Application of Parallel Genetic Algorithm for Exam Timetabling Problem

Shiburaj Pappu, Kiran T. Talele, Junaid Mandviwala

Abstract— Exam Timetabling Problems (ETTP) are a complex set of NP-Hard problems, solutions to which by using traditional methods may be impossible or time consuming. We describe an effective solution to solve this problem by using different form of Genetic Algorithms like Steady State Genetic algorithm (SSGA), Simple Genetic Algorithm (SGA) and Generation Genetic Algorithm (GGA) running in parallel in a distributed nature. The main drawback of using any variant of genetic algorithm is its convergence time to obtain optimal solutions. In this paper we propose and implement a parallel system for executing the genetic algorithms to yield optimal solution in less time.

Index Terms— Genetic Algorithm, Parallel Computing, Distributed Computing, Timetabling Problems, NP-Hard problems,Parallel Genetic Algorithm.

1 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.

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 genetic algorithm (PGA) for exam timetabling to improve both execution time and solution quality. A small number of strategies to build parallel algorithms have been proposed [1, 2]. But, they are based on different parallelism models and none of these strategies has addressed the exam timetabling problem. Our purposed PGA is based on distributing computations over different processors with limited amount of tree-based communication.

A GA is a meta-heuristic search technique which allows for large solution spaces to be non-deterministically searched in polynomial time, by applying evolutionary techniques from nature [3]. GAs use historical information to exploit the best solutions from previous searches, known as generations, along with random mutations to explore new regions of the solution space. In general, a GA repeats three steps (selection, crossover, and random mutations) as shown by the pseudo code in Fig. 1. Selection according to fitness is a source of exploitation, and crossover and random mutations promote exploration.

- *Shiburaj Pappu is currently pursuing Masters Degree from Sardar Patel Institute of Technology under Mumbai University, India. E-mail: shiburaj.pappu@gmail.com*
- *Kiran Talele is currently Asst. Professor at Sardar Patel Institute of Technology. E-mail: kttalele@spit.ac.in*
- *Junaid Mandviwala is currently pursuing Masters Degree from Thakur college of Engineering under Mumbai University, India. Email: junaidmandviwala@gmail.com*

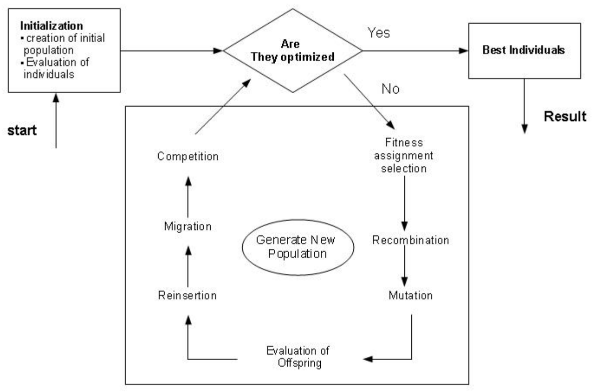

Fig. 1. General Scheme of Genetic Algorithm

2 Genetic Algorithms & its Variants

2.1 Genetic Algorithm

Genetic Algorithms (GA) are applicable to a wide variety of problems. In particular GA's have been very successful in obtaining near-optimal solutions to many different combinatorial optimization problems [4]. Genetic Algorithms are based on the principles of natural genetics and survival of the fittest. GA searches for solutions by emulating biological selection and reproduction. In a GA the parameters of the model to be optimized are encoded into a finite length string, usually a string of bits. Each parameter is represented by a portion of the string. The string is called a chromosome or individual, and each input variable (feature) is called a gene. Each string is given a measure of "fitness" by the fitness function sometimes called the objective or evaluation function. The fitness of a chromosome determines its ability to survive and reproduce offspring. The lowest fitness or the "weakest" chromosomes of the population are displaced by more fit chromosomes. GA is a robust search and optimization technique using probabilistic rules to evolve a population from one generation to the next. The transition rules going from one generation to the next are called genetic recombination operators, which include reproduction of new individual from fittest one, Crossover where portions of two chromosomes are exchanged and Mutation where it is performed infrequently. Crossover combines the fittest chromosomes and passes superior genes to the next generation, thus providing new points in the solution space. A new individual is created by altering one of the genes of an Individual. Mutation ensures the entire state space will eventually be searched and can lead the population out of local minima..

2.2 Simple Genetic Algorithm

GA offspring are saved in a separate pool until the pool size is reached. Then the children's pool replaces the parent's pool for the next generation for SGA. [5]

Algorithm 1: *Simple Genetic Algorithm*

```
t = 0
Initialize P (t)
Evaluate P (t)
while Termination condition is not satisfied do
  Selection P(t + 1) from P(t)
  Perform Crossover  on P(t + 1)
  Perform Mutation on P(t + 1)
  Evaluate P(t + 1)
  Replace P(t) with P(t + 1)
  t = t + 1
end while
```

2.3 Generation Genetic Algorithm

This GA produces a complete generation before discarding the old one. Solutions from new generation are now used as parents for the next generation and also some of the best parents of the previous generation are also copied to implement elitist. [6]

Algorithm 2: *Generation Genetic Algorithm*

```
t = 0
Initialize P (t)
Evaluate P (t)
while Termination condition is not satisfied do
  Selection P(t + 1) from P(t)
  Perform Crossover on P(t + 1)
  Perform Mutation on P(t + 1)
  Evaluate P(t + 1)
  Replace P(t) with P(t + 1) but keeping some of the
  best fit P(t)
  t = t + 1
end while
```

2.4 Steady State Genetic Algorithm

In a steady state GA the offspring and parents occupy the same pool. Each time an offspring is generated it is placed into the pool, and the weakest chromosome is dropped off the pool. [5]

Algorithm 3: Steady State Genetic Algorithm

```
t = 0
Initialize P (t)
Evaluate P (t)
while Termination condition is not satisfied do
  Selection P(t + 1) from P(t)
  Perform Crossover on P(t + 1)
  Perform Mutation on P(t + 1)
  Evaluate P(t + 1)
  Replacement(P(t),P(t + 1))
  t = t + 1
end while
```

3 Exam Timetabling Problem

We consider exam timetabling problem as follows:

A. Required Input
1. Courses to schedule.
2. Rooms/labs available.
3. For each course to schedule:
   a. Room restrictions (Size of Room)
   b. Number of sections of that course.
   c. Number of student in each course.
4. For each professor:
   a. Preferences of which courses he/she likes to teach

B. Fitness Calculation

A fitness function computes a single positive integer to represent how good the schedule is. The fitness of each individual in the population is calculated by adding positive value represent the violated constrain weight.

A perfect schedule would contain no conflicts in rooms or time conflicts, furthermore, would not violate that the student may take only two exams at a day since first and second exam must be held within specific period of time, in our case five day. For constrains, there are hard constraints that have to be repaired in order to get to a working schedule. Also, there is another type of constraints; soft constraints, such as a professor may have two or more exam at a day that may be violated.

Schedule would start with a fitness value of zero, and each instance found of a constraint violated would add a value associated with that constraint importance. After all constraints were tested the total value is connected to that schedule as its fitness. Then each schedule generated could be easily compared with other schedules simply by looking at which one is closer to zero. If a fitness value of zero is found, we can say that we have found a ideal schedule. The following constraints are used to calculate the fitness of the chromosome F. (1).

1. Hard Constraints
   a. Each student has at most two exams a day {X}

b. Each instance of a room hold one exam at a time.{Y}
c. Different sections for one course must be hold in the same schedule.{Z}

2. Soft Constraints
   a. For one course can be hold in different rooms.{U}
   b. Instructor may have two or more exams at a day.{V}

$$F = \alpha.X + \beta.Y + \eta.Z + \lambda.U + \Lambda.V \quad (1)$$

Where X, Y, Z, U, V are either 1 or 0 depending on whether the constraints are violated or not respectively. α, β, η, λ, Λ gives the penalty amount to be added to the fitness variable.

TABLE I
Penalty for violation of hard constraints

Constraints	*Penalty*
α for X	100
Β for Y	100
η for Z	100

TABLE II
Penalty for violation of soft constraints

Constraints	*Penalty*
λ for U	5
Λ for V	5

4 Proposed Algorithm (PGA)

Small populations could mean less genetic diversity while larger populations could bog down a computer due to extra memory storage. Thus the basic need of genetic algorithm is to have more diversity (large population sizes) which is achieved at the cost of time & memory. Thus most systems running any of the above mentioned variants of genetic algorithm will need to have ample amount a memory. Thus we propose a parallel genetic algorithm which are run simultaneously on many clients to decrease the convergence time. The following algorithm describes the PGA:

Algorithm 4: *PGA : Server Side Algorithm*

```
Initialize P
Evaluate P
Broadcast(P,SP)
  while New solution in P do
    if Broadcast time then
      Broadcast($P_{updated}$,SP)
    else if Client $P_{update}$ then
      Update($P_{client}$)
    end if
  end while
```

Algorithm 5: *PGA : Client Side Algorithm*

```
Recieve($P_{server}$, SP)
Update P with $P_{server}$ if better fitness
  while Termination condition is not satisfied do
    if Better P then
      Send(P) to server
    else
      Run(SGA/GGA/SSGA)
    end if
  end while
```

The algorithm starts with a single server along with many clients. The server generates the initial population with the provided input. This Population, P is then evaluated for the fitness function (1). Along with the P the setup parameters, SP is broadcasted through the network for the clients.

On the other side the clients are listening for this broadcast message from the server. When this message is received each clients update this data to their initial population if and only if the population is better than the existing ones. If the client is yet to start it uses the population received as it is and uses the setup parameter to begin the algorithm. Once updated the clients run one of the three variants of GA described earlier. This process continues till there is a better offspring produced. As soon as this is done the client enters into an update procedure and sends this data to the server.

The server is either waiting for the populations from other clients or is going to broadcast the updated population. As soon as the server receives the update message from any one of the clients it starts the update procedure where it replaces the weak population with the new ones. After every fixed duration the server broadcast this updated population to all the clients.

The process stops when the server finds that there are no new population formed and no update is required. The best chromosome amongst the final population is selected as the solution to the problem. The broadcast message sent at equal intervals to the clients serve two main purposes.

1) It reduces the chances of early convergence to the solution for local minima, as the update from the server helps the client in discovering new regions of the search space.
2) It also helps in having clients added to the system as and when needed. Since every message broadcast from the server has the new population as well as the setup parameters any client may be added at any point of time without the need to stop the process.

5 Experimental Results

TABLE III
Comparison of variants of Genetic algorithm with PGA

Parameters	*SGA*	*GGA*	*SSGA*	*PGA*
Iterations	500	500	390	345
Server + Clients	1	1	1	4
Population	8	8	8	8
Chromosome Size(bits)	112	112	112	112
Fitness (Avg.)	0	0	0	0
Avg. Convergence Time	53s	44s	22s	12s

6 Conclusion

We have proposed a parallel genetic 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 PGA produces good exam timetables with good parallel efficiency. Further work will compare PGA with other parallel 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 genetic algorithms.

References

[1] 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.

[2] 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.

[3] J. H. Holland, "*Adaptation in Natural and Artificial Systems*",Cambridge, MA, USA: MIT Press, 1992.

[4] W. AlSharafat, R Naoum, "Adaptive Framework For Network Intrusion Detection by using Genetic-Based Machine Learning", *IJCSNS International Journal of Computer Science and Network Security*. vol.9,No.4, 2009.

[5] AlSharafat, W.S.; AlSharafat, M.S.; , "Adaptive Steady State Genetic Algorithm for scheduling university exams", *Networking and Information Technology (ICNIT)*, 2010 International Conference on , vol., no., pp.70-74, 11-12 June 2010.

[6] Wong, T.; Cote, P.; Gely, P.; , "Final exam timetabling: a practical approach", *Electrical and Computer Engineering, 2002. IEEE CCECE 2002*. Canadian Conference on , vol.2, no., pp. 726- 731 vol.2, 2002.