Proceedings of the 5th International Conference of Control, Dynamic Systems, and Robotics (CDSR'18)
Niagara Falls, Canada – June 7 – 9, 2018
Paper No. 127
DOI: 10.11159/cdsr18.127

Timetabling with Three-Parent Genetic Algorithm: a Preliminary Study

Achini K. Herath and Dawn E. Wilkins
Department of Computer Science, University of Mississippi
University, MS 38677, USA

***Abstract* -** As institutes get larger in size and complexity, administrators face the problem of timetabling for their respective staff. The growing number of constraints involved due to expansion of the activities, make it difficult to produce a simple hand written timetable for the smooth running of a given institute. While a perfect solution is often not tractable, there are many techniques available to find optimal solutions to such problems. Among them, genetic algorithms (GA) seem to play a key role in many instances. A considerable amount of GA work has been carried out involving two parent populations. The present work which takes a multi-parent approach to the problem, seems to address the timetabling problem effectively.

***Keywords*:** Genetic Algorithm, Timetabling, Evolution, Three-Parent Crossover.

1. Introduction

The real-world NP-hard problems have no promising algorithms which could explore and exploit all possible solutions to find their global maxima. To develop effective optimization algorithms for such problems, it is necessary to maintain an appropriate balance between exploring new and unknown areas in the search space and exploiting a narrow area which could hopefully lead to a global optimum. Scientists have been developing heuristic and meta-heuristic algorithms to find quality solutions within a reasonable time frame to problems in science, engineering economics and business [1]. Genetic algorithms is one such method which can provide a reasonably good answer during an acceptable time frame, with limited information, by sacrificing completeness for speed. However, being population dependent, this technique is often handicapped with relatively long computational time. Genetic algorithms is a category of evolutionary algorithms which adopt a meta heuristic approach. It is a nature inspired searching and optimization technique involving crossover, mutation and selection which finds applications in providing high quality solutions to problems associated with optimization, scheduling, economics, bioinformatics etc. [2, 3]. This technique has been commonly used to solve timetabling problems.

In a previous study we applied genetic algorithm using a two-parent system to solve timetabling problems in education institutes [4]. Subsequently, studies were extended to improve the outcome using three-parents. As a preliminary step only hard constraints using a limited number of class rooms, staff members and students were considered to determine whether such an exercise could produce positive results. The results obtained were satisfactory enough for further investigation using more constraints to solve timetabling problems in higher educational institutes in the future.

2. Background

A timetable could be described as a tabulated document which contains all the information about the allocation of appropriate rooms to staff and students of a given Institute for well-defined time periods. For a small institute having a few staff members and students it is often a manual operation which is usually handled by a single person within a reasonable time frame. Composing a perfect timetable for a highly complex Institute such as a university is indeed a difficult task. A high level of optimization (global convergence) may be reached by way of mathematical programing. However, the high computational time makes such an approach infeasible [5]. The other option is to use approximation algorithms [6]. They are known to give satisfactory solutions at an acceptably low computational cost. One such method is metaheuristics, which includes genetic algorithm (GA) [7], discrete artificial bee colony (DABC) [8], tabu search (TS) [9], and simulated annealing (SA) [10]. GA with uniform three-parent crossover technique is used in the present work.

GA is an efficient optimization search technique which uses exploitation and exploration methods to find a global maximum [11]. The operators that provide exploration are the mutations and those that help the population to converge to better solutions (exploitation) are the cross-overs. Classical GA usually uses randomly selected chromosome pairs (parents) to produce two offspring which become part of the population. However, gathering more information using more than two parents could aid the process of evolution to bring forth much favourable changes to the next generation [12]. Thus, the multi-parent approach to problems has yielded favourable results. The experiments conducted by Eiben and van Kemenade [13] have shown significant gain with more than two parents especially with computationally less expensive diagonal crossover technique. Further, unimodal distribution crossover (UNDX) method which involves multi-parents has also shown impressive results [14].

The uniform three-parent crossover technique [15] involves the random selection of three parents followed by the comparison of each bit of the first parent with the corresponding one of the second parent. If they are the same it is being taken for the offspring. If it is different, the bit from the third parent is taken. Fig. 1 below shows an example of the logic involved.

Parent A	1	0	0	0	1	0	1	1
Parent B	1	0	0	1	0	1	1	0
Parent C	0	0	1	1	0	1	1	0
Offspring	1	0	0	1	0	1	1	0

Fig. 1: Three parent Crossover.

3. Method

3.1. Encoding

The total number of classes that needs to be scheduled can be obtained by summing the number of student groups multiplied by the number of modules each student group is enrolled in.

For each class scheduled by this application the following hard constraints will be considered.

- Classes can only be scheduled in free classrooms.
- A professor can only teach one class at any given time.
- Classrooms must be able to accommodate the respective student groups.

When encoding the class schedule for a given course class properties should be defined. They include the timeslot the class is scheduled, the ID of the professor and the classroom information.

In this investigation a numerical code is allocated to each timeslot, professor, and classroom. A chromosome can then be used to encode an array of integers to represent each class (Fig. 2).

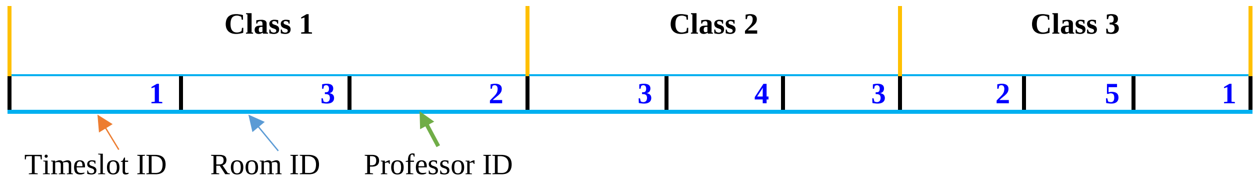

Fig. 2: Array is split into three to retrieve information for each class.

3.2. Initialization

A timetable needs to be built around the following criteria [16]: the rooms, professors, timeslots, courses/modules, and student groups.

The room class will contain information about the classroom, such as the room ID, room number and the capacity of students that can be accommodated. This class will accept a room Id, a room number and the capacity as well as provide methods to get the room's properties.

The timeslot class represents the day of the week and time that a class takes place. It holds the timeslot Id and the timeslot details.

The professor class accepts a professor ID and professor name properties. It also makes an allowance to retrieve this information as well.

A module class will store the information on the course modules. Each module can have several sections and groups of students taking the course at different times of the week with different professors. The module class accepts a module

ID, module code (“CSCI111” or “ENGL 101”), module name, an array of professor ID’s (professors who teach the module).

A group class will hold the information about the student groups. This class accepts group ID, a group size, module IDs the group is taking.

The Class class combines all of the above information. It will take a student group that takes a section of a module at a given timeslot, in a specific room with a specific professor.

A timetable class will capture all these objects and will co-ordinate how different constraints interact with each other. This class will also parse a chromosome and produce a candidate timetable to be evaluated and recorded. The timetable class serves two purposes. First, it is aware of all the available rooms, timeslots, professors, modules and groups. Second, it can read a chromosome, create a subset of classes from that chromosome, and help assess the fitness of the chromosome. This class consists of two important methods. The create class method and the calculation of clashes method. When creating a class for the timetable, an individual (chromosome) must be accepted, read and allocated information (timeslot, room, professor) to each class. As a result, the create class method uses a genetic algorithm and the subsequent chromosome to try different combinations of timeslots, rooms, and professors. The timetable class stores this information for future use. The clashes method then checks each one of the classes that has been built and counts the number of clashes. A clash is a hard constraint violation. Examples of clashes: room is too small, conflict with professor and timeslot, conflict with room and timeslot.

Clashes are used later in the genetic algorithm to calculate the fitness value. As each class is compared to all other classes, a “clash” is added, if any of the hard constraints are violated. The total number of clashes are calculated. This is then used to calculate the fitness value. The fitness value is the inverse of the number of clashes (1/clashes+1). If there are no clashes then the fitness value will be 1.

The main method in the timetable class creates the timetable and initializes it with all of the available courses, timeslots, rooms, modules, groups and professors. As a result, tournament selection and uniform crossover is used for the genetic algorithm.

A termination check is set up such that the decisive factors are the number of generations and the fitness factor. Combining both of these factors will terminate the genetic algorithm either after a certain number of generations or if it finds a valid solution. As such the fitness value depends on the number of broken constraints. As a result, the perfect solution will have a fitness value of 1. The number of generations is set to 1000.

3.3. Three Parent Crossover

A uniform three-parent crossover technique is used in the present work [11]. It involves the random selection of three-parents followed by the comparison of each bit of the first parent with the corresponding one of the second parent. If they are the same it is being taken for the offspring. Otherwise, the bit from the third parent is taken.

3.4. Mutation

Mutation is executed in such a way that a new random valid individual is created. The random individual created is used to select genes to copy into the individual to be mutated. This is called *uniform mutation* [17]. This technique ensures that all the mutated individuals are valid.

4. Results and Discussion

The hard constraints were tested using 20 trials to ensure that all the solutions obtained were valid. Four tests were carried out for the two-parent and the three-parent scheduler. The optimized solution for the timetable consisted of the following factors and their values. In Test 1, the population size was 100 with a mutation rate of 0.01, the number of elite individuals was 2 and the tournament size was 5. With these values the resulting timetable for the two parent scheduler had zero number of clashes with the fitness value of 1. The three-parent course scheduler results show that a total of ten clashes took place. The fitness value ranged from 1 to 0.25. However, the number of generations to reach an optimum timetable was achieved during the 14th (Table1) run, whereas in the two parent it took place at the 15th run (Table 2). In Test 2, the population size was increased from 100 to 2000. The mutation rate was kept constant at 0.01, the number of elite individuals was 2 and the tournament size was 5. The three-parent scheduler resulted in a decrease in the number of generations (Table 3) it takes to reach a solution when compared to the two parent scheduler (Table 4). In Test 3, the number of clashes were tested when the mutation rate was changed from 0.01 to 0.20. The three-parent scheduler resulted in only three clashes (Table 5), whereas the two-parent scheduler (Table 6) showed total of 74 clashes. Finally, in Test4, mutation rate was tested with the fitness value. The results showed that the three-parent scheduler (Table 7) kept a constant fitness during the initial 10 runs, and then rapidly decreased after the 21st run. However, Table 8 shows that the fitness value decreases rapidly. All four tests shows that the three-parent scheduler performed better than the two-parent.

4.1. Architectural Specifications

JAVA version 8 was used on a 6th Generation Intel(R) Core (TM) i7 Quad Core (6M Cache, up to 3.5 GHz) 16GB Dual Channel DDR4 256GB PCle Solid State Drive with NVIDIA GTX960M 2GB DDR5 for the design of the 3-Parent Course Scheduler.

4.2. Performance Analysis

Table 1: **(**Test 1): The number of generations the three-parent course scheduler takes to reach an optimal solution was tested.

Number of runs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Generations	816	427	51	1000	195	411	1000	1000	1000	869	972	1000	1000	34	1000	453	283	1000	893	77
Fitness value	1	1	1	0.5	0	0	1	1	3	0	0	1	1	0	1	0	0	1	0	0
Number of clashes	0	0	0	1	0	0	1	1	3	0	0	1	1	0	1	0	0	1	0	0

Table 1 shows that clashes occurred nine times during the 20 runs of the system.

The optimized solution for the three-parent course scheduler/timetable was obtained with a population size of 100, a mutation rate of 0.01, the number of elite individuals was 2 and the tournament size was 5. With these values the resulting timetable showed clashes occurring ten times. The fitness value ranged between 1 (where no clashes occurred) and 0.25 (where three clashes occurred). The minimum number of generations was taken during the 14th run of the program. The number of generations taken to obtain a timetable with a fitness value of 1 was 34. Fig. 3 is the bar diagram based on Table 1.

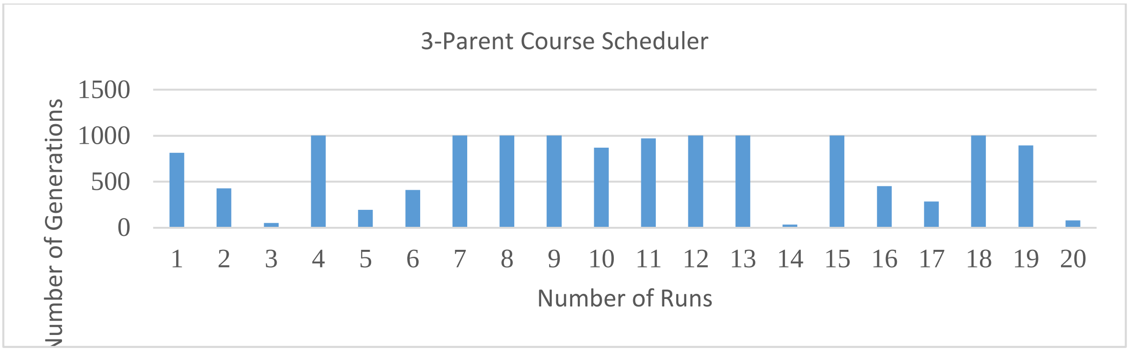

Fig. 3: The number of generations the three-Parent Course Scheduler takes to reach an optimal solution. The minimum number of generations was taken during the 14th run of the program.

Table 2: (Test 1): (two-parent): The number of generations the two-parent course scheduler takes to reach an optimal solution was tested.

Runs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Generations	105	96	47	54	56	41	50	91	39	47	44	48	67	45	37	63	51	49	42	60

The results show 0 clashes occurred during the 20 runs of the system and the fitness value was 1.

The optimized solution for the three-parent course scheduler/timetable was obtained with a population size of 100, a mutation rate of 0.01, the number of elite individuals was 2 and the tournament size was 5. With these values the resulting timetable showed no clashes. The fitness value was 1. The minimum number of generations was taken during the 15th run of the program. The number of generations taken to obtain a timetable with a fitness value of 1 was 37. Fig. 4 is the bar diagram based on Table 2.

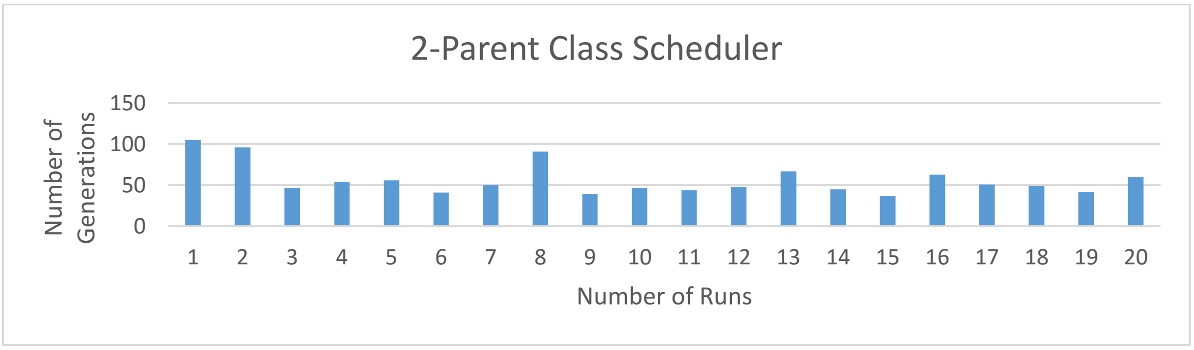

Fig. 4: The number of generations the two-parent Course Scheduler takes to reach an optimal solution. The minimum number of generations was taken during the 15th run of the program.

The two-parent scheduler took longer and three extra generations in order to give a final class schedule. So in this instance, the 3 parent scheduler performed better when compared to the two parent scheduler.

Table 3: **(**Test 2): Population size vs Number of generations for the three-parent scheduler.

Population size	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
Generations	300	115	61	10	20	13	11	10	12	20	12	11	11	11	11	15	10	14	12	12
Clashes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3 shows the increase in population size with the number of generations it takes to reach a solution. The mutation rate remains constant at 0.01, number of elite individuals was 2 and the tournament size remains 5. The resulting data shows a steady decrease in the number of generations that takes to reach a valid solution. Fig. 5 is the bar chart for Table 3.

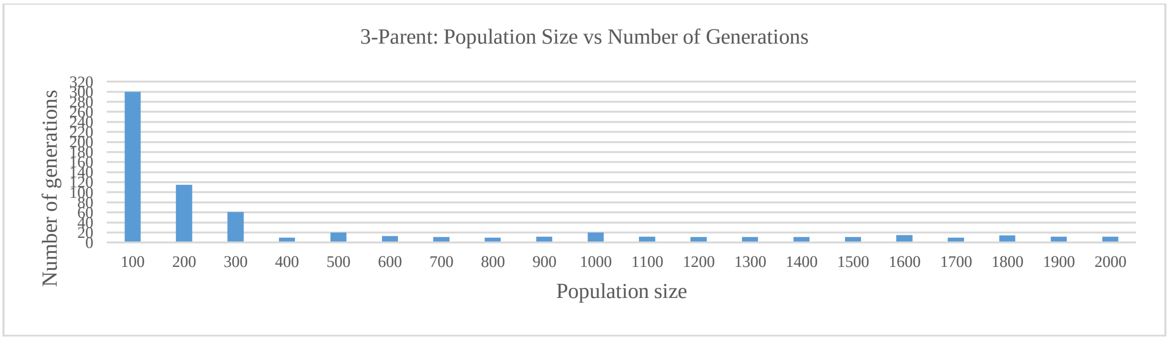

Fig. 5: (three-parent): Population size vs Number of generations. The minimum number of generations taken to reach a solution occurs when the population size is increased to 400. The number of generations is 10.

Table 4: **(**Test 2): (two-parent) Population size vs Number of generations for the two-parent scheduler.

Population size	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
Generations	61	55	69	63	71	79	77	84	90	100	88	84	40	100	105	111	88	78	116	68
Clashes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o

Table 4 shows the increase in population size from 100 to a 2000, with the number of generations. The resulting data shows that the number of generations that takes to reach a valid solution remains high when compared to the three-parent results. Fig. 6 displays the bar chart for the data in Table 4.

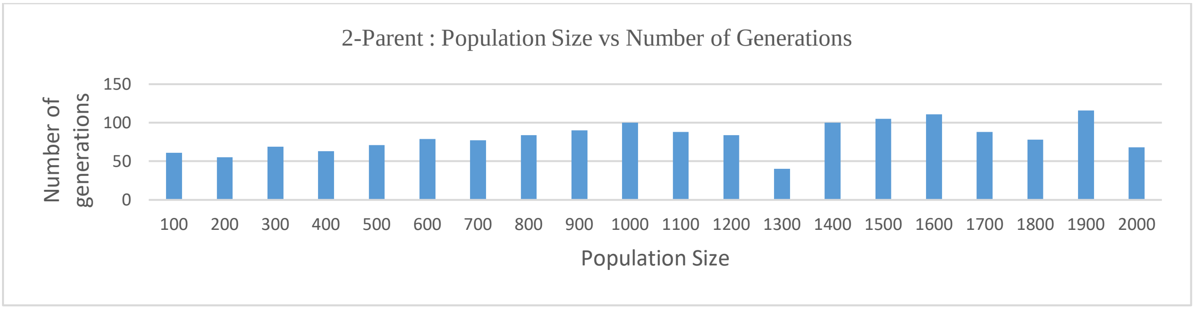

Fig. 6: (two-parent): Population size vs Number of generations.

The minimum number of generations taken to reach a solution occurs when the population size is increased to 1300. The number of generations taken to reach this is 40. This is 4 times larger than using the 3 parent scheduler.

The three-parent course scheduler has a larger search area. Therefore, with the increase in population size, the three-parent scheduler shows that a lesser number of generations are needed to obtain a solution. The average number of generations for the three-parent scheduler to reach a solution after 20 runs is 34.55. The two-parent scheduler takes an average of 81.35 generations to reach a solution after the 20 runs. The three-parent course scheduler shows better results for this test.

Table 5: (Test 3): Mutation rate vs Number of clashes for the three-parent scheduler.

Mutation rate	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.20
Clashes	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0

Increase in mutation rate shows a steady decrease in the occurrence of the number of clashes. The mutation rate was increased from 0.01 to 0.20, population size was kept at 100. The number of elite individuals was 2 and the tournament size was 5. With these values the resulting timetable showed 3 clashes. The first clash occurred at the very beginning when the mutation rate was 0.01, the second occurred at 0.08 and the third at 0.16. Fig. 7 displays the bar chart for the data in Table 5.

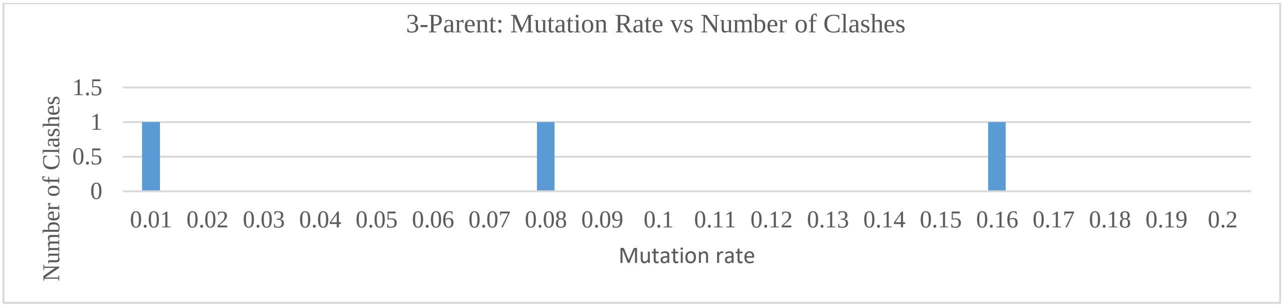

Fig. 7: (three-parent): Mutation rate vs Number of clashes. Three clashes occur when the mutation rate was increased from 0.01 to 0.20. The clashes takes place when the mutation rate was 0.01, 0.08 and 0.16.

Table 6: (Test 3): Mutation rate vs Number of clashes for the two-parent scheduler.

Mutation rate	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.20	0.21
Clashes	0	0	0	1	0	2	1	2	3	3	4	4	6	6	6	5	4	6	8	7	7

Increase in the mutation rate shows an increase in the occurrence of the number of clashes. The mutation rate was increased from 0.01 to 0.21 and population size was kept at100, the number of elite individuals was 2 and the tournament size was 5. With these values the resulting timetable showed 16 clashes occurring at regular intervals. Clashes fluctuated between a minimum of 0 at a mutation rate of 0.01, 0.02, 0.03 and 0.05. The maximum of 8 clashes occurred when the mutation rate was at 0.19. Fig. 8 displays the bar chart for the data in Table 6.

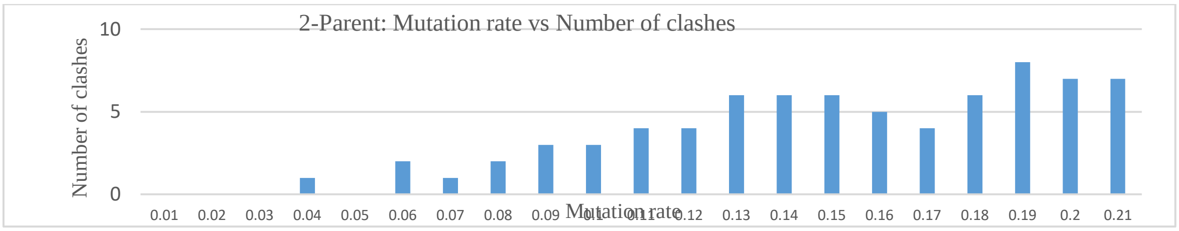

Fig. 8: (2-parent): Mutation rate vs Number of clashes. Seventeen clashes occurred when the mutation rate was increased from 0.01 to 0.21. The clashes occurred when the mutation rate was 0.04, and at the range 0.06 – 0.21.

Test 3 shows that the number of clashes in three-parent scheduler was far less (3 clashes) when compared to the two-parent scheduler (17 clashes). The wider search area when the three-parent technique is applied to the scheduler, guarantees that few clashes occur, even when more diversity is introduced to the population.

Table 7 (Test 4): Mutation rate vs Fitness value for the three-parent scheduler.

Mutation rate	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.20
Fitness	1	1	1	1	1	1	1	1	1	1	0.5	1	1	1	0.5	1	1	1	0.5	0.5

Mutation rate	0.21	0.22	0.23	0.24	0.25	0.26	0.27	0.28	0.29	0.30
Fitness	1	0.33	0.33	0.33	0.25	0.25	0.25	0.25	0.25	0.33

Further tests show that when the mutation rate was increased to 0.30 the fitness value decreases from 1 to 0.33 (Table 7).

Sudden decrease in fitness occurred at the mutation rate of 0.19. Further decrease in the mutation rate causes fitness to decrease at a regular interval.

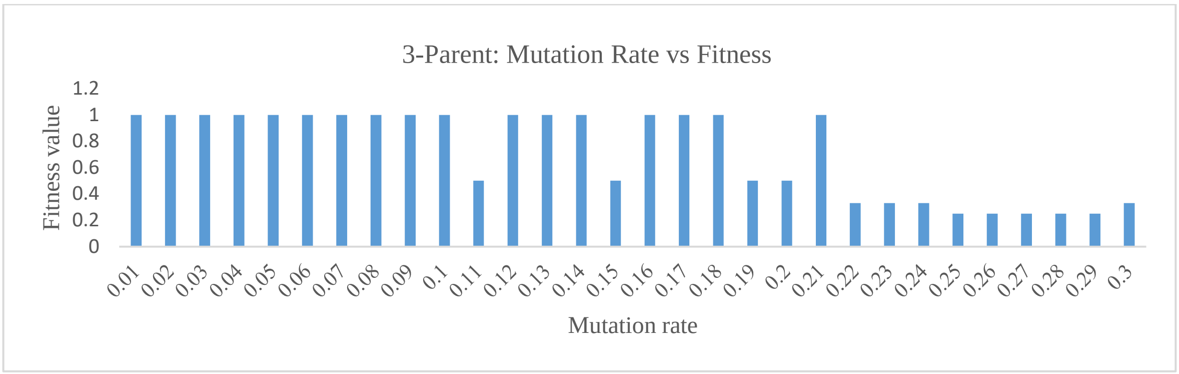

Fig. 9: (three-parent): Mutation Rate vs Fitness. The fitness value remains 1 when the mutation rate is 0.01 – 0.10, 0.12-0.14, 0.16-0.18 and 0.21. Afterwards, the fitness value of the resulting schedule shows a rapid and regular decrease.

Table 8 (Test 4): Mutation rate vs Fitness value for the two-parent scheduler.

Mutation	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.20	0.21
Fitness	1	1	1	0.5	1	0.33	0.5	0.33	0.25	0.25	0.2	0.2	0.14	0.14	0.14	0.16	0.2	0.14	0.11	0.12	0.12

The two-parent scheduler was tested with a mutation rate range between 0.01and 0.21. Even though, mutation rates 0.01, 0.02, 0.03 and 0.05 showed a fitness value of 1, the decrease in fitness value started at a lower mutation rate when compared to that of the 3-parent scheduler. Fig.10 shows the bar chart for Table 8.

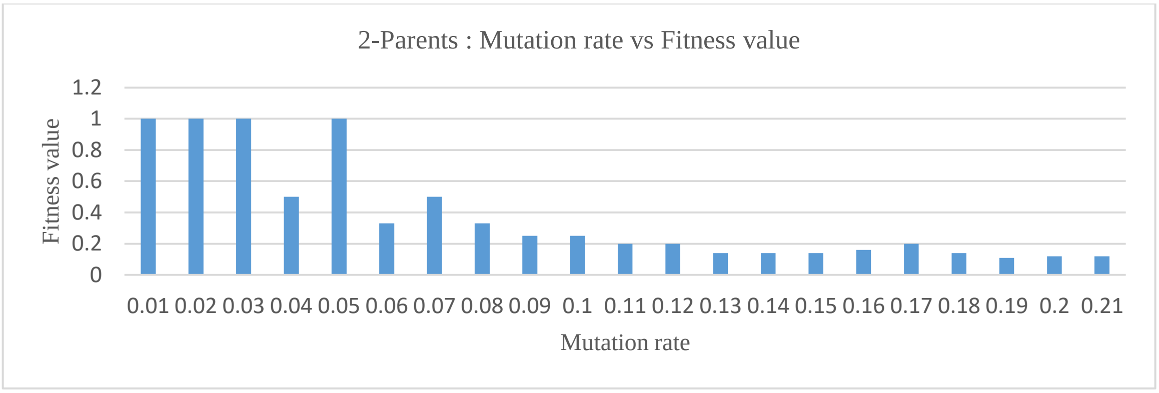

Fig. 10: (two-parent): Mutation Rate vs Fitness. The fitness value remains 1 when the mutation rate is in the range 0.01-0.03 and at 0.05. Afterwards, the fitness value of the resulting schedule shows a rapid and regular decrease.

Test 4 results showed that the three-parent scheduler change in fitness was far better when compared to the two-parent scheduler. The higher search area when the three-parent technique is applied to the scheduler, guarantees that the fitness value is better as more diversity is introduced into the population by way of increasing the mutation rate. .

5. Conclusion

When compared to the two-parent course scheduler [4], the three-parent shows the following advantages. An increase in the population size shows a steady decrease in the number of generations that it takes to reach a valid solution. There were no clashes during this run. In addition, when the mutation rate is increased, the three-parent scheduler shows only a little change in the fitness value, for a considerable number of runs of the system. In comparison, the two-parent scheduler shows a significant effect on the fitness factor right from the beginning, with increase in mutation rate. The promising results of this preliminary investigation involving three-parents could be used in the future to extend this application to find solutions to various constrains in timetabling problems.

References

[1] W. J. Suh, C. S. Park and D. W. Kim, "Heuristic vs. meta-heuristic optimization for energy performance of a post office building," in *Proceedings of the 12th conference of international building performance simulation association*, Sydney, Australia: IBPSA. pp. 704-711, 2011.

[2] H. M. Pandey, A. Choudhary and D. Mehrotra, "A comparative review of approaches to prevent premature convergence in GA," *Applied Soft Comp.*, vol. 24, pp. 1047-1077, 2014.

[3] B. L. Miller and D. E. Goldberg, Genetic Algorithms, Tournament Selection, and the Effects of Noise," *Complex Systems*, vol. 9, pp. 193-212, 1995.

[4] A. K. Herath, "Genetic algorithm for university," M.S. Thesis, Dept. Comp. Inform, Sc., Univ. Mississippi, USA.

[5] N. H. Moin, O. Chung Sin and M. Omar, "Hybrid Genetic Algorithm with Multiparents Crossover for Job Shop Scheduling Problems," *Math. Probl. Eng.*, vol. 12, pp. 00-00, 2015(210680).

[6] A. Jones, L. C. Rabelo, and A. T. Sharawi, "Survey of job shop scheduling techniques," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, New York, NY, USA, 1999.

[7] Z. Yaqin, L. Beizhi, and W. Lv, "Study on job-shop scheduling with multi-objectives based on genetic algorithms," in the *International Conference on Computer Application and System Modeling (ICCASM '10)*, pp. v10294–v10298, IEEE, 2010.

[8] M. Yin, X. Li, and J. Zhou, "An efficient job shop scheduling algorithm based on artificial bee colony," *Scientific Research and Essays*, vol. 6, no. 12, 2578-2596, 2011.

[9] C. Zhang, P. Li, Z. Guan, and Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job with a new neighborhood structure for the job shop scheduling problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3229-3242, 2007.

[10] S. Z. Song, J. J. Ren, and J. X. Fan, “Improved simulated annealing algorithm used for job shop scheduling problems,” in *Advances in Electrical Engineering and Automation*, vol. 139, pp.17-25, Springer, Berlin, Germany, 2012.

[11] W. Wan, J. B. Birch, “An Improved Hybrid Genetic Algorithm with a New Local Search Procedure,” *J. Appl. Math.*, 2013.

[12] R. Patel, M. M. Raghuwanshi, “Multi-objective Optimization Using Multi Parent CrossoverOperators,” *J. Journal of Emerging Trends in Computing and Information Sciences*, vol. 2, no. 2, pp. 33-39, 2010.

[13] A. E. Eiben, C. H.M. van Kemenade, (1995) Performance of multi-parent crossover operators on numerical function optimization problems. Technical Report TR-95-33, Leiden University. [Online]. Available: http://www.liacs.nl/TechRep/1995/

[14] S. M. Elsayed, R. A. Sarker and D. L. Essam. “2011b. GA with a new multi-parent crossover for solving IEEE-CEC2011 competition problems,” in *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 1034-1040, 2011.

[15] T. Manning, R. D. Sleator and P. Walsh, “Naturally selecting solutions: the use of genetic algorithms in bioinformatics,” *Bioengineered*, vol. 4, no. 5, pp. 266-278, 2013.

[16] M. Norberciak, "Universal Method for Timetable Timetable Construction based on Evolutionary Approach," *World Academy of Science Engineering and Technology,* pp. 91-96, 2006.

[17] N. Soni and T. Kumar, “Study of Various Mutation Operators in Genetic Algorithms,” *International Journal of Computer Science and Information Technologies (IJCSIT),* vol. 5, no. 3, pp. 4519-4521, 2014.