International Journal of Electrical and Computer Engineering (IJECE)
Vol. 7, No. 2, April 2017, pp. 1096 – 1102
ISSN: 2088-8708, DOI: 10.11591/ijece.v7i2.pp1096-1102

Parallel Genetic Algorithms for University Scheduling Problem

Artan Berisha, Eliot Bytyçi, and Ardeshir Tërshnjaku
Department of Mathematics, Faculty of Mathematics and Natural Sciences, University of Prishtina, Kosovo

Article Info

Article history:
Received Nov 21, 2016
Revised Mar 9, 2017
Accepted Mar 27, 2017

Keyword:
Genetic algorithm
Parallel computing
Scheduling problem

ABSTRACT

University scheduling timetabling problem, falls into NP hard problems. Re-searchers have tried with many techniques to find the most suitable and fastest way for solving the problem. With the emergence of multi-core systems, the parallel implementation was considered for finding the solution. Our approaches attempt to combine several techniques in two algorithms: coarse grained algorithm and multi thread tournament algorithm. The results obtained from two algorithms are compared, using an algorithm evaluation function. Considering execution time, the coarse grained algorithm performed twice better than the multi thread algorithm.

Corresponding Author:
Eliot Bytyçi
Department of Mathematics, Faculty of Mathematics and Natural Sciences, University of Prishtina
rr. Nëna Terezë, p.n, Kosovo
eliot.bytyci@uni-pr.edu

1. INTRODUCTION

Every institution needs a schedule to represent their functionality. If the institution is small and not so complex, they may choose to draft their schedule manually. Otherwise, if the institution is bigger and needs to represent more complex relations between its functions, they may choose to use an application that generates the schedule automatically. Even then, some of the schedules are harder to create and manage than the others like nurse scheduling problem [1]. One that falls into the hardest ones, is also the university course timetabling problem, which has been proved decades ago, that it represents a NP problem [2].

University schedule needs to represent the location and time of all courses held in a semester, or year, in the university. By doing this, it achieves its main goal: information of the students and professors about the lectures to be held. Another important aspect of the schedule, is trying to accommodate the needs of the professors and students. So, first consideration is on finding the schedule that fits within university working hours and physical space available. Secondly, the schedule can be improved in order to accommodate needs of the professors and students such as not so long working hours, not subsequent lectures, taking into consideration breaks and even personal preferences such as lectures in the morning or in the afternoon.

In [3] [4] authors have described proposed usage of the distributed approach for solving the university timetabling problem, mainly due to the emergence of multi core systems. Furthermore, they argue that effectiveness of the parallel processing, need to be studied additionally. Taking this into the account, we have proposed two versions of roulette genetic algorithms: one based on islands and the other one on threads, to improve solution of the university timetabling problem. Therefore, one of the main aims of the paper is to use the parallel processing algorithm to obtain better results. That has been proven also in [5] and [6] but our solution represents first of all a fusion of a model used in [5] and another in [6] and in addition the tournament usage is done in multi thread. According to our experiments the tournament selection method allows for very quick solutions in light-constrained problems, using a fine-grained parallelism method, and roulette selection can be used in a coarse-grained parallel algorithm with slightly slower but better results in even harder problems Therefore, the advantage of our model is in cases of bigger population.

This paper is further organized as followed: section 2 describes the related work on the problem with emphasis on genetic algorithms used. Section 3 described in depth the problem and its constraints, while section 4 describes

algorithms used. In section 5 results from experiments are presented and section 6 represents the conclusion and future work.

2. RELATED WORK

Solving university schedule has been attempted in several ways, by using different kind of algorithms. In [7], authors have used particle swarm optimization algorithm with local search. Two types of the particle swarm optimization have been evaluated and the obtained results were satisfactory, both for teachers and classes. It should be emphasized that all conflicts have been handled, even though in the case when the local search was added, the satisfaction was better.

A specific case study of an Italian school timetable was presented by the authors of [3], by using genetic algorithms. Every single one of their solutions generated by the algorithm, have met the required constraints. The penalty for every constraint was set, from the ones with the highest importance, such as two teachers in the same class in the same hour) to the less significant ones such as teacher specific preference, which had a smaller penalty. They have also used mutation and crossover operators but with some restrictions due to the initialization phase.

In another paper [8], a parallel genetic algorithm was used in order to minimize one of the biggest flaws of genetic algorithm, time needed to obtain the result. According to the study, time was efficiently reduced using commercial shared memory multiprocessor. In the algorithm the chromosome is divided into n parts, every part consisting of m tuples. The operators used are mutation and crossover operator and the constraints presented, as the authors conclude themselves, are not the only ones to be used in a real case. Besides the usual usage of the operators for gene mutation, authors in [9] have used another type of operators called a bad genes operator that changes the genes that cause hard constraint violations. Such an operator is said to have improved the performance of algorithm. This approach has been considered also in our case.

Genetic algorithm was used to solve the school timetable problem [5], where roulette wheel was used for selection purpose. So, the best parents will have better probability to get selected and therefore allowing for better solutions to be achieved. This method was used in our case as well.

In one of latest papers [6] on the problem of school timetable, authors have used a parallel genetic algorithm in a high performance parallel computing machine created from personal computer hardware, Beowulf. Their fitness function is used to sum all the penalties, while the tournament was chosen to select parents, combined with number of operators such as mutation, crossover and reproduction to create a successor. The algorithm model was coarse-grain parallel model. Challenging the results from this paper, our algorithm will be tried in a personal computer with regular performance and comparing two algorithms: coarse-grained parallel and multi thread with tournament.

3. DESCRIPTION OF THE PROBLEM

The problem of the scheduling consists of several elements that comprise it, professors, student groups, classrooms, subjects taught by professors and related constraints between them. In the following, requirements, terms used and constraints, will be defined.

- A group represents certain number of students; which number depends on exercises to be held: numerical or laboratory. Numerical groups can be up to 30 students but not below 20 students, while laboratory groups can be up to 15 students but not below 10 students. For the lectures, the above mentioned groups, have to be merged
- A classroom consists of limited capacity of seats, where not every group can be accommodated in those classrooms. Besides that, classrooms can be usual for numerical exercises and lectures, while laboratories are usual for laboratory exercises
- A professor can be lecturer or teaching assistant
- A subject is a teaching course taught to a group by professor in certain time
- A period is timetable slot, which tells the maximum number of courses (both lectures and exercise) during working day
- A waiting period is a time slot which represents maximum waiting time between two consecutive lectures

Each schedule is defined in terms of professors, classrooms, students, period and waiting time. Therefore, several problems can arise such as the one described above regarding limited capacity of seats. Other problems, such

as overlapping of time for the same subject, same classroom, same professor and same group of students. Furthermore, the sequence (classroom, professor, group of students, subject) must be assigned a time and day. This requires fulfilment of many constraints, that can be divided into hard and soft ones. By hard constraints are defined all those mentioned above. In other hand, soft constraints, represent all other constraints that even when not fulfilled, do not affect the schedule operability.

3.1. Constraints used

As in [10], these constraints are divided in four main groups: Time, Courses, Subjects, and Resources. The Time is divided into time slots, considering it as number of sequences of nonnegative integer for number of time slots per day, and the day represents the day of the week. A Course it is a collection of events together with the group of students and subject. Subject are represented as Mathematics, Informatics and Resources are meant for students, teachers and classrooms. We divided constraints in two main groups: hard and soft. By hard we mean the requirements like there has to be no conflict in lecture time for professor. Otherwise with soft constraint we define requirements like waiting period. The list of hard and soft constraints is following:

Hard constraints:

- No conflict in lecture time for professors
- No conflict in lecture time for classrooms
- A group cannot be in two lectures at the same time
- Number of students cannot exceed the number of seats of a classroom
- Laboratory exercises should be held in laboratories.

Soft constraints:

- Not to have lectures and exercise of the same subject within the same day
- If there is no need for lectures to be held in laboratory, then the classroom mustnt be laboratory
- Waiting time for a professor should not exceed more than two hours
- Waiting time for group should not exceed more than two hours.

4. ALGORITHMS USED

Algorithms used are based on the basic roulette wheel selection. Initially it creates a population by generating individual instances randomly. Instances are examined and their values are assigned based on the heuristic function. Heuristic function return smaller values for better instances but in order to compute the cost, the value is inversed so that the best instances have the biggest value. The worst instance has value of 1. Obtained values are used as intervals for the roulette function, which assigns randomly individuals as parents. Since better instances have bigger values, the chance for creation of better individuals is greater. The newly created generation is based on the parents chosen and the crossover process.

4.1. Formal encoding of the real scheduling problem in terms of genetic paradigm constraints used

In terms of modelling, instances describe a whole individual schedule whereas genes show for a specific lecture time and classroom. Regarding the selection and the fitness functions, it should be noted that roulette wheel selection, implemented with whole integers, is used on all cases except on the asynchronous multi thread instance where the whole population is man-aged by different threads. Those threads dont synchronize between generations and thus must use tournament selection as a selection method that allows for asynchrony because it doesnt need to sort the whole population or find the sum of the entire populations fitness. The fitness function returns an integer, depending on how many constraints are violated. The returned integer is higher because its a simple sum of weighted constraint violations, where for each violation the sum is increased by P_h or P_s points, where P_h is the penalty for violating a hard constraint and P_s the penalty for violating a soft constraint. Because of the specific conflict elimination crossover used in this experiment, and because the fitness function of all individuals is measured well

before theyre crossed over, it is the fitness functions responsibility to also mark with an additional number each gene that has been found to represent a class that violates some constraint, the marker can be called conflict-marker. The same marker is used in the crossover procedure, wherein the higher the conflict mark, the less the chances of the gene are to be used in the child, and usually if a gene of a parent has a high conflict-marker it gets replaced with the other parents gene that represents the same lecture, but in a different time or place, thus hopefully eliminating the conflict much faster. Formally conflicts are presented in such a way that for every lecture that violates a constraint, a marker is used to represent the level of constraint that has been violated in the gene that represents that specific lecture. For hard constraints the marker gets a higher value than for soft constraints. A gene conflict represents lectures that do not fulfil any constraint.

4.2. Crossover

Crossover method for conflict elimination is based on uniform crossover [11] but with an additional condition which penalizes selection of genes with conflicts. This allows substitution of genes with conflict of one of the parents with the genes from the other parents, that have less or no conflicts. In this case, every gene of every parent is tried and compared. So, if the value of the conflict of a parent gene is lower than the value of the conflict of the other parent, then that gene is selected for the new individual. But, if the value of the conflict is the same then the gene of new the individual in that position is not changed for the sake of efficiency. The crossover operator can be expressed with the following pseudocode:

```
crossover (parent1, parent2):
child = copy(parent1)
for i = 0 to N do
if parent1.gene[i].conflict() ¿ parent2.gene[i].conflict() do
child.gene[i] = parent2.gene[i]
return child
```

Besides the crossover operator, algorithms use also another operator called mutation first fit.

4.3. Mutation first fit

Mutation first fit is used to fix the time and classes in which the lectures will be held, by using mutation, in order to achieve an acceptable timing and place and therefore improve the schedule and eventually achieve optimal schedule. To achieve the appropriate class for the lecture, the requirement for the lecture are taken into the account. If the lecture requests a bigger class, then it should be chosen from the classes that can accommodate that request. It is the same also in the case when a lab is needed for the lecture. But, there exists also the possibility that the class can be changed randomly in order not to be blocked in one class that is appropriate but doesnt have free time slots.

4.4. Coarse grained

Coarse grained algorithm or island algorithm, creates several instances of roulette based genetic algorithm, which are called islands because its population can communicate only through the managing algorithm. In every island, on every nth generation, best instance migrates in all other islands. This increases the chance that the population in general doesnt get stuck on local maximums. On the other hand, it allows the best schedules to become part of other is-lands but also allows islands to diverge considerably from each other.

4.5. Multi thread with tournaments

The total population with n individuals is divided and managed by k threads, with the responsibility over t = n/k individuals. Every thread manages a time interval of the population [t*i, t*(i+1)-1], where i the index of the thread. Again, the schedules are generated randomly and then crossover and mutation are applied to create the new individual. The best individual is removed from the replacement loop, because of elitism. The following pseudocode describes the multithread with tournament algorithm:

```
best = find best individual from populat. [t*i:t*(i+1)-1]
for every individ in population [t*i:t*(i+1)-1]:
if individual == best, go to next individual
```

choose parent1 with tournament with p=2 contenders from the overall population
choose parent2 with tournament with p=2 contenders from the overall population
child= crossover(parent1, parent2)
child.mutation(fraction OfGenesPerMutation)

4.6. Algorithm evaluation

In the end, in order to compare the results, we can quantify the performance of the algorithm, with the following formulae:

$$VP = (VT + 29) * P + T \tag{1}$$

Where VP represents value after penalization and VT, the value of timetable, P represents penalization and T time in seconds. So, the minimal value, which can be reached if all constraints are satisfied in the timetable, is -29. The value -29 is gained from five hard constraints, each represented by 5 points and four soft constraints, each represented by 1 point. The penalization when all constraints are satisfied, is only the time expressed in seconds, and the value of penalization increases for each point away from the minimal value. In other hand, to find the most accurate performance value of an algorithm parameter, one should take the sum of all performances of that algorithm (executed several times) and finding its average value. After what, it can be compared with other parameters average and the lowest value describes the best performance of that specific parameter.

5. RESULTS FROM THE EXPERIMENTS

The algorithms used for the university scheduling problem, where tried in a personal computer with the I5 processor and 4 Gb of RAM. Beforehand, several experiments to acknowledge the important parameters where performed. After that, important parameters were used to perform a set of experiments, which gave the following results.

5.1. Results from coarse grained algorithm

The coarse grained algorithm results are shown in Table 1. Population values chosen were 100, 200, 400, 600 and 800. For those values, after several tests we have obtained the best results. In all cases, algorithm has performed optimally finding the best value possible, which is -29 when all constraints are fulfilled. The best time performance has resulted in case of population 400.

Even though, the dataset used in our paper cannot be compared to any specific dataset used in other cases, for example in [5], we can argue on the results that as bigger the population, the time to achieve the best result is increased, similarly to the results in [5].

Table 1. Results from coarse grained algorithm

Population	Value	Generation	Gene mutation	Time in seconds
100	-29	873	60	39.569
200	-29	3397	60	248.241
400	-29	248	60	42.523
600	-29	330	40	86.926
800	-29	254	40	86.619

5.2. Results from multi thread with tournament algorithm

Also when dealing with the multi thread tournament algorithm (four threads) with period eight (maximum number of courses during working day) and class mutation 1/5, we have obtained results as shown in Table 2. Population values chosen were same as previous: 100, 200, 400, 600 and 800. Also multi thread algorithm in every case, has performed optimally finding the best value -29. The best result was obtained in the case of population 800.

In the case of multi thread tournament algorithm, a Crossover with conflict gene removal has been adapted from the graph colouring problems and has shown improvement in the timetable problem, compared to splice crossover and uniform crossover. The time to achieve the results is decreasing as well as the number of generations created.

Table 2. Results from multi thread tournament algorithm

Population	Value	Generation	Gene mutation	Time in seconds
100	-29	873	40	46.406
200	-29	254	40	27.277
400	-29	458	80	83.513
600	-29	177	40	48.587
800	-29	86	60	29.919

Table 3. Mean score for coarse grained and multi thread tournament algorithm

Population	Coarse Grained (Mean score in seconds)	Coarse Multi thread Tournamnet (Mean score in seconds)
100	527.60075	809.151
200	123.76925	335.29625
400	187.77375	508.424
600	351.903	443.14875
800	201.9715	148.04225

6. CONCLUSION AND FUTURE WORK

For every population we calculated the score value for both algorithms using equation (1) in Section 3, then we took the mean value overall cases, as presented in Table 3. Comparing results, we can see that Coarse grained algorithm is almost twice better then Multi thread Tournament algorithm. Although the multi thread computing should be faster. The Coarse grained algorithm fulfilled both hard and soft constraints in 60% of cases, while fulfilment of hard constraints and not all soft constraints is achieved within 95% of cases. The Multi thread Tournament algorithm fulfilled both hard and soft constraints in 45% of cases, while fulfilment of hard constraints and not all soft constraints is achieved within 65% of cases. Based on these findings and Table 3 we conclude that in this scheduling problem, Coarse Grained Algorithm is more effective and efficient than Multi Thread Tournament Algorithm. As previously concluded, the coarse grained algorithm was proved more effective and efficient due to the fact that it can use roulette selection and which cannot be used in fine grained, in which tournament can be used. Therefore, we have not used the obvious case of comparing coarse grained with fined grained through tournament, since it wouldnt make much sense if the roulette is better in more difficult problems. As well from our case, the roulette has often proven to find better solutions or didnt get blocked on local maximums. For future work, since the algorithm has unpredicted output we will use this property to generate keys for various cryptographic algorithms. This will speed up the encryption/decryption time also will increase security of these algorithms because of its pseudorandom output property. The efficiency and effectiveness will increase also by implementing these algorithms in its parallel versions as in CUDA and executing them in Graphics Processing Unit (GPU).

REFERENCES

[1] Maya Widyastiti, Amril Aman, Toni Bakhtiar. "Nurses Scheduling by Considering the Qualification using Integer Linear Programming. TELKOMNIKA, Vol.14, No.3, September 2016, pp. 933 940.
[2] Even, S.; Itai, A.; Shamir, A. On the Complexity of Timetable and Multi-Commodity Flow Problems. In Proceedings of the 16th IEEE Annual Symposium on Foundations of Computer Science, California, CA, USA, 1315 October 1975; pp. 184193.
[3] Colorni, Alberto, Marco Dorigo, and Vittorio Maniezzo. "Genetic algorithms: A new approach to the timetable problem." Combinatorial Optimization. Springer Berlin Heidelberg, 1992. 235-239.
[4] Yuliant Sibaroni, Fitriyani, Fhira Nhita. "The Optimal High Performance Computing Infrastructure for Solving High Complexity Problem." TELKOMNIKA Vol.14, No.4, December 2016, pp. 1545 1551.
[5] Fernández, Cristina, and Matilde Santos. "A non-standard genetic algorithm approach to solve constrained school timetabling problems." In Computer Aided Systems Theory-EUROCAST 2003, pp. 26-37. Springer Berlin Heidelberg, 2003.
[6] Šrndič, Nedim, Emir Pandžo, Mirza Dervisevič, and Samim Konjicija. "The application of a parallel genetic algorithm to timetabling of elementary school classes: a coarse grained approach." In Information, Communication and Automation Technologies, 2009. ICAT 2009. XXII International Symposium on, pp. 1-5. IEEE, 2009.
[7] Chen, Ruey-Maw, and Hsiao-Fang Shih. "Solving university course timetabling problems using constriction par-

ticle swarm optimization with local search." Algorithms 6.2 (2013): 227-244.
[8] Abramson, David, and J. Abela. A parallel genetic algorithm for solving the school time-tabling problem. Division of Information Technology, CSIRO, 1991.
[9] Fernandes, Carlos M., João Paulo Caldeira, Fernando Melicio, and Agostinho C. Rosa. "Evolutionary Algorithm for School Timetabling." In GECCO, p. 1777. 1999.
[10] Post, G., Ahmadi, S., Daskalaki, S., Kingston, J. H., Kyngas, J., Nurmi, C., Ranson, D., Ruizenaar, H. (2008). An XML format for benchmarks in high school timetabling. In Proceedings of the 7th international conference on the practice and theory of automated time-tabling (PATAT 2008), Montreal.
[11] Di Stefano, Calogero, and Andrea GB Tettamanzi. "An evolutionary algorithm for solving the school time-tabling problem." In Applications of Evolutionary Computing, pp. 452-462. Springer Berlin Heidelberg, 2001

ACKNOWLEDGEMENT

This research was supported by University of Prishtina.