LJAST
Libyan Journal of Applied Science and Technology
مجلة ليبيا للعلوم التطبيقية والتقنية

Volume 12 Issue 01
June 2024
ISSN 2958-6119

Classes Scheduler sing Genetic Algorithm

Murad AlBarki[1], Ahmed Alardawi[2], Abobakr Aboshgifa[3], Nabil Belhaj[4]
[1,2] The College of Computer Technology Tripoli,Libya
Ahmed.Alardawi@cctt.edu.ly
[3,4] Research Department ,The Libyan higher technical center for training and production،
Tripoli, Libya

Received 28 May 2024; revised 30 May 2024; accepted 01 June 2024

Abstract

In the ever-evolving of software development, the surge in artificial intelligence (AI) and deep learning technologies has been nothing short of astonishing. As the world bears witness to this transformative era, staying abreast of cutting-edge problem-solving methodologies becomes paramount. To master these techniques, practical experimentation is essential. Among these techniques, the genetic algorithm stands as a venerable tool for addressing complex problems. Its longevity in the field attests to its efficacy.

Genetic algorithms find their niche in scenarios that demand iterative sorting under diverse and multifaceted conditions. Whether optimizing schedules, streamlining resource allocation, or enhancing decision-making processes, these algorithms offer a powerful framework. Their versatility extends across various domains, making them excellent tools for modern problem solvers.

The study of genetic algorithms involves practical validation. By subjecting them to rigorous testing and empirical study, to uncover their limits, capabilities, and potential applications. As delve deeper into this field, novel solutions will be unlocked and pave the way for further advancements.

The primary focus of this research paper is to demonstrate genetic algorithm principles, specifically in the context of optimizing timetable schedules through the use of Evolutionary Algorithms (EAs). The study proceeds by presenting a developed (.NET) application that employs the prescribed methodology of numerous genetic algorithms to seek the most optimal solution for scheduling college classes, subject to multiple constraints. Subsequently, a comparative analysis of the results obtained from two selected algorithms is conducted.

Keywords : Evolutionary Algorithm, Genetic Algorithms, Timetable Schedules ,Scheduling Model.

1. *Introduction*

In the recent years, metaheuristic algorithms are used to solve real-life complex problems arising from different fields such as economics, engineering, politics, management, and engineering. Intensification and diversification are the key elements of metaheuristic algorithm. The proper balance between these elements is required to solve the real-life problem in an effective manner. Most of metaheuristic algorithms are inspired from biological evolution process, swarm behavior, and physics' law. As is known from experience, the manually designing and preparation of time schedules that are best suited to the purposes for which they were prepared is a difficult process that takes a huge amount of time and energy. Worse still, the process has to be recalculated

for each input individually each time the parameters change.
Fortunately, artificial intelligence (AI) can streamline this process, automating the heavy lifting and providing efficient solutions, and Genetic Algorithm is one of the most commonly used method in the meantime.
The Genetic Algorithm (GA) is a search-based optimization technique rooted in principles inspired by genetics and natural selection. GAs are commonly used to discover optimal or nearly optimal solutions for challenging problems that would otherwise require an extensive amount of time to solve manually. Their applications span various domains, including optimization problems, scientific research, and machine learning[1].
The objectives of this paper are:

- Demonstrate genetic algorithm principles.
- Design and develop an app to use multiple genetic algorithm approaches to schedule classes of an education institution with different constrains.
- Implement the web app and compare the results between two chosen methods.

2. *Genetic Algorithms*

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA).

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

The (GA) is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The method repeatedly modifies a population of individual solutions[2].

2.1 *Notion of Natural Selection*

The process of natural selection starts with the selection of the fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

This notion can be applied for a search problem. A set of solutions for problem is considered and selected the set of best ones out of them[1].

2.2 *Phases of Genetic Algorithm*

There are six phases considered in a genetic algorithm:

Volume 12 Issue 01
June 2024
ISSN 2958-6119

- Initial population
- Fitness function
- Selection
- Crossover
- Mutation
- Termination

2.2.1 Initial Population

The process is initiated with a set of individuals which is called a Population. Each individual is a solution to the problem needs to be solved, those individuals may or may not include the optimal values.

An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution).

In genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). This process is referred to as encoding the genes in a chromosome. Figure (1) shows an example of Population, Chromosomes and Genes.

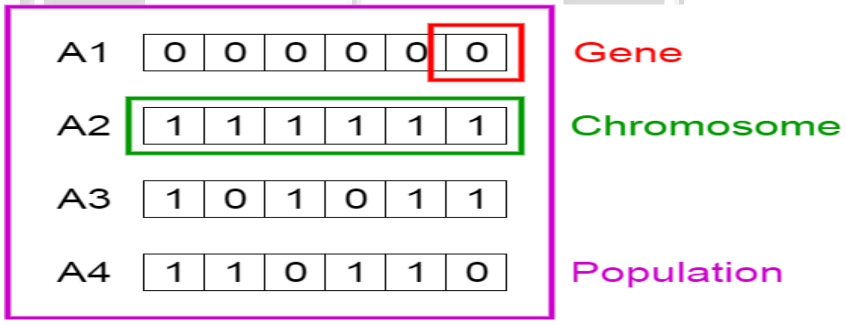


Figure (1) Population, Chromosomes and Genes.

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found[2].

2.2.2 Fitness Function

Fitness Function (also known as the Evaluation Function) evaluates how close a given solution is to the optimum solution of the desired problem. It determines the ability of an individual to compete with other individuals. It gives a fitness score to each individual.

The probability that an individual will be selected for reproduction is based on its fitness score.

There are generic requirements should be satisfied by any fitness function:

- Clearly defined.
- Efficiently implemented.
- Quantitatively measured .
- Generate intuitive results.

Each problem has its own fitness function. However, certain functions have been adopted by data scientists regarding certain types of problems. Coming up with a fitness function is the hardest part when it comes to formulating a problem using genetic algorithms.

Typically, for classification tasks where supervised learning is used, error measures such as Euclidean distance and Manhattan distance have been widely used as the fitness function.

For optimization problems, basic functions such as sum of a set of calculated parameters related to the problem domain can be used as the fitness function. A very famous scenario where genetic algorithms can be used is the process of making timetables or timetable scheduling.

Consider a weekly timetable being created for a particular batch of college classes. Classes must be arranged, and a timetable developed, to ensure no clashes occur. The task, therefore, is to search for the optimum timetable schedule.

Since no collisions among classes should occur, minimizing the number of students with class conflicts is essential. The fitness function can be formulated as the inverse of the number of students with class conflicts. The fewer the students with class conflicts, the more fit the class schedule is[3].

2.2.3 Selection

The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation.

Two pairs of individuals (parents) are selected based on their fitness scores, individuals with high fitness have more chance to be selected for reproduction.

During each successive generation, a portion of the existing population is selected to reproduce for a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g., computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used[4].

2.2.4 Crossover

Also called recombination, is a genetic operator used to combine the genetic information of two parents to generate new offspring. It is one way to stochastically generate new solutions from an existing population and is analogous to the crossover that happens during sexual reproduction in biology. Solutions can also be generated by cloning an

existing solution, which is analogous to asexual reproduction.
Newly generated solutions are typically mutated before being added to the population.

Different algorithms in evolutionary computation may use different data structures to store genetic information, and each genetic representation can be recombined with different crossover operators. Typical data structures that can be recombined with crossover are bit arrays, vectors of real numbers, or trees.

Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes. For example, consider the crossover point to be 3 as shown in Figure (2).

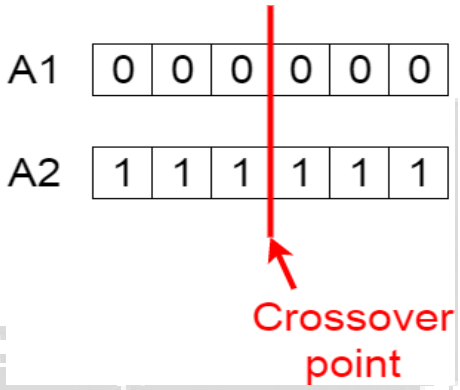


Figure (2) Crossover point.

Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached. Figure (3) shows the process of exchanging genes among parents.

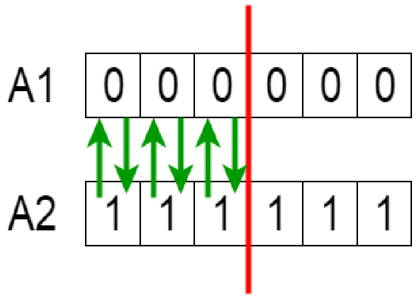


Figure (3) Exchanging genes among parents.

The new offspring are added to the population as seen in Figure (4).

Volume 12 Issue 01
June 2024
ISSN 2958-6119

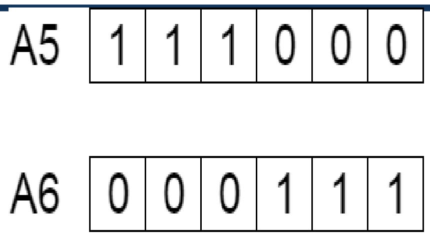


Figure (4) New offspring.

Traditional genetic algorithms store genetic information in a chromosome represented by a bit array. Crossover methods for bit arrays are popular and an illustrative example of genetic recombination.

- **One-point crossover**

A point on both parents' chromosomes is picked randomly and designated a 'crossover point'. Bits to the right of that point are swapped between the two parent chromosomes. This results in two offspring, each carrying some genetic information from both parents. See Figure (5)

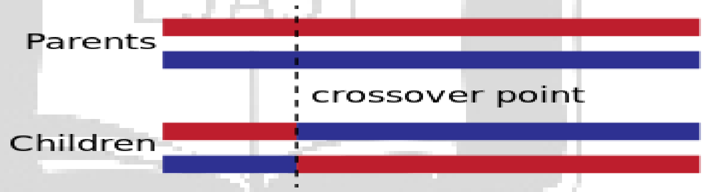


Figure (5) One Point Crossover.

- **Two-point and k-point crossover**

In two-point crossover, two crossover points are picked randomly from the parent chromosomes. The bits in between the two points are swapped between the parent's organisms.

Two-point crossover is equivalent to performing two single-point crossovers with different crossover points. This strategy can be generalized to k-point crossover for any positive integer k, picking k crossover points. See Figure (6).

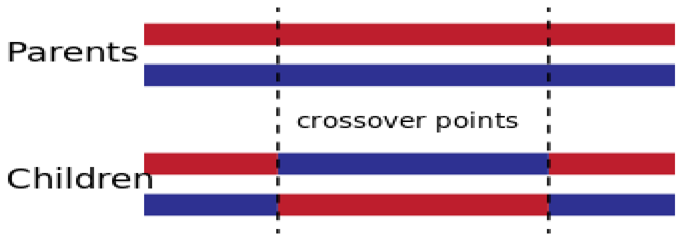


Figure (6) Two Points Crossover.

- **Uniform crossover**

In uniform crossover, typically, each bit is chosen from either parent with equal probability.[6] Other mixing ratios are sometimes used, resulting in offspring which inherit more genetic information from one parent than the other. In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. It's also possible to bias the coin to one parent, to have more genetic material in the child from that parent[5].

2.2.5 Mutation

In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped. See Figure (7).

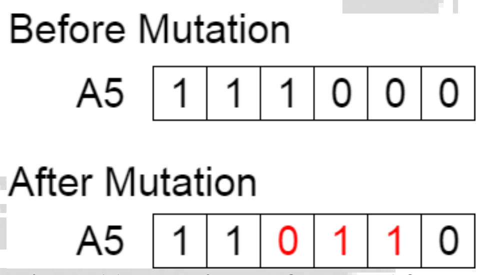


Figure (7) Mutation: Before and After.

Mutation occurs to maintain diversity within the population and prevent premature convergence. Mutation is a genetic operator used to maintain genetic diversity of the chromosomes of a population of a genetic or, more generally, an evolutionary algorithm (EA). It is analogous to biological mutation.

The classic example of a mutation operator of a binary coded genetic algorithm (GA) involves a probability that an arbitrary bit in a genetic sequence will be flipped from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be flipped. This mutation procedure, based on the biological point mutation, is called single point mutation. Other types of mutation

operators are commonly used for representations other than binary,
such as floating-point encodings or representations for combinatorial problems.

The purpose of mutation in EAs is to introduce diversity into the sampled population. Mutation operators are used in an attempt to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping convergence to the global optimum. This reasoning also leads most EAs to avoid only taking the fittest of the population in generating the next generation, but rather selecting a random (or semi-random) set with a weighting toward those that are fitter.

The following requirements apply to all mutation operators used in an EA:

- Every point in the search space must be reachable by one or more mutations.
- There must be no preference for parts or directions in the search space (no drift).
- Small mutations should be more probable than large ones.
- For different genome types, different mutation types are suitable.

  - **Bit string mutation**

The mutation of bit strings ensues through bit flips at random positions.

Example:

$$\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ & & & \downarrow & & & \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

The probability of a mutation of a bit is 1/L, where L is the length of the binary vector. Thus, a mutation rate of 1 per mutation and individual selected for mutation is reached. [6]

2.2.6 Termination

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to the problem[7].

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria.
- Fixed number of generations reached.
- Allocated budget reached (computation time/money).
- The highest-ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results.
- Manual inspection.
- Combinations of the above[7].

3. Project Architecture and Implementation

An application has been developed using C# programming language to accept parameters related to the classes and restrictions then use one of the genetic algorithms to provide a schedule of the classes with the optimal solution where the less conflicts happen.

3.1 *Application Architecture*

The application is a one solution project that contain three components, the genetic algorithms component, the models component and the main console application component, as shown in table (1).

Table (1) Application Components.

Algorithms	Console	Models
The methods and algorithms.	The main app, the data used as an input, the output in two formats, HTML and JSON.	The models used in the project.

3.1.1 Models' component

This component has the models that represents the entities used in the application and global variables:

- Course: Pertains to course-related details.
- Professor: Pertains to professor-related details and methods.
- Room: Pertains to classroom-related details.
- Students-Group: Pertains to student group-related details.
- Chromosome: A set of parameters which define a proposed solution of the problem that the genetic algorithm aims to solve.
- Schedule: Initializes chromosomes with configuration block.
- Criteria: Checks constraints such as "Is there an overlap?" or "Are there enough seats?".
- Reservation: Manages Time, Day, and Room reservation.
- Configuration: Contains methods to configure the timetable such as getters, calculations, and parsing files.
- Constant: Holds global variables used in the system, such as the number of days in a week and hours available in the schedule.
- CourseClass: Represents the relationship between a course and a classroom.

3.1.2 Genetic Algorithms component

- **NSGA-II:** A fast and elitist multi-objectives genetic algorithm[8].
- **NSGA-III:** An Evolutionary Many-Objective Optimization Algorithm Using Reference Point-Based Nondominated Sorting Approach[9].
- **APNSGA-III:** Adaptive Population NSGA-III with Dual Control Strategy[10].
- **AMGA2:** Archive-based Micro Genetic Algorithm[11].
- **EMoSOA:** Evolutionary multi-objective seagull optimization algorithm[12].
- **BGA:** Genetic algorithm with a new biological operator[13].
- **Hgasoo:** Hybrid Genetic Algorithm and Sperm Swarm Optimization[14].
- **Ngra:** Non-dominated Ranking Genetic Algorithm[15].

3.1.3 App component

The app component contains 3 main parts:

- Data: A JSON file that seeds the data of the entities such as Professors, Courses, Classes, etc..

- Output Files: Files with different formats to represent the output, such as in HTML and JSON format.
- Console App: The main program app that initiate the process, call the methods, and represent the output using the output files.

3.2 Application implementation

The number of working days is set to 5 days a week, with 8 working hours per day. Five constraints are applied:

- no classroom overlapping (R).
- sufficient seats for students in each class (S).
- adequate equipment for classes if needed (L).
- no scheduling conflicts for professors (P).
- and no overlapping between student groups (G).

After the global variables are declared, the constraints are set, and the data is filled, the console app is run. the following steps will demonstrate what happens logically in the code:

1. Start a stopwatch to count the time taken in the process.
2. Configure the configuration file using the data filled in the data file.
3. Create a schedule using the configuration file.
4. Call the desired Genetic Algorithm file and pass the schedule parameter to it.
5. Get the results from the algorithm function and represent it with the desired output file format.
6. Display notes and other parameters on the console terminal, such as how many generations it takes to reach the optimal solution, and how much time.

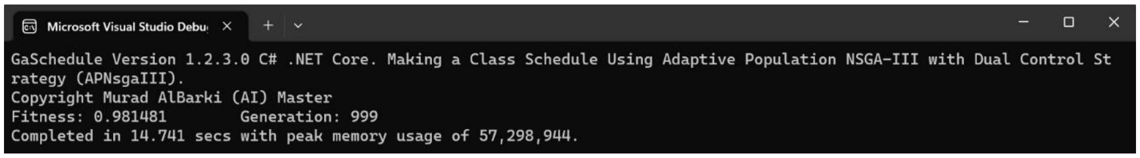
3.3 *APNsgaIII vs Emosoa*

Two methods of the genetic algorithms have been taken to the comparison, Adaptive Population NSGA-III with Dual Control Strategy (APNsga-III) and Evolutionary multi-objective seagull optimization algorithm (EMoSOA) with maximum iteration of 1000 generation and minimum fitness of 99.9% to stop the search, whichever reached first.

Volume 12 Issue 01
June 2024
ISSN 2958-6119

- ***APNsgaIII***

The results of implementing this method as seen in Figure (8)



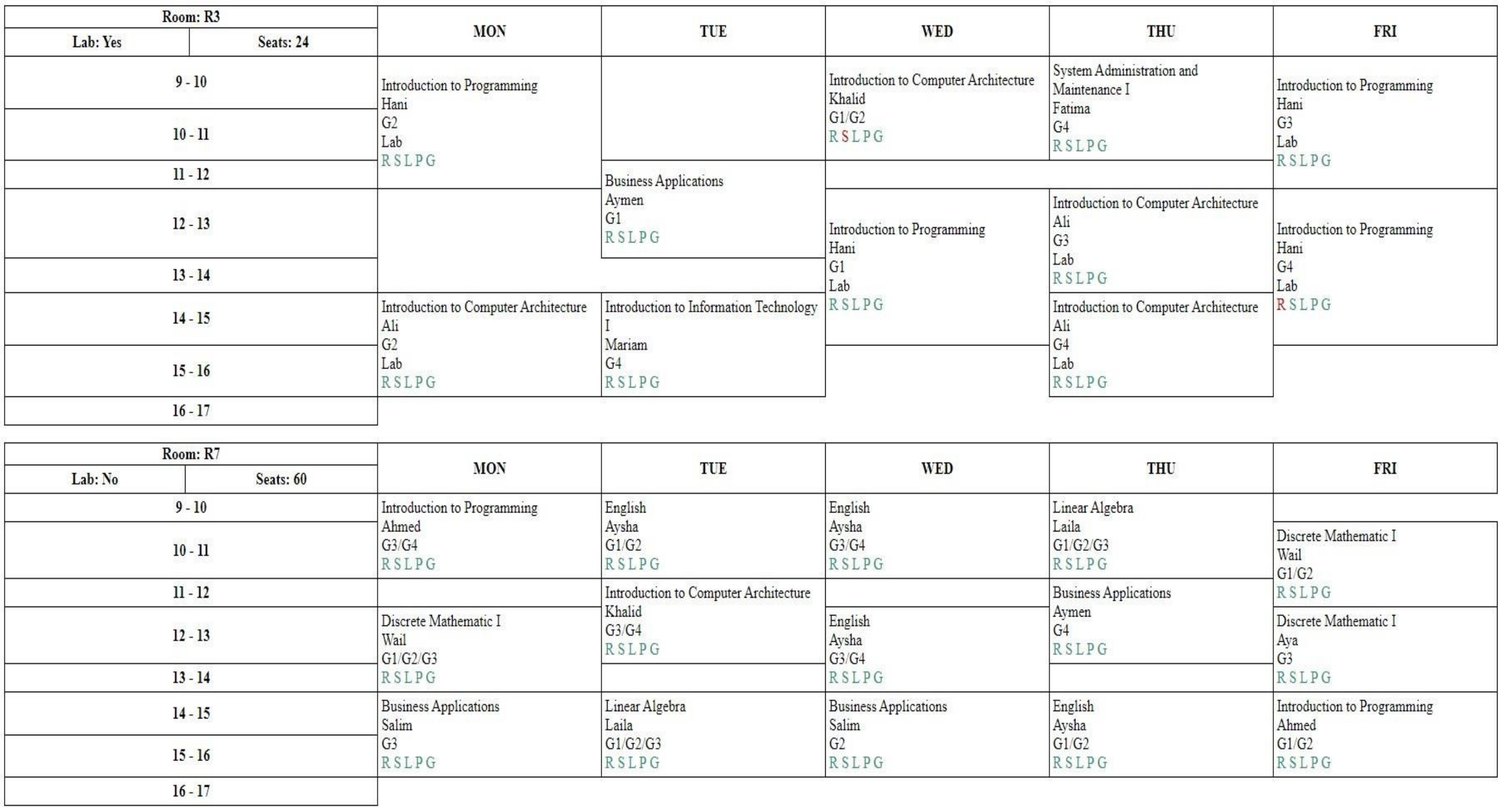
```
GaSchedule Version 1.2.3.0 C# .NET Core. Making a Class Schedule Using Adaptive Population NSGA-III with Dual Control St
rategy (APNsgaIII).
Copyright Murad AlBarki (AI) Master
Fitness: 0.981481        Generation: 999
Completed in 14.741 secs with peak memory usage of 57,298,944.
```

Figure (8) APNsgaIII Console Results.

The best fitness reached was 98.1481%, the search stopped after reaching the maximum iteration of 1000 generation while it took 14.741 seconds to reach the result in Table (2).

The result shows that there is two constrains have not been achieved in two classes in the schedule, once with not enough student seats (S) and one with conflict in the room (R).

Table (2) APNsgaIII HTML Output Results



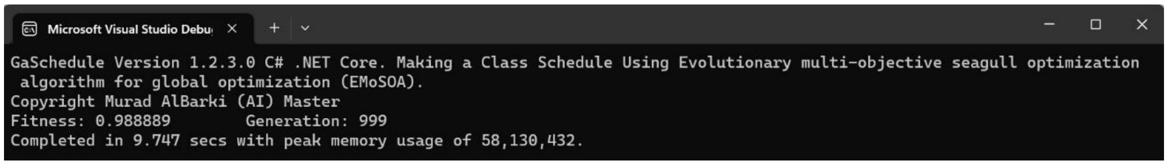
Room: R3 Lab: Yes / Seats: 24	MON	TUE	WED	THU	FRI
9 - 10	Introduction to Programming Hani G2 Lab R S L P G		Introduction to Computer Architecture Khalid G1/G2 R S L P G	System Administration and Maintenance I Fatima G4 R S L P G	Introduction to Programming Hani G3 Lab R S L P G
10 - 11					
11 - 12		Business Applications Aymen G1 R S L P G			
12 - 13			Introduction to Programming Hani G1 Lab R S L P G	Introduction to Computer Architecture Ali G3 Lab R S L P G	Introduction to Programming Hani G4 Lab R S L P G
13 - 14					
14 - 15	Introduction to Computer Architecture Ali G2 Lab R S L P G	Introduction to Information Technology I Mariam G4 R S L P G		Introduction to Computer Architecture Ali G4 Lab R S L P G	
15 - 16					
16 - 17					

Room: R7 Lab: No / Seats: 60	MON	TUE	WED	THU	FRI
9 - 10	Introduction to Programming Ahmed G3/G4 R S L P G	English Aysha G1/G2 R S L P G	English Aysha G3/G4 R S L P G	Linear Algebra Laila G1/G2/G3 R S L P G	
10 - 11					Discrete Mathematic I Wail G1/G2 R S L P G
11 - 12		Introduction to Computer Architecture Khalid G3/G4 R S L P G		Business Applications Aymen G4 R S L P G	
12 - 13	Discrete Mathematic I Wail G1/G2/G3 R S L P G		English Aysha G3/G4 R S L P G		Discrete Mathematic I Aya G3 R S L P G
13 - 14					
14 - 15	Business Applications Salim G3 R S L P G	Linear Algebra Laila G1/G2/G3 R S L P G	Business Applications Salim G2 R S L P G	English Aysha G1/G2 R S L P G	Introduction to Programming Ahmed G1/G2 R S L P G
15 - 16					
16 - 17					

- ***Emosoa***

The results of implementing this method as seen in Figure (9).

Volume 12 Issue 01
June 2024
ISSN 2958-6119



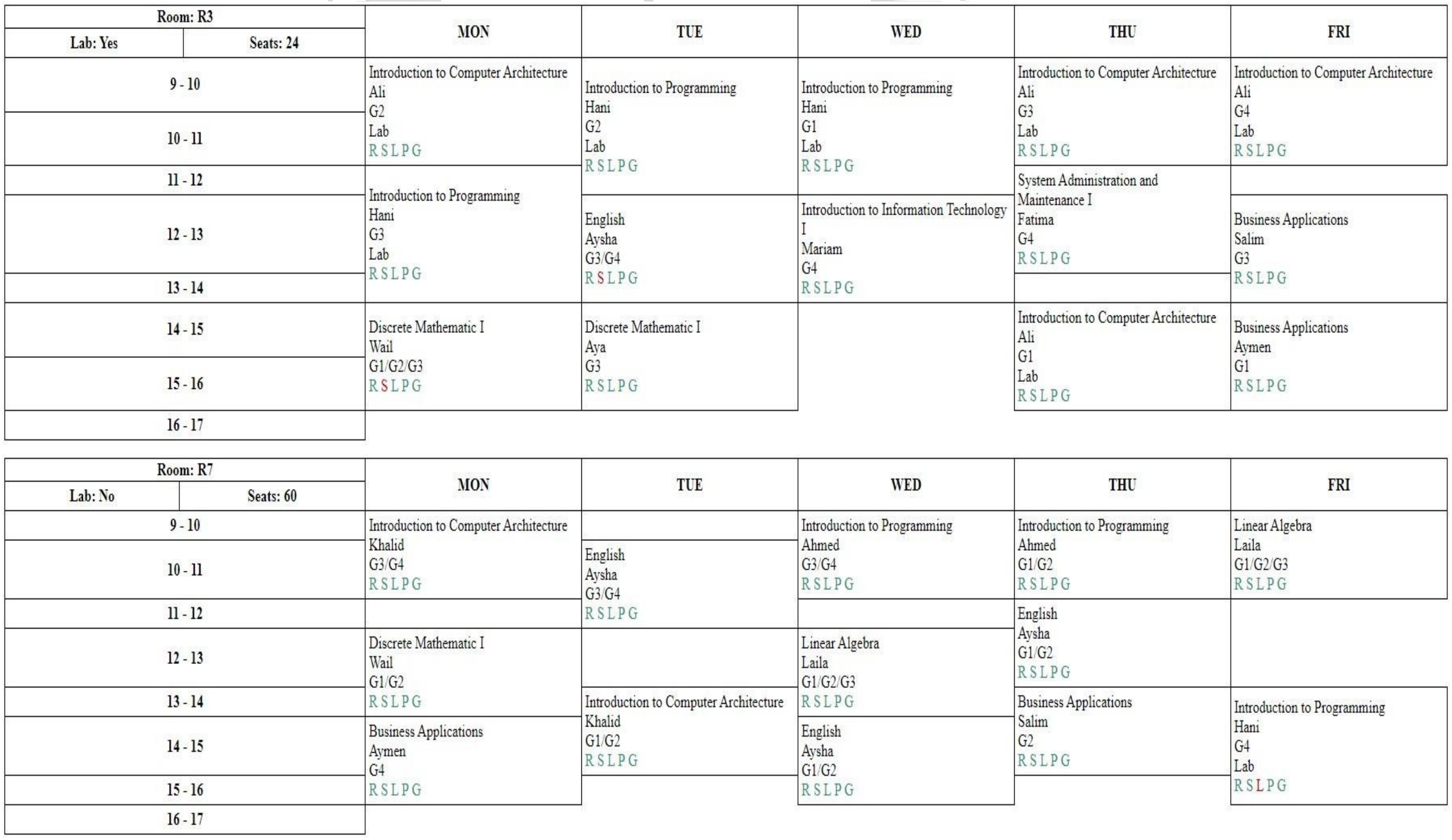
```
Microsoft Visual Studio Debu
GaSchedule Version 1.2.3.0 C# .NET Core. Making a Class Schedule Using Evolutionary multi-objective seagull optimization
 algorithm for global optimization (EMoSOA).
Copyright Murad AlBarki (AI) Master
Fitness: 0.988889        Generation: 999
Completed in 9.747 secs with peak memory usage of 58,130,432.
```

Figure (9) Emosoa Console Results.

The best fitness reached was 98.8889%, the search stopped after reaching the maximum iteration of 1000 generation while it took 9.747 seconds to reach the result in Table (3).

The result shows that there is two constrains have not been achieved in two classes in the schedule, once with not enough student seats (S) and one with not enough equipment in the lap (L).

Table (3) Emosoa HTML Output Results.



Room: R3		MON	TUE	WED	THU	FRI
Lab: Yes	Seats: 24					
9 - 10		Introduction to Computer Architecture Ali G2 Lab R S L P G	Introduction to Programming Hani G2 Lab R S L P G	Introduction to Programming Hani G1 Lab R S L P G	Introduction to Computer Architecture Ali G3 Lab R S L P G	Introduction to Computer Architecture Ali G4 Lab R S L P G
10 - 11						
11 - 12		Introduction to Programming Hani G3 Lab R S L P G			System Administration and Maintenance I Fatima G4 R S L P G	
12 - 13			English Aysha G3/G4 R S L P G	Introduction to Information Technology I Mariam G4 R S L P G		Business Applications Salim G3 R S L P G
13 - 14						
14 - 15		Discrete Mathematic I Wail G1/G2/G3 R S L P G	Discrete Mathematic I Aya G3 R S L P G		Introduction to Computer Architecture Ali G1 Lab R S L P G	Business Applications Aymen G1 R S L P G
15 - 16						
16 - 17						

Room: R7		MON	TUE	WED	THU	FRI
Lab: No	Seats: 60					
9 - 10		Introduction to Computer Architecture Khalid G3/G4 R S L P G		Introduction to Programming Ahmed G3/G4 R S L P G	Introduction to Programming Ahmed G1/G2 R S L P G	Linear Algebra Laila G1/G2/G3 R S L P G
10 - 11			English Aysha G3/G4 R S L P G			
11 - 12					English Aysha G1/G2 R S L P G	
12 - 13		Discrete Mathematic I Wail G1/G2 R S L P G		Linear Algebra Laila G1/G2/G3 R S L P G		
13 - 14			Introduction to Computer Architecture Khalid G1/G2 R S L P G		Business Applications Salim G2 R S L P G	Introduction to Programming Hani G4 Lab R S L P G
14 - 15		Business Applications Aymen G4 R S L P G		English Aysha G1/G2 R S L P G		
15 - 16						
16 - 17						

The results of our comparison shows the advantage of using the Emosoa method over the APNsgaIII method when it comes to the fitness level and time consumption.

Conclusion

The conclusions drawn from this paper can be summarized as follows:

- Genetic Algorithm Overview: The genetic algorithm is an optimization technique for solving both constrained and unconstrained problems, inspired by the principles of natural selection in biological evolution. This algorithm iteratively modifies a population of individual solutions.

- Population Dynamics: The population size remains constant. As new generations are created, individuals with the lowest fitness are eliminated, making room for new offspring.

- Generational Improvement: Through a repeated sequence of phases, each new generation produces individuals that are generally better than those in the previous generation.

- The fitness function must effectively measure the quality of a given solution. Specifically, it should be capable of evaluating the generated solutions and providing clear guidance on how to improve them.

  For instance, a fitness function that assigns a zero value unless the solution is correct is suboptimal, as it fails to provide incremental feedback on the proximity of a solution to the optimal answer. Similarly, a fitness function that increases with improving solutions but does not distinctly identify the optimal solution is also inadequate, as it may cause the population to converge prematurely and stagnate without reaching the optimal solution.

References

[1] “Introduction to Genetic Algorithms — Including Example Code | by Vijini Mallawaarachchi | Towards Data Science.” Accessed: May 26, 2024. [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
[2] “Genetic algorithm - Wikipedia.” Accessed: May 26, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Genetic_algorithm
[3] “How to define a Fitness Function in a Genetic Algorithm? | by Vijini Mallawaarachchi | Towards Data Science.” Accessed: May 26, 2024. [Online]. Available: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4
[4] “Genetic Algorithm.” Accessed: Mar. 06, 2023. [Online]. Available: https://www.unzmarkt-frauenburg.at/blog/genetic-algorithm.php
[5] “Crossover in Genetic Algorithm - GeeksforGeeks.” Accessed: May 26, 2024. [Online].

Available: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/
[6] J. Carr, “An introduction to genetic algorithms,” *Sr. Proj.*, vol. 1, no. 40, p. 7, 2014.
[7] W. H. Hsu, “Genetic algorithms,” *Dep. Comput. Inf. Sci. Kansas State Univ.*, vol. 234, pp. 62302–66506, 2004.
[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.
[9] K. Deb and H. Jain, “An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints,” *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, 2013.
[10] M. Wu *et al.*, “Adaptive population nsga-iii with dual control strategy for flexible job shop scheduling problem with the consideration of energy consumption and weight,” *Machines*, vol. 9, no. 12, p. 344, 2021.
[11] S. Tiwari, G. Fadel, and K. Deb, “AMGA2: improving the performance of the archive-based micro-genetic algorithm for multi-objective optimization,” *Eng. Optim.*, vol. 43, no. 4, pp. 377–401, 2011.
[12] G. Dhiman, K. Singh, A. Slowik, V. Chang, A. Yildiz, and A. Kaur, “EMoSOA: A New Evolutionary Multi-objective Seagull Optimization Algorithm for Global Optimization,” *Int. J. Mach. Learn. Cybern.*, vol. 12, Feb. 2021, doi: 10.1007/s13042-020-01189-1.
[13] R. Lakshmi, K. Vivekanandhan, and R. Brintha, “A New Biological Operator in Genetic Algorithm for Class Scheduling Problem,” *Int. J. Comput. Appl.*, vol. 60, pp. 6–11, Dec. 2012, doi: 10.5120/9742-4293.
[14] H. Shehadeh, H. Mustafa, and M. Tubishat, “A Hybrid Genetic Algorithm and Sperm Swarm Optimization (HGASSO) for Multimodal Functions,” *Int. J. Appl. Metaheuristic Comput.*, vol. 13, Jan. 2022, doi: 10.4018/IJAMC.292507.
[15] O. Al Jadaan, L. Rajamani, and C. R. Rao, “NON-DOMINATED RANKED GENETIC ALGORITHM FOR SOLVING MULTI-OBJECTIVE OPTIMIZATION PROBLEMS: NRGA.,” *J. Theor. Appl. Inf. Technol.*, vol. 4, no. 1, 2008.