation algorithms. We implemented several evolutionary computation based algorithms. Using these sequential algorithms as a simple building blocks, we tested the behavior of resulting distributed algorithm on actual problem instance.

In this paper several topologies are tested and results are discussed. In the next two sections University Exam Timetabling Problem and applied meta-heuristic algorithms will be explained in more details. Section 4 gives the description of topologies that were tested in this study. Section 5 discusses experimental results and Section 6 concludes the paper.

2. Exam Timetabling Problem

Exam timetabling problem is a well-known NP-complete combinatorial problem present in colleges and universities with a large number of students and courses, especially if many courses are elective. The problem is how to efficiently assign timeslots to courses in presence of additional constraints. There are two types of constraints: hard and soft constraints. Hard constraints are those whose violation makes the timetable infeasible. On the other hand, soft constraints can be violated, but their violation should be reduced to a minimum. The problem described in this paper is the one we face at our institution. The difference from similar problems is in the choice of soft constraints and the evaluation method of timetable quality.

The exam timetabling problem can be described as follows. It has a fixed duration (e.g., two weeks) and the timeslots are defined in advance, without overlapping. The students are divided into several modules, where some modules have a lot of enrolled students and some have only a few. Different modules can have overlapping set of mandatory courses. During the scheduling process, perceived course difficulty, later denoted as *course weight*, must also be taken into account. We collect these weights using an anonymous poll. Time distance between more difficult courses and those sharing a lot of enrolled students should be as large as possible. However, it is more important to satisfy this requirement for courses belonging to same modules and for courses offered on the same year. The latter two constraints constitute soft constraints. Hard constraints are defined as follows: (i) no two courses sharing at least one enrolled student can be assigned to the same timeslot, and (ii) the total number of students in courses assigned to each timeslot must not exceed the predefined timeslot capacity. The problem of assigning exam rooms to each course once the timeslots are assigned is handled separately [13] and will not be described here.

The exam timetabling problem we tackled can be semi-formally defined as follows. We define a set of courses as $C = \{c_1, \ldots, c_{n_C}\}$, set of timeslots as $T = \{t_1, \ldots, t_{n_T}\}$, and the set of students as $S = \{s_1, \ldots, s_{n_S}\}$. Let P_a denote a set of all course pairs that share at least one enrolled student. We define a set $P = \{(c_j, c_k)\}$ as the largest subset of P_a, where courses c_j and c_k are both mandatory on same module. We define functions $cPenalty$, $tPenalty$, and $dPenalty$. Function $cPenalty(c_i, c_j)$ represents a penalty between the courses c_i and c_j placed close together in the time (the penalty is higher for the courses which have greater weight and have a larger number of enrolled students). Function $tPenalty(t_x, t_y)$ returns the penalty between the timeslots t_x and t_y. Function $dPenalty(d_{c_x}, d_{c_y})$ returns the additional penalty that is applied to the pairs of courses (c_x, c_y) belonging to the same module.

Function $cPenalty(c_i, c_j)$ is defined as a sum of penalties calculated for each student enrolled in both courses. Each student can increase the penalty value by 1 (if both courses are in student's regular module), by 0.5 (if both courses belong to the same academic year the student is currently enrolled, but one or both of them are not a part of the module), by 0.25 (if one course is a part of the academic year the student is currently enrolled, but the other one is not), or by 0.125 otherwise. To obtain the final value, this sum is then multiplied by $min(w_i, w_j)$ where w_k denotes the weight of course c_k.

The function $tPenalty(t_x, t_y)$ is a monotonically decreasing, defined by:

$$tPenalty(t_x, t_y) = \alpha^{1-\frac{distance(t_x,t_y)}{24}} \quad (1)$$

where $distance(t_x, t_y)$ denotes a distance between the timeslots, expressed in hours.

The function $dPenalty(d_{c_x}, d_{c_y})$ is monotonically decreasing function defined as:

$$dPenalty(d_{c_x}, d_{c_y}) = \begin{cases} 200, & \Delta = 0 \\ 100, & \Delta = 1 \\ d(\Delta), & \Delta \geq 2 \end{cases} \quad (2)$$

where

$$d(\Delta) = \frac{60}{1 + \exp\left((0.2 \cdot \Delta)^3\right)}. \quad (3)$$

This function is calculated only for the pairs of courses (c_x, c_y) from set P. The value Δ represents the distance between the days in which courses c_x and c_y are scheduled. Value d_{c_i} represents the day in which a course c_i is scheduled.

The penalty of a timetable, denoted p, is the measure of soft constraints violation. It has two contributors p_1 and p_2:

$$p = p_1 + p_2. \quad (4)$$

The penalty p_1 is defined as:

$$p_1 = \sum_{(c_x,c_y)\in P} dPenalty(d_{c_x}, d_{c_y}), \quad (5)$$

while the penalty p_2 between all timeslots is calculated by:

$$p_2 = \sum_{i=1}^{n_T-1} \sum_{j=i+1}^{n_T} tPenalty(t_i, t_j) \cdot r(t_i, t_j) \quad (6)$$

and

$$r(t_i, t_j) = \sum_{c_k\in c(t_i)} \sum_{c_l\in c(t_j)} cPenalty(c_k, c_l) \quad (7)$$

where function $c(t_i)$ returns a set of courses that are assigned to the timeslot t_i.

All constants used in equations (1), (2), and (3) are appropriate for the problem instance we had to solve, and are obtained by trial-and-error attempts. In our experiments, we choose $\alpha = 7$. For other problem instances, we suggest the same function shape as defined by equations (1), (2), and (3) but the specific constant values can be adjusted to the problem at hand.

3. Evolutionary Computation Applied on Timetabling

In this section we briefly describe metaheuristic algorithms we used to solve exam the timetabling problem defined in Section 2.

Genetic Algorithms are a part of metaheuristic algorithms inspired by biological evolution by means of natural selection [6]. They are population-based algorithms that use several genetic operators to improve solutions in population. Operators such as selection, crossover, and mutation are used in order to produce new solutions. There are several different selection operators [4]. In our implementation 3-tournament parent selection is used. When solutions are selected, crossover operator recombines the two selected timetables and creates a new solution. It is expected for a new solution to be somewhere in between the two parent solutions in the search space. If the crossover operator would be the only operator used for search, a premature convergence would occur. To amend this, a mutation operator is introduced. Mutation takes a solution and randomly changes some parts of the timetable. We implemented two genetic algorithms: Steady-state and Generation Genetic Algorithm. Steady-state algorithm operates with a relatively stable population. When a new solution is created it replaces the worst solution in current population thus keeping the population size constant. Generation Genetic Algorithm produces a complete new generation before discarding the old one. Solutions from old generation are used as parents, and the best parent is copied into the new generation in order to implement elitist strategy.

Harmony search [9] algorithm is a population based algorithm sharing some similarities with

Genetic algorithms. This algorithm was derived from the natural phenomena of musicians behavior when they collectively play their musical instruments to come up with the aesthetically appealing harmony. This aesthetics is represented by evaluation function of produced timetables. More detailed description of Harmony Search Algorithm can be found in [1].

Simple Immune Algorithm [5, 10] is an algorithm inspired by mammal immune system. The algorithm maintains a population of antibodies (timetables) that are subjected to expansion process in each iteration of the algorithm. The expansion process includes the cloning of timetables and the application of a hyper-mutation operator. The cloning operator duplicates entire population a predefined number of times. The hyper-mutation then attempts to change a timeslot assigned to each course in accordance with predetermined probability.

$\mathcal{MAX}$-$\mathcal{MIN}$ *Ant System* ($\mathcal{MM}$AS) [7] uses a set of virtual agents, called artificial ants, to find good solutions. To apply $\mathcal{MM}$AS, problem has to be transformed into a problem of finding shortest path on a weighted graph. The ants incrementally build solutions by stochastically moving on the graph. Movement toward good solution is biased using artificial pheromone that the ants deposit during their graph movement. Amount of deposited artificial pheromone is proportional to the quality of the solution.

4. Distributing Evolutionary Computation Algorithms

The trend of parallelization in computing is becoming more important. Concerning the fact that CPU clock is not increasing as before, keeping the rate of performance growth is achieved by integration of more cores within a single CPU. Besides parallelization on a single computer, using a cluster of computers for solving problems is also common. Some meta-heuristic algorithms are very appropriate for parallelization. Parallelization of meta-heuristic algorithms can increase the speed of execution and modify the behavior of meta-heuristics, which can in turn raise the quality of achieved results [2, 3].

This paper describes coarse-grained parallelization. Coarse-grained parallelization implies the usage of several populations that occasionally exchange solutions. Each population can potentially search a different part of the search space, thus more separated populations can search wider areas of the search space. Occasional exchange of solutions from different areas of search space can allow good solution parts to combine, and can steer the search process towards the global optimum.

Each of the evolutionary computation algorithm has its advantages and disadvantages. It is expected that the disadvantages of some algorithms can be compensated by cooperation with other algorithms.

In order to specify a distributed algorithm, we must define its topology and migration parameters. The migration parameters are following: which solution to migrate, when to perform the migration, and the method for integration of the received solution into the population. There are many different topologies and some of them are shown on Figure 1. The ring topology can be unidirectional or bidirectional. In the experiment described in Section 5 we use the unidirectional type of the ring. Populations inside a single island (shadowed regions on Figure 1-b) rapidly exchange solutions, but islands mutually exchange solutions at a much slower rate. The idea behind this topology is that the algorithms working with populations inside a single island are exploring only one part of the search space. With the occasional exchange of solutions between islands, we are trying to combine the good parts of solutions from different islands. The toroidal grid is a 2D grid, where upper and lower rows, as well as leftmost and rightmost columns, are connected. Edges can be unidirectional (e.g., direction of solution's movement is from right-to-left and up-to-down) or bidirectional. In our experiments we use the bidirectional type of the toroidal grid. In the binary tree topology connections are directed upwards. The leaves of the tree are entirely independent. Nodes in every upper layer of the tree combine the solutions received from their children. The root node integrates all populations and receives the solutions from all of the explored areas of the search space.

A Java-based framework was developed for solving exam timetabling problem in a distributed manner. The framework can be used to

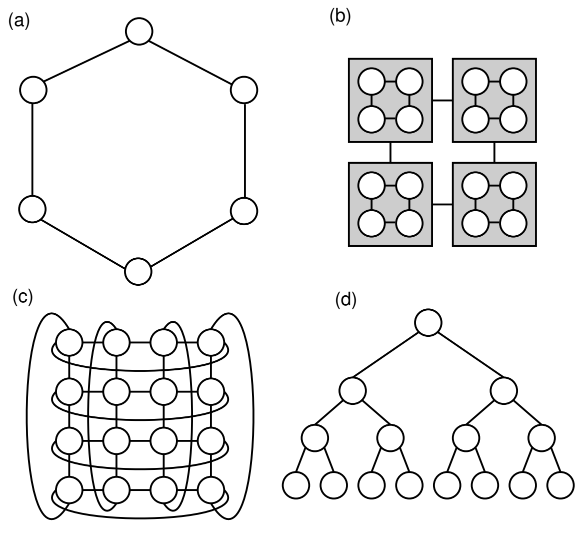

Figure 1. Examples of topology: (a) ring, (b) islands, (c) toroidal grid, and (d) binary tree.

create a distributed parallel algorithm composed of many sequential evolutionary computation algorithms. The user of the framework can accomplish this parallelization without being concerned with single algorithm parallelization issues. The framework takes the responsibility for transporting the solutions between algorithm instances; however, selection of solutions to be transported and the integration policy is left to the algorithms themselves. A framework can be easily adapted for solving other problems. It is designed to take care of communication between algorithms, to collect statistics, results, and other relevant information.

5. Experimental Results and Discussion

In order to test the algorithms and the distribution framework, we performed several experiments. Experiments were carried out on computers with quad-core Intel Core 2 CPU and 4 GiB of RAM, so every computer was able to execute up to four algorithms. We performed several experiments using different topologies: binary tree topology used 15 and ring, islands, and toroidal grid topology used 16 algorithms. For each topology we used four computers (one algorithm per a single processor core). Migration was performed at constant intervals of one or five seconds. In the island topology, migration between islands was performed four times slower then migration within islands. The best individual from

Table 1. The results for different topologies.

Topology	Average penalty	Min. penalty
Migration takes place every 5 seconds		
Ring	4072.3 ± 59.2	3992.2
Islands	4100.0 ± 69.8	3962.2
Toroidal grid	4077.0 ± 61.6	3955.3
Binary tree	4087.5 ± 51.6	4003.4
Migration takes place every 1 second		
Ring	4096.0 ± 68.4	3980.1
Islands	4107.4 ± 63.5	3993.5
No migration		
Unconnected	4083.9 ± 35.9	4000.6

the current population was chosen as the migrating solution. To obtain average results, each experiment was repeated 20 times.

The distributed algorithm execution was stopped when there was no progress during a period of 10 minutes in all sequential algorithm instances, i.e., no better solutions were found. The specific problem we experimented with was exam timetabling for the winter semester of the academic year 2010/2011 at our institution. The problem we solved was comprised of 135 courses with a total of 3455 students, which needed to be scheduled in 40 timeslots during a 10 day period. Each timeslot had capacity of 800 students and the number of exam taking was 17241, so the average occupancy was 54%.

The results of experiments are presented in Table 1. All obtained solutions satisfied hard constraints: there were no students scheduled to more than one exam at the same time, and there were no overcapacitated timeslots. The solutions only differed by the achieved quality as defined by (4) (see column 2 of the Table 1). It may be noted that the ring topology achieved the lowest average value of the penalty function, but these results must be taken with care because of the moderately large deviation. For a better comparison, it should be noted that repeatedly running a single algorithm until the 10-minute convergence period resulted with average quality of 4453.6 ± 295.8. Comparison of topologies by best solution over time is shown in Figure 2.

The time needed for convergence by different migration parameters is shown in Table 2. The fastest convergence was achieved by ring topology. It is encouraging that these results are the results with the smallest deviation.

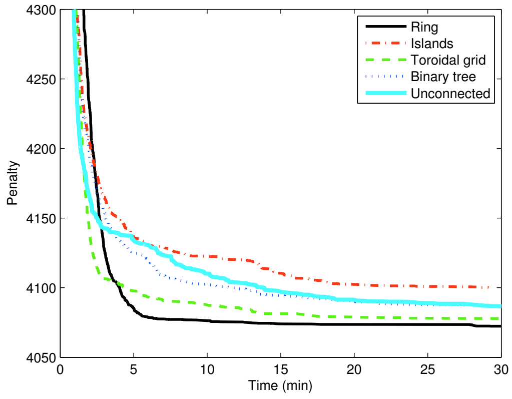

Figure 2. Average best solution by time for different topologies. Migration takes place every 5 seconds.

Table 2. The time needed for algorithms to converge (in minutes).

Topology	Average time	Best time
Migration takes place every 5 seconds		
Ring	9:31 ± 6:35	2:58
Islands	15:15 ± 8:04	3:51
Toroidal grid	12:50 ± 12:00	3:23
Binary tree	14:26 ± 7:26	2:05
Migration takes place every 1 second		
Ring	8:47 ± 6:54	1:47
Islands	10:39 ± 7:00	2:35
No migration		
Unconnected	18:28 ± 12:46	1:57

6. Conclusion

In this paper, we describe a distributed evolutionary framework for exam timetabling. Using the evolutionary computation, good-quality solutions can be found. As our experiments suggest, creating distributed multi-algorithm environment can produce even better results. By introducing migration of solutions, the execution time stability can be further improved.

Using the described distribution framework and the developed algorithms, we were able to find a satisfactory solutions for our institution. However, there is still a lot of work to be done with regard to the best topology selection and accompanying adequate migration parameters. For instance, island topology performed worse than unconnected topology which requires further investigation.

References

[1] M. Al-Betar, A. Khader, and T. Gani. A harmony search algorithm for university course timetabling. In *In 7th Intl. Conf. on the Practice and Theory of Automated Timetabling*. Springer Netherlands, 2008.

[2] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462, 2002.

[3] E. Alba and J.M. Troya. A survey of parallel distributed genetic algorithms. *Complexity*, 4(4):31–52, 1999.

[4] L.D. Davis and M. Mitchell. *Handbook of genetic algorithms*. Van Nostrand Reinhold, 1991.

[5] L.N. De Castro and F.J. Von Zuben. The clonal selection algorithm with engineering applications. *Proc. of the Genetic and Evolutionary Computation Conf., Las Vegas, Nevada, USA*, pages 36–37, 2000.

[6] K. A. De Jong. *Evolutionary Computation*. The MIT Press, 2006.

[7] M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, 2004.

[8] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*. WH Freeman and Company, San Francisco, Calif, 1979.

[9] Z.W. Geem. *Music-inspired harmony search algorithm: theory and applications*. Springer, 2009.

[10] M.R. Malim, A.T. Khader, and A. Mustafa. Artificial immune algorithms for university timetabling. In *The 6th Intl. Conf. on the Practice and Theory of Automated Timetabling*, pages 234–245. Masaryk University, 2006.

[11] E.G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009.

[12] D. De Werra. An introduction to timetabling. *European Journal Of Operational Research*, 19(2):151–162, 1985.

[13] M. Čupić, M. Golub, and D. Jakobović. Applying AI-techniques as help for faculty administration – a case study. *Central European Conference on Information and Intelligent Systems*, 2010.