

Using Genetic Algorithms to Solve Computer Science Olympiad Optimization Problems

Can Almelek, Ozgur Guzeldereli

Uskudar American Academy - Selamiali, Vakıf Sk. No:1 D:No:1, 34664 Üsküdar/İstanbul, Turkey; canalmelek44@gmail.com,

ABSTRACT: Genetic algorithms are random search algorithms based on biological evolution theory. In this study, the utilization of genetic algorithms in solving optimization problems from informatics Olympiads is investigated. It is hypothesized that genetic algorithms will be able to propose solutions to the optimization problem within a predetermined percent margin of the actual solution. A genetic algorithm is implemented in the C# programming language to test this hypothesis. Three tests are conducted, and their data is examined with various parameters taken into consideration. Consequently, the implemented genetic algorithm has been demonstrated to successfully solve the varying number of inputs (500, 1000, 2000) within an acceptable range. It is also concluded that several improvements and optimizations must be made to utilize this algorithm in a competitive landscape as time and memory constraints are exceeded. Furthermore, the tests concede that the number of inputs and the number of generations required to converge on an optimal solution is directly correlated. Therefore, this outcome should be taken into consideration when designing and tuning genetic algorithms with various parameters if the problem is aimed to be solved within the time and memory constraints of the competitive landscape.

KEYWORDS: Systems Software; Algorithms; Competitive Programming; Genetic Algorithms; Optimization.

■ Introduction

Computer science has become the epitome of global development into a more modern and enhanced society. With countless advancements in the fields of Artificial Intelligence (AI), Virtual Reality (VR), Augmented Reality (AR), robotics technologies, etc.; computer science has validated its crucial role in the new age. Therefore, there is a constant endeavor to improve the effects of computer science by coming up with new methods as well as educating more people. To be precise, the percentage of high schools offering computer science courses in the US has increased from 35% to 51% in the last 3 years.¹ This is a testament to the fact that more education institutions, students, teachers, and individuals emphasize computer science and its potential in the future.

As computer science's popularity prospers, there have been numerous means to assess one's knowledge and skills on a computer. There are coding websites, mainly Codeforces and Hackerrank designed to impose problem-based coding with occasional contests in a competitive manner. This system of problem-based coding in a competitive manner is essentially called competitive programming. The broader rendition would be an Olympiad in Informatics on regional and international scales. The world's most prestigious computer science competition is the International Olympiad in Informatics (IOI), which attracted 327 contestants from 97 countries in IOI 2019 Baku, Azerbaijan; all of whom passed several stages in national contests.² In these contests, there are common problem types such as Graph Theory, Game Theory, and recursion which are regularly displayed in problem tasks. One of the key objectives is problems that require finding the best solution from all feasible solutions such as optimization problems. There are numerous algorithms designed to solve

optimization problems with the motive of determining the most optimal solution. Genetic algorithms (GAs) are one of the most fundamental algorithms that incorporate biology and genetic classification into computer science as a viable solution method.³

This study demonstrates the utilization of genetic algorithms in solving optimization problems from the Olympiad in informatics. It is hypothesized that genetic algorithms would be able to propose a solution within a certain percent margin of accuracy to Olympiad problems which are conventionally solved by other means, given the correct parameters. Within this paper, the structure of genetic algorithms will be explained, optimization problems will be defined, our implementation of a genetic algorithm will be shown, and with the given implementation an optimization problem will be attempted to be solved.

■ Methods

Genetic Algorithms:

Genetic Algorithms are optimization algorithms that are inspired by natural selection. Genetic algorithms were first designed by John Holland at the University of Michigan in the early 1970s.⁴ They involve processes such as crossover and mutation to optimize for a constraint. Genetic algorithms can be used to generate feasible and high-quality solutions for search and optimization problems. Although the structure of genetic algorithms can vary slightly with their implementation, most, if not all, genetic algorithms have several commonalities: initialization, selection, offspring generation, mutation, and termination.⁵

Initialization:

All genetic algorithms start with an initialization step. Genetic algorithms consist of a population of individuals.

Individuals carry a gene sequence or DNA. The DNA of an individual represents a solution to the problem that is aimed to be solved and the DNA should be structured accordingly. Therefore, each individual is required to be initialized to ensure they have at least some DNA. Although many techniques could be applied to initialize the individuals, two techniques are most prevalent. Of these two methods, assigning random genes to individuals is the most common. The other method is seeding the individual which signifies giving random genes that could be precise to the optimal value of the system or the problem. Initializing the individuals is quintessential to a genetic algorithm. Randomly assigning gene values makes genetic-diversity possible within the population which is key to spanning the entirety of the solution set of a particular problem.

Selection:

After individuals are initialized, the selection process takes place. Besides a DNA sequence, the other metric every individual has is the fitness value of their DNA. The fitness value of an individual determines how optimal it is, or how close the individual is to an optimal solution relative to other individuals. The fitness value is calculated via a fitness function which is a problem-specific function determining the optimality of a solution that is proposed by the genes of an individual. Therefore, it is important to form a feasible fitness function and it is the hardest process that is involved in constructing a genetic algorithm. If the fitness function fails to produce high-quality fitness values, the genetic algorithm will not be able to create high-quality solutions to the problem. After a valid fitness function is created, every individual's fitness is calculated. Then the individuals are sorted in the population according to their fitnesses and a percentage of lowest fitness individuals is discarded. Some low-fitness individuals are left in the population to ensure the genetic diversity within the population.

Offspring generation:

After the selection process, a new empty generation is created and filled via the crossover of the previous generation's individuals. Crossover is the process in which the genes of an individual are mixed with another individual's genes to create offspring. The process involves selecting random points in the genes of the individuals. Then two empty children are created. The genes of the children are set to be the gene of one of the parents (one for each child). In the points that are randomly selected, the parent that the child is getting the gene from switches to the other parent thus crossing over. This process can be repeated multiple times to ensure more diverse results. Crossover's main purpose is to generate new offspring from high fitness individuals to further increase the fitness of the population. By selecting from high fitness individuals and crossing over their genes, a gene sequence with a higher fitness level is created.

Mutation:

As with crossover, the mutation is a genetic operator that is used in creating higher fitness solutions in a genetic algorithm. The mutation is the process of randomly changing the genes of the individual. There could be several consequences of this

action. Firstly, a higher fitness solution could be reached which is the optimal case. Another case would be a lower fitness solution that could be reached in which case the mutated child will not be able to pass its genes to the next generation. Either way, genetic diversity is created within the population to ensure that the system doesn't converge to a local optimum. By randomly changing the genes of the children of the new generation, enough new genes are inputted into the system to widen the search space of the algorithm. In elitist genetic algorithms where the fittest individuals of each generation are passed directly to the next generation, the mutation is also excluded from elite individuals to protect the progress of the genetic algorithm as mutations can lead to a decrease in fitness.

Termination:

Termination is the last step of a genetic algorithm. The termination of the genetic algorithm may depend on several factors and several termination conditions. For example, the number of generations could be constrained as well as the time spent on the problem. Other constraints include but are not limited to reaching a certain fitness level, the solution satisfies the bare minimum requirements of the problem, or a point is reached where the successive generations no longer produce new results. Usually, after termination, a solution is outputted as a solution to the problem or optimization. Data from each generation can be used to plot data graphs and the evolution of the population can be analyzed to draw further conclusions.

Pseudocode of a Basic Genetic Algorithm:

1. Initialization - All individuals of the population are initialized with n-length genes which represent a solution to the problem.
2. Fitness - The fitness of each individual is calculated using a fitness function.
3. New generation - Follow the proceeding steps to create a new generation.
4. Selection - A percent of the population is selected. The higher the fitness the higher the chance of selection.
5. Crossover - From the selected individuals, randomly select 2 parents. Crossover their genes to make 2 new offspring.
6. Accepting - Place new offspring in a new population
7. Elitism - If the genetic algorithm is elitist, select the highest fitness individuals of the population and also place them into the new population.
8. Replace - Use the newly generated population for another run of the algorithm.
9. Test - Test for a termination condition. If the condition is met, terminate the algorithm and return the necessary data as a solution to the problem or optimization.
10. Repeat - Go to step 2.

Optimization Problems:

As mentioned previously, optimization problems are the problems that require finding the best solution out of all feasible solutions. The whole entity of feasible solutions is often referred to as sub-optimal (local) solutions.⁶ A specific goal is asked to be met at an optimal value out of all sub-optimal solutions. These problems must provide the variable con

straints as specifications necessary for constructing a solution. Accordingly, programmers are generally obligated to find the minimum and maximum values of a function that satisfies the task's objective.

Moreover, optimization problems can be classified in two manners according to their variables: continuous or discrete. In the cases of discrete variables in an optimization problem, an integer, graph, or permutation must be identified out of a countable set. In discrete optimization, some or all of the variables in a model are required to belong to a discrete set; this is in contrast to continuous optimization in which the variables are allowed to take on any value within a range of values. The discrete set is mainly initialized as subsets, combinations, graphs, or sequences. On the other hand, the cases of continuous variables require a continuous function to be found. Continuous optimization means finding the minimum or maximum value of a function of one or many real variables, subject to constraints. The constraints usually take the form of equations or inequalities.

In general, a rather popular method for solving optimization problems is Dynamic Programming (DP).⁷ DP is essentially a recursive solution with repeated method calls, storing the results of subproblems for the eventual solution. Compared to the aforementioned genetic algorithms, DP solutions usually use a lot of memory and have a longer running time. Even though genetic algorithms are typically harder to implement, efficiency in time and memory capacity are highly valuable assets for competitive programming which makes genetic algorithms more effective in Olympiad in Informatics.⁸

■ Results and Discussion

Genetic Algorithms can be considered systematic random search algorithms where the random search algorithm also considers the optimality of previous trials and evolves accordingly. Randomly initializing the individuals causes the proposed solutions to span the search space. With each generation, the solutions then attempt to converge at a global optimum. Mutations add genetic diversity to the gene pool and aim to ensure that the point the population converges is the global optimum. Knowing these it can still be inferred that a genetic algorithm may not be able to converge on the most optimal solution even with the best possible parameters as it is still a randomized and probabilistic algorithm. Although that may be the case, it could be expected that the final result of a genetic algorithm is close to the global optimum. For the set of runs that will be conducted in this paper, it is assumed that the problem is solved if the value of the solution proposed by the algorithm is within 5% of the actual solution after 2000 generations. The percentage values of how close the solution is to the actual solution are calculated using the following formula:

$$\left| \frac{\text{proposed solution} - \text{actual solution}}{\text{actual solution}} \right| \times 100$$

Another factor to consider is that genetic algorithms are intrinsically parallel compared to most algorithms which are serial. Serial algorithms only permit linear exploration that does not store every value reached by the algorithm's "infer" steps. On the contrary, genetic algorithms contain multiple

offspring which allows multiple dimensional explorations. Depending on the accuracy of the offspring, some are eliminated, whereas some are pursued as promising solutions. Due to the parallelism, they can operate resembling a graph structure, evaluating multiple tracers at once. Thanks to their structure, particularly problems containing vast amounts of potential solutions are easier to solve which would take an immense amount of time with a simple search algorithm.

The screenshot shows the problem page for the Zonal Computing Olympiad 2014, 30 Nov 2013, Problem 2: IPL. The page includes the IARCS logo, the event name, and the time (2:00 pm - 5:00 pm IST). The problem statement describes the IPL 2015 season and the goal of maximizing earnings through match fees. It includes input and output formats, sample inputs and outputs, and test data constraints.

Figure 1: ZCO 2014 Problem 4 IPL.⁹

Identifying the sections of a problem:

First and foremost, it is essential to identify crucial sections of a problem task before analyzing any algorithms or writing code. In this problem titled IPL, a certain number of games (N) are given according to the user's input. Afterward, N number of games is expected from the user with each containing a particular fee awarded if played in the game. The problem calls for the maximum amount of money that could be earned for an individual (Nikhil) through the N games. However, there is a condition in which an individual is obligated to play at most 2 consecutive games. Herewith, a certain algorithm must be constructed which maximizes salary while taking at most 2 consecutive game conditions into account (Figure 1).

Furthermore, a prediction about the complexity of a problem could be made per the given test data constraints. A maximum of $2 \cdot 10^5$ number of games along with 10^4 for each game can be inputted by the user. An average computer is able to make approximately $2 \cdot 10^6$ calculations per second regardless of the operating system. Thus, a brute force algorithm with time complexity of $O(n^2)$ would be sufficient for a solution to pass this problem. The time complexity of $O(n^2)$ signifies the worst solution possible for a problem which usually persists in iterating through every single value until the correct answer is reached. Nonetheless, a brute force solution wouldn't pass for this problem if the user could input $2 \cdot 10^8$ number of games. In a hypothetical case, a genetic algorithm would be one of the most ideal solution methods for this problem with the time complexity of $O(nm)$. Among the time complexity of

O(gnm), "g" stands for the number of generations, "n" stands for the population size, whereas "m" signifies the size of individuals.¹⁰ In brief, although in this particular case it's not imperative, a genetic algorithm solution would be a fast and viable solution to the problem among random search and brute force algorithms.

Genetic Algorithms Implementation:

The genetic algorithm is implemented via the programming language C# which is a general-purpose, object-oriented programming language developed by Microsoft. The following section breaks down the code into several sections - namely initialization, selection, crossover, mutation, and termination.

Before the function of the code is analyzed, its inner structure must be understood.¹¹ First of all, the code starts with a class named Individual, which represents one individual in the entirety of the population. Each contains a set of genes that are of template type, a double fitness value, and a chromosome length which is common throughout the entire population.

```
class Individual<T> {
    public T[] genes; //instance variables
    private int chromosomeLength;
    public double fitness;
    ...
}
```

Its implementation is as follows:

All of these instance variables are initialized with a constructor as each instance of the Individual class is created. Another structure within the algorithm is the Population class. The class contains an array of individuals, a generation number, a population size, a chromosome size, and various probability values for the genetic algorithm to work on. The implementation of

```
class Population<T> {
    public Individual<T>[] individuals;
    public double generationNumber;
    public Individual<T> fittestMember;
    private int populationSize;
    private int chromosomeSize;
    private double mutationProbability;
    private Func<T> GetRandomGene;
    private Func<T[], double> CalculateFitness;
    private double elitismPercent;
    private double selectPercent;
    ...
}
```

its instance variables is as follows:

As evident from the code snippet, the population class also contains two functions: GetRandomGene() and CalculateFitness() functions. These functions are provided by the client of these classes and are called within the population's various functions. GetRandomGene() is the function that returns a random gene to initialize the members. CalculateFitness() is a function that returns a double value according to the genes of the individual; the number returned signifies the fitness of that particular gene set or individual. As cognizant of solely the meaning and implementation of these values, it is possible to analyze the code entirely. The first step in the code is the initialization of the individuals. The initialization method iterates through the array of individuals and sets the gene of each individual using the GetRandomGene() method. For the particular problem, the genes of the members are an array of integers. All values in the array are either 1 or 0. The gene sequence represents the sequence of games and each 1 in the

gene represents that a match is going to be played while each 0 represents that the match will be passed.

After the initialization of the population is complete, the process of selection begins. Before the selection can order and remove individuals from the population, the fitness of each individual has to be calculated. The CalculateAllFitness() function located inside the Population class iterates through all of the population's individuals once more and calls the CalculateFitness() method on their gene sequence and sets their fitness value to be the return value of the method. The CalculateAllFitness() method is implemented as such:

```
public void CalculateAllFitness() {
    foreach (var individual in individuals) {
        individual.fitness = CalculateFitness(individual.genes);
    }
}
```

The CalculateFitness() function is provided by the client of the Population class. For this particular problem, the fitness is calculated to be the highest fee that can be reached within the gene sequence. A fee is calculated from a series of matches until 3 consecutive matches are played which is derived from the condition of the problem. If 3 consecutive matches are played, the fitness value is reset to 0 and the value it contains before resetting is compared to the *biggestFitness* value. If the fitness calculated from this particular series of matches within the gene sequence is bigger than the biggestFitness value, the *biggestFitness* value is set to be the fitness of this series. This process aims to maximize the fee earned from the gene sequence as it attempts to maximize for the largest value of fee possible while also keeping the amount of three consecutive matches played to the lowest possible value.

After all of the fitnesses are calculated the selection takes place. The selection function orders each Individual according to their fitness and a percent of the population whose fitness is lower is removed from the population. The percentage which the population is selected is given as a parameter to the method (selectPercent). The selection method is also a part of the Population class, and it is implemented in the following code snippet:

```
public void Selection(double selectPercent) {
    Individual<T>[] SortedList = individuals.OrderByDescending
        (o => o.fitness).ToArray();
    fittestMember = SortedList[0]; /* Fittest member will later be used to
        collect data about the progress of the algorithm */
    individuals = SortedList.Take
        ((int)(populationSize * selectPercent)).ToArray();
}
```

After selection is completed, a new generation is created. Another array of individuals is initialized with all of the individuals being empty. Then a percentage of the previous generation is passed onto the next generation starting from the highest fitness. This is called elitism. The remaining empty places in the population are filled using the offspring of the individuals of the previous generation. To produce offspring two parents are selected using weighted random selection which uses the individuals' fitness as weight. After the selection is made, crossover and mutation are applied. The two offspring are placed into the new generation and this process is repeated until the new generation becomes just as populated as the pre

vious generation was. The code for making a new generation is implemented as follows:

```
public void MakeNewGeneration() {
    List<Individual<T>> currentGeneration = individuals.ToList();
    List<Individual<T>> newGeneration = new
        List<Individual<T>>(populationSize);
    newGeneration = individuals.Take
        ((int)(elitismPercent * populationSize)).ToList(); /* Elitism is
        applied, a percent of the highest fitness individuals are passed onto the next
        generation */
    Individual<T> parent1, parent2;
    double totalFitness = 0;
    foreach (var ind in currentGeneration){
        totalFitness += ind.fitness; /* Total fitness is calculated to weight
        selection of new parents */
    }
    while(newGeneration.Count < populationSize) {
        parent1 = GetRandomByFitness(totalFitness); /* Weighted random selection
        is applied; higher the fitness, higher the chance of being selected */
        parent2 = GetRandomByFitness(totalFitness);
        var children = parent1.CrossoverReproduction(parent2);
        foreach (var child in children) {
            child.Mutate(mutationProbability, GetRandomGene);
            if (newGeneration.Count < populationSize)
                newGeneration.Add(child);
        }
    }
    individuals = newGeneration.ToArray();
}
```

All of these functions are put together in a PropagateGeneration method to simplify the entire process. PropagateGeneration is defined in the following way:

```
public void PropagateGeneration() {
    CalculateAllFitness();
    Selection(selectPercent);
    MakeNewGeneration();
}
```

The last step is the termination. For this particular problem, the termination condition is the number of generations that the algorithm ran for. In the tests made, every algorithm was run for 2000 generations before being terminated. The fitness levels were not adjusted or altered and were taken as is from when the algorithm stopped.

Evaluation of the Solution:

A total of three runs were made. Each algorithm is left to run for 2000 generations and with a population of 10000 individuals. Runs were made using N values of 500, 1000, and 2000. In these tests, fitness also represents the proposed solution of a gene sequence which in this case is the highest fee an individual can earn. This works as both the fitness and the solution to the problem are trying to be maximized. In each generation, 50 percent of the low fitness population is removed, and others are allowed to produce offspring. The mutation chance is 80%. This value can alter with the rate of convergence and prevent it dramatically, but it is required to be extremely high to cover the vast search space. Elitism percent is 5% meaning in each generation's highest fitness 5% of the population is passed onto the next generation without being subjected to any alteration. The processed data from the runs are entered into Table 1. The best fitness for each generation is plotted as a function of generation count and Figures 2, 3, and 4 are created.

The data suggests that for the tests done, the algorithm is successful in finding an optimal solution to the problem. It should be noted that for the third test the percentage difference is very close to the threshold value. This increase in the percentage difference is caused by inadequate parameters in

the genetic algorithm, most notably the generation count and

Solution Qualities	TEST I	TEST II	TEST III
Generation Count:	2000	2000	2000
Population Size:	10000	10000	10000
Inputs	N = 500	N = 1000	N = 2000
Best Fitness	1920717	3794093	7243546
Actual Solution	1950445	3874387	7614676
Percentage Difference	1.5%	2.1%	4.8%
Status	Solved	Solved	Solved

population size. Given a higher generation count, it is expected that the genetic algorithm converges closer to the value, resulting in a lower percentage difference. Another test with the same parameters except for generation count as test 3 is done to show this phenomenon. Note that the actual solution is different from test 3 as all the input values are randomized.

As can be observed from this test (Table 2), the percentage difference drops significantly when a larger generation count is used. Table 2: ZCO 2014 Problem 4 test II.

	Generation Count	Best Fitness	Actual solution	Percentage Difference
TEST III (Revised)	4000	7440479	7615239	2.3%

difference drops significantly when a larger generation count is used. Considering all of these tests, it could be concluded that genetic algorithms are adequate in solving such problems but require a lot of time and memory to run efficiently. Along with this, as the amount of input values gets bigger, the amount of generation required to converge on an optimal solution increases. This fact must then be taken into consideration when designing problem-specific genetic algorithms. Since time and memory are usually restrained on Olympiad questions, an efficient algorithm must be implemented with the best parameters possible.

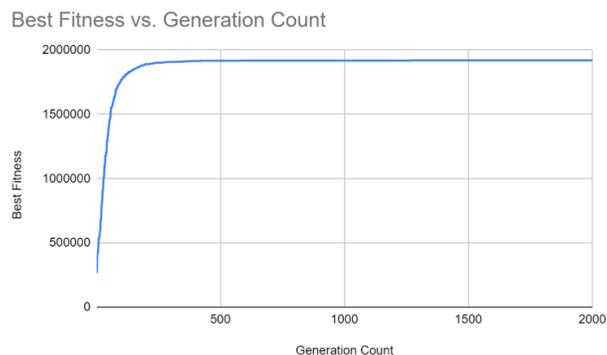


Figure 2: Best Fitness vs Generation Count Graph for Test I.

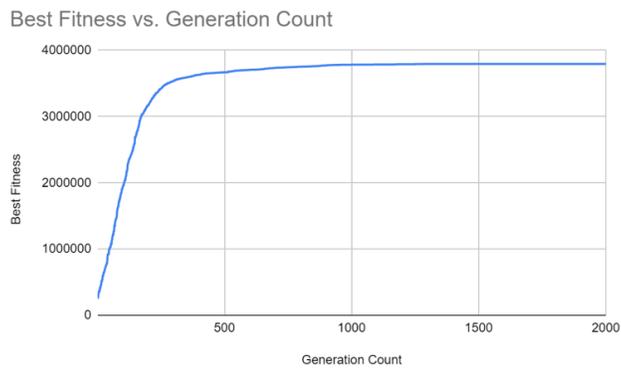


Figure 3: Best Fitness vs Generation Count Graph for Test II.

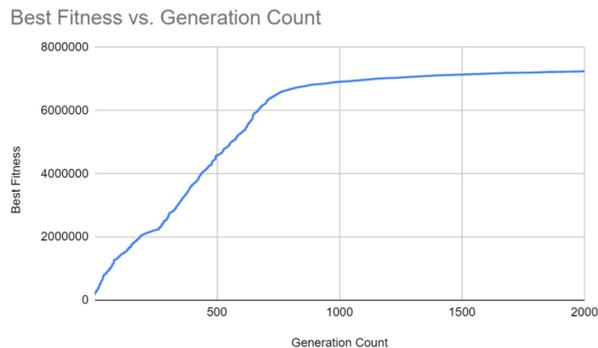


Figure 4: Best Fitness vs Generation Count Graph for Test III.

■ Conclusion

Thanks to the nature of genetic algorithms, competitive programmers and programming enthusiasts have the upper hand in solving optimization problems. Although genetic algorithms are able to provide optimal solutions for a majority of optimization problems, a couple of drawbacks are noted from the solution of IPL. Firstly, since genetic algorithms require a set of generations to be processed in order to reach the final accurate answer, time and memory constraints are generally exceeded. For instance, in the IPL example, the time limit is stated as 1 second, whereas the memory limit is stated as 32 Megabytes (MB). The solution that is at hand utilizing genetic algorithms requires at least 370 seconds and encompasses 200 MB of memory. Thus, certain alterations to the program should be made if genetic algorithms are to be used in a competitive environment. A suggestion for an alteration would be to stop new generation formation when an accurate solution is reached. A genetic algorithm provides a comprehensive search methodology for optimization. The problem of finding an accurate solution in a space with many sub-optimal solutions is a classic problem for all systems that can adapt and learn.

In conclusion, there have been benefits and drawbacks that have emerged as genetic algorithms were utilized in this study in order to solve optimization problems from informatics Olympiads. It is observed that genetic algorithms are generally able to solve such problems but are unable to comply with the memory and time restrictions. It should also be noted that converging on higher input values, requires more generations to be run on the algorithm, or increased population size. Consequently, a competitor in such informatics Olympiads must consider the possibility of using genetic algorithms as a tool

for solving the problems yet must also approach the algorithm with caution as the implementation has to be optimized heavily in order to comply with time and memory constraints of the Olympiads.

■ Acknowledgements

Our IB High-Level (HL) Computer Science (CS) teacher Özgür Çiçek encouraged us to research genetic algorithms since it has been the IB CS HL Paper 3 topic for the past 2 years.

■ References

1. Klein, A. More than half of high schools now offer computer science, but inequities persist. <https://www.edweek.org/teaching-learning/more-than-half-of-high-schools-now-offer-computer-science-but-inequities-persist/2021/11> (accessed Jan 7, 2022).
2. IOI Statistics. <https://stats.ioinformatics.org/olympiads/2019> (accessed Jan 10, 2022).
3. Techopedia. What is a genetic algorithm? - definition from Techopedia <https://www.techopedia.com/definition/17137/genetic-algorithm> (accessed Jan 10, 2022).
4. Sablier. Coding games and programming challenges to code better <https://www.codingame.com/playgrounds/334/genetic-algorithms/history> (accessed Jan 30, 2022).
5. Worden, K.; Liu, S.; Das, D.; Avi Ceder, A. Genetic Algorithm <https://www.sciencedirect.com/topics/engineering/genetic-algorithm> (accessed Jan 11, 2022).
6. Lecture12-dukecomputerscience. <https://www.cs.duke.edu/courses/fall05/cps130/lectures/skiena.lectures/lecture8.pdf> (accessed Jan 12, 2022).
7. Dynamic Programming <https://www.geeksforgeeks.org/dynamic-programming/> (accessed Jan 30, 2022).
8. Continuous optimization. <https://uwaterloo.ca/combinatorics-and-optimization/research-combinatorics-and-optimization/research-areas/continuous-optimization> (accessed Jan 12, 2022).
9. Indian Computing Olympiad Archives 2012 - IARCS. <https://www.iarcs.org.in/inoi/2014/zco2014/zco2014-2b.php> (accessed Jan 17, 2022).
10. Time complexity of genetic algorithms exponentially ... <https://fernandolobo.info/p/gecco00.pdf> (accessed Jan 13, 2022). Agnishom. ZCOSolutions/ipl.cpp at master · Agnishom/Zcosolutions.
11. Bajpai, P.; Kumar, M. Genetic Algorithm - an Approach to Solve Global Optimization Problems <https://www.researchgate.net/> (accessed Jan 20, 2022).

■ Authors

Özgür Güzeldereli is a 17-year-old sophomore student at Uskudar American Academy in Turkey. He is interested in computer programming, theoretical computer science, and algorithms. He specializes in operating system development.

Can Almelek is a 16-year-old sophomore student at Uskudar American Academy in Istanbul, Turkey. His fields of interests in computer science entail competitive programming, algorithm analysis, artificial intelligence and object-oriented programming. He's also fond of visual arts, table tennis, basketball, and public speaking.

An Analysis on the Possibility of Establishing a Threshold Energy Level for the Solar Flares on the Interactions of Hard X-Ray Beams and the Ionosphere

Can Erol, Evren Sanlı, Defne Lara Balsarı, Muhsin Erhan

American Collegiate Institute, Göztepe İnönü St. No:476, Konak, İzmir, 35290, Turkey; canerol2411@gmail.com

ABSTRACT: This paper presents an analysis of the interactions of the hard x-rays emitted through the high-energy solar flares, often seen in the X and Y class, within the ionosphere which is located in the thermosphere. Hence, the paper concentrates on whether it's possible to specify a critical threshold energy level for the hard x-ray emissions from a solar flare that cause the x-ray beam to reach the Earth's crust and its inhabitants, assuming that the intensity of the hard x-ray beam is kept as a constant. This should serve as a critical point in determining the harmful effects of the solar flares in the atmosphere and furthermore on Earth as a whole. The methodology that has been followed is the application of a well-known scientific effect to the problem, the photoelectric effect, and the evaluation of the implications of its conclusions. It is then seen that as the intensity of the X-ray beam is kept as a constant, the energy of the incoming x-ray beam does not affect the process of atmospheric absorption, since energy is quantized by the photons and the decreasing wavelength does not contribute to the number of electrons liberated in the ionospheric D region.

KEYWORDS: Physics and Astronomy; Quantum Physics; Solar Flares; Ionosphere; X-Ray Emission.

■ Introduction

This research paper is focused on the examination of the mutual correlation between the hard x-rays emitted through the X and Y class high-energy solar flares with the ionosphere which is an integral part of the thermosphere.¹ The question we as a team specified our research into is "Can there be a critical energy threshold for the hard X-rays emitted through the solar flares in which the Earth's atmosphere cannot process it and if so, what would it be?" Our hypothesis was that the threshold could be found through the calculation and understanding of the Photoelectric effect that is naturally occurring by the interactions between the hard X-rays and the competence of the Earth's atmosphere and surface.²

For reference data, we researched the November 4, 2003, Giant Solar flare, it was the largest one currently recorded by scientists.³ In essence, solar flares are incidents that have immense effects on both the natural species of our planet and the infrastructure of telecommunication technology. After thoroughly examining the example of the November 4, 2003, Giant Solar flare, we started to research a joint threshold that can define the amount of radiation the Earth's atmosphere could absorb. Solar flares emit many kinds of waves ranging from Gamma to X-Rays; however, for the ease of comprehension and drawing the limits of our paper, we narrowed our research down to the examination of the amount of X-Rays that solar flares emit only.³

To better understand how X-rays interact with the structure of the Earth's atmosphere, we firstly gathered information on the Earth's atmospheric layers, namely the thermosphere and its integral part the ionosphere, as shown in Figure 1. These two layers' main function is to absorb any rays that enter the atmosphere with the abundance of Oxygen and Nitrogen

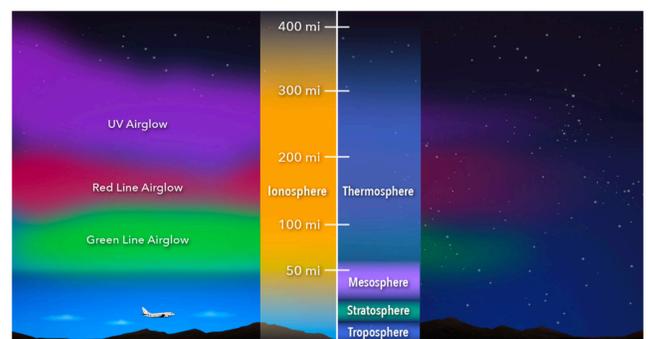


Figure 1: Diagram showing the layers of the Earth's atmosphere and ionosphere.⁴

gasses in their compositions.⁵ Essentially, the thermosphere is the layer of the atmosphere which lies just above the mesosphere. Within this layer of the atmosphere, UV light induces photoionization of molecules, resulting in the formation of ions.⁵ The formation of the ions further translates into the subsequent formation of the ionosphere. The ionosphere is a dense layer of electrons and ionized atoms and molecules that extends from 48 kilometers above the surface to 965 kilometers, intersecting with the previously mentioned layers of the mesosphere and thermosphere.⁶ This mobile zone expands and contracts in response to solar circumstances, and it is subdivided into three regions: D, E, and F, depending on the wavelength of solar light absorbed.⁶ Throughout the paper, the reader will come across the D region the most, mainly because this is the region where hard X-Ray beams interact with the atmospheric composition and are absorbed.⁷ Since the ionosphere is composed of charged particles, it is extremely sensitive to changes in magnetic and electric conditions in space. These circumstances, along with other phenomena like bursts of charged particles, are referred

to as space weather and are frequently associated with solar activity. The ionosphere is an essential connection in the Sun-Earth interaction chain, since through the ionosphere radio communications can be possible.⁷ This paper portrays an analysis of a sophisticated and detailed research question regarding the X-Ray absorption abilities of our atmosphere that hasn't been thoroughly discussed by previous literature before.

■ Results and Discussion

The paper consists of two sections: Limitations of the Photoelectric Absorption and The Effect of Quantized Energy on the Amount of Emitted Electrons.

Limitations of the Photoelectric Absorption:

The photoelectric effect is the starting point of quantum mechanics in its modern form, and the idea of quantized energy. Hence, understanding the limitations of the photoelectric effect and the absorption of a photon by a material is crucial in evaluating the interactions between the x-rays and the molecules in the D region of the ionosphere. X-rays, and their carriers, photons, are by no means affected by the magnetosphere of the Earth, since they are not electrically charged. Thus, the only significant interaction between the incoming X-rays and the atmosphere occurs in the D region of the ionosphere which is located at the thermosphere of the atmosphere.⁷ The incoming X-rays often collide with the atoms, and their electrons, located in the D region which is highly famous with the high density of electrons it includes. Hence, as the X-rays collide with N₂ and O₂ atoms, the energy coming from the X-rays are absorbed by the atom, and as a result, an electron is set free with a specific amount of kinetic energy. This is known as the photoelectric effect, and hence is important in evaluating the hypothesis since the increasing energy of the incoming photons does not specifically cause an increase in the emission of the electrons of a molecule after a certain frequency threshold.⁸ However, it should be noted that the intensity of the incoming X-rays are to be taken into account as a constant, since the increase in the intensity of the X-rays is linearly correlated with the amount of electrons emitted.⁸

The Effect of Quantized Energy on the Amount of Emitted Electrons:

The calculation of the energy of an emitted electron via the photoelectric effect is made by the equation:⁹

$$K = hf - \Phi$$

In which K is the maximum kinetic energy that the electron can possess after being set free from the atom of the material, h is the Planck constant, f is the frequency of the incoming radiation, and Φ is the work function, which denotes the required energy needed to tear off an electron from the specified atom. Work function is a function of the material, and is higher in nonmetals compared to metals.⁹ Recalling the approved postulations of the photoelectric effect, one can see that the quantized energy carried by singular photons can be only used in the tearing off of one electron and that after a threshold frequency, which is also a function of the specified material, the value of the frequency, and thus, the wavelength of the specified radiation is not accounted regarding the

number of electrons emitted from a set of materials. The excess energy after subtracting the work function from the incoming photon, energy only affects the kinetic energy of the emitted electrons, and hence does not have any significant effect on the number of liberated electrons, as shown in Figure 2.

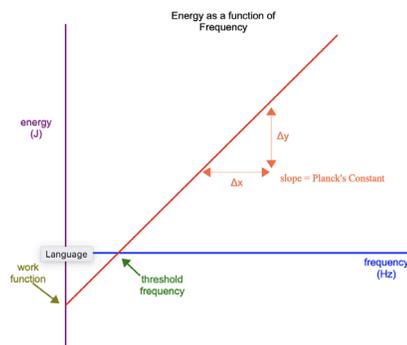


Figure 2: Energy as a function of frequency.²

This is indeed crucial in our understanding of the hypothesis and the x-ray interactions with the ionosphere. In the hypothesis we established, we consider the energy of an incoming beam, and we are trying to determine whether it is possible for the incoming x-rays to reach the Earth's crust and its inhabitants after a critical threshold energy level. However, our hypothesis does not specifically account for the intensity of the incoming beam, since it is also a major observational problem that is yet to be resolved by the scientific community. The minor interactions of the X-ray beams with the outermost parts of the atmosphere cause the diffraction and scattering of beams before entering the ionosphere and therefore prevent us from setting a specific conclusion regarding a critical threshold energy value for a solar flare that could be then seen as a danger threshold for the life on Earth. Briefly, the intensity of the incoming X-ray beams cannot be specified and vary significantly from their predicted source values. At this point, it is needless to say that the hard x-rays that this research concentrates on have wavelengths high above the threshold frequency needed to enable the photoelectric effect on the N and O molecules of the atmosphere.¹⁰

■ Conclusion

The research we have conducted over the course of the last two months allowed us to examine the interaction between the X-rays emitted through the solar flare activities and respected layers of the atmosphere of the Earth, namely the thermosphere and the ionosphere, especially the ionospheric D region. In the end, we have reached the conclusion that it is highly unlikely to determine a critical threshold energy level for the solar flares that enable the hard X-ray beams emitted from them to reach the Earth's crust, due to the complex diffraction and scattering patterns of the X-ray beams and the fact that as the intensity of the beam is kept as constant, the amount of energy a beam possesses does not affect the number of electrons ionized. Hence, it should be noted that further specifications and enhancements could be made by rigorous mathematical analyses.

■ Acknowledgements

We would like to thank our physics teacher Muhsin Erhan for his endless support to find our inspiration and motivation for the paper we successfully completed.

■ References

1. Tandberg-Hanssen, E.; Emslie, A. G. *The Physics of Solar Flares*; Cambridge University Press: Cambridge, 2009.
2. Kontar, E. P.; MacKinnon, A. L.; Schwartz, R. A.; Brown, J. C. Compton backscattered and primary X-rays from solar flares: Angle Dependent Green's function correction for photospheric albedo. <https://www.aanda.org/articles/aa/abs/2006/06/aa3672-05/aa3672-05.html> (accessed Apr 17, 2022).
3. Kane, S. R.; McTiernan, J. M.; Hurley, K. Multispacecraft observations of the hard X-ray emission from the giant solar flare on 2003 November 4. <https://ui.adsabs.harvard.edu/abs/2005A%26A...433.1133K/abstract> (accessed Apr 17, 2022).
4. GMS: Ionosphere Graphics. <https://svs.gsfc.nasa.gov/12960> (accessed Apr 17, 2022).
5. Center for Science Education. <https://scied.ucar.edu/learning-zone/atmosphere/thermosphere> (accessed Apr 17, 2022).
6. HOLLINGWORTH, J. Structure of the Ionosphere. *Nature* 1934, 134 (3386), 462–462.
7. Sutherland, L. C.; Bass, H. E. Atmospheric Absorption in the Atmosphere up to 160 Km. *The Journal of the Acoustical Society of America* 2004, 115 (3), 1012–1032.
8. The Editors of Encyclopaedia. Photoelectric effect. <https://www.britannica.com/science/photoelectric-effect> (accessed Apr 17, 2022).
9. PRATT, R. H.; RON, A. K. I. V. A.; TSENG, H. K. Atomic Photoelectric Effect above 10 Kev. *Reviews of Modern Physics* 1973, 45 (2), 273–325.
10. Watanabe, K. Ultraviolet Absorption Processes in the Upper Atmosphere. *Advances in Geophysics* 1958, 153–221.
11. Lesson 34: Photoelectric effect graphs - studyphysics. http://www.studyphysics.ca/2007/30/07_emr/34_photo_graphs.pdf (accessed Apr 16, 2022).
12. A parameterized model of x-ray ... - wiley online library. <https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2018RS006666> (accessed Apr 16, 2022).
13. Švestka, Z. Solar Flares. 1976.

■ Authors

Can Erol is a student at American Collegiate Institute, Izmir, that is truly interested in physics and mathematics, and an avid learner of science looking forward to expanding his research interests. He currently wants to double major in physics and mathematics.

Evren Sanlı is a student at the American Collegiate Institute in Izmir, Turkey. He is deeply interested in physics, especially particle and astrophysics, and is hoping to widen his understanding in these fields through future research. Currently, he is planning to double major in mechanical engineering and econometrics.