

# **TDK dolgozat**

**Készítette:**

**Szokol Péter**

Debreceni Egyetem  
Informatikai Kar  
Számítógéptudományi Tanszék

**A genetic algorithm with interval fitness method for the  
minimum vertex cover problem**

**(Egy genetikus algoritmus intervallumos fitness függvénnel a  
minimális lefedő csúcshalmaz problémára)**

**Témavezető:**

**Dr. Nagy Benedek**  
egyetemi docens

**Készítette:**

**Szokol Péter**  
2. évfolyam MSC

Debrecen  
2011

# 1 Introduction

A vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set. Minimum vertex cover means that the size of the vertex cover set is minimal. Finding the minimum vertex cover is a classical NP-complete problem. As such, genetic algorithm (GA) is an ideal method for the minimum vertex cover problem. Genetic algorithms are based on the idea of Charles Darwin's evolution. GA is a heuristic search algorithm, it finds a relatively good solution in a short amount of time, and it can be stopped any time, it will always have a (relatively) good solution [8, 9].

Every GA has a population of candidate solutions. For the sake of simplicity, from now on we call these solutions, instead of candidate solutions. Each of these solutions have a genome, the entirety of their hereditary information. The genome contains information about the solution. The algorithm must calculate the fitness of the solutions, so the next generation can inherit the genomes of the best solutions. A child can inherit genes from one or more parents. There is also a small chance for mutation. The new generation builds their own solutions based on their genomes, then the algorithm measures the fitness of each solution again, and so on.

GAs give a lot of freedom to the programmer. Creating the structure of the genome is the first step. The most obvious way to do this for the minimum vertex cover problem, is to create a gene for every node. The gene can have two values: true or false. True means that the corresponding node is in the vertex cover set, false means that the node is not in the vertex cover set. We present a new idea, genotype and phenotype in GAs. We adapt these concepts from biology, and it results a better variety of the gene pool. It is possible that the vertex cover set is not covering the graph. In this case, the genotype would not be a vertex cover, but the algorithm can add nodes to the vertex cover so the phenotype becomes one. With the fast, and effective greedy algorithm, the phenotype will be significantly better. Creating the phenotype can be viewed as a part of the fitness function. The fitness function assigns fitness values to the genomes. Fitness value indicates how good the solution is, i.e. how close it is to an ideal, or optimal solution. Inheritance is an other main part of GAs, and there are lots of opportunities here as well. We present a new idea, the interval fitness method. We apply to the GAs the idea of intervals [5, 3]. It means that the child inherits gene sequences from one or more parents. If a parent phenotype has a good fitness value on an interval, then the genes of the parent on this interval will be added to the child's genome. It is useful if it is easy to distinguish the different characteristics of a solution, and these characteristics are independent, or dependent subtly from each other. For the vertex cover problem, different characteristics of a graph is not very straightforward, but we present some idea.

The rest of the paper is organized as follows: we first introduce the problem, the mathematical representation, and the structure of a genome. In Section III, we describe four ideas for determining the partitions of a graph. In Section IV, we explain the Greedy error correction, a method to repair invalid vertex covers. We present a GA for determining

the partitions of a graph in Section V, then describe the new generation of the minimum vertex cover GA in Section VI.

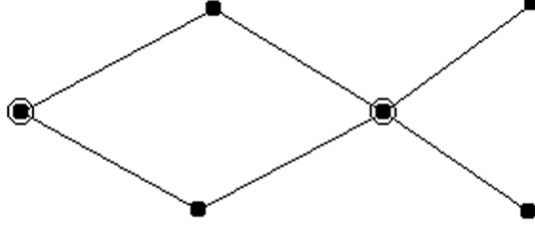


Figure 1: A graph with its minimum vertex cover

## 2 Minimum vertex cover, and the structure of a genome

Graph  $G$  with  $n$  vertices consists of a set of vertices  $V$ . The order of the vertices in  $V$  is arbitrary. The aim is to find the minimum vertex cover  $C$ , a set of vertices, where  $C$  is a subset of  $V$ . Each edge of  $G$  is incident to at least one vertex in  $C$ . The size of  $C$  is minimal. For an example of a simple graph and its minimum vertex cover see Figure 1.

The population of the GA consists of a set of genomes  $\langle F, I, K \rangle$ , where  $F$  is the fitness of the genome,  $I = \{I_1, I_2, \dots, I_m\}$  is a set of interval fitness values of the genome, and  $K = \{K_1, K_2, \dots, K_n\}$  is the information set of the vertices.

The fitness  $F$  is a number, it is calculated by the fitness function, and it represents the overall fitness of the solution. Some GAs make no difference between the genotypes and phenotypes. Since the GA shown in this paper does make a difference, we must declare that this fitness represents the fitness of the phenotype. In other words, first we use the greedy error correction on the solution of a genome, then we execute the fitness function on the modified solution. A simple fitness function would be:  $|C| + E * x$ , where  $E$  is the error, i.e. the number of edges that are not incident to any vertex of  $C$ , and  $x$  is the weight of the error. With the greedy error correction however,  $E = 0$ , so the function is:  $|C|$ .

Let  $P$  denote partitions of  $V$ , where  $P_j (j = 1, 2, \dots, m)$ . If  $P_j = a_1, a_2, \dots, a_k$ , then  $I_j$  is the interval fitness of the solution on the subgraph of  $V_{a_1}, V_{a_2}, \dots, V_{a_k}$ .  $P$  is the same for every genome, and every generation. If  $K_i = \text{true}$ , then vertex  $V_i$  is in the cover set  $C$  proposed by the genotype of this genome.

For an example, see Figure 2.

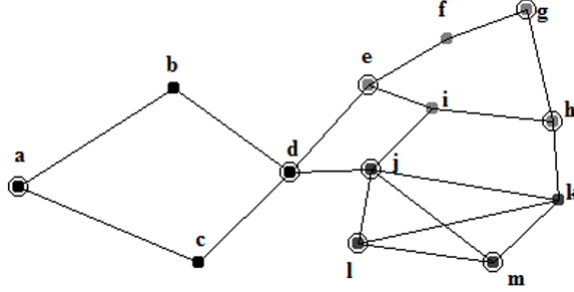


Figure 2: A graph where  $P=[(a, b, c, d), (e, f, g, h, i), (j, k, l, m)]$ , and one of the possible solutions, where  $F=8(=|C|)$ ,  $I=[2,3,3]$ , and  $K=[1,0,0,1,1,0,1,1,0,1,0,1,1]$

### 3 Determining the partitions

The interval fitness function works the best if every  $P_j$  contains closely related vertices. There are several methods to determine  $P$ .

- Random partitions.
- Divide by the time of creation of the vertices.
- Manually determine the partition of each vertex.
- GA, with a fitness function that calculates fitness by the dissimilarity of the neighbors of each vertex.

The partitioning does not have to be optimal. It can help even if the partitioning is random, since as it will be detailed later, for every generation, only one genom will be created with the interval fitness method. For example, if the population size is constantly 200, then 199 child will evolve normally, only 1 child (super-child) will be created from the best intervals. Even if the partitioning is random, there is a good chance that the super-child will be successful. If that happens, the quality of the gene pool will significantly increase. Dividing by the time of creation is an idea only slightly better than the random partitioning. It is based on the assumption that there is a connection between the vertex structure and the time of creation of the vertices. GA seems to be the best choice, since this subproblem is hard, but the solution does not have to be optimal, only close to optimal. Creating the partitions should not take much time from the actual work, so it has to be fast too. Later in this paper, we will present the idea of such a GA in detail.

### 4 Greedy error correction

There is a problem with the simple GA approach: most of the time, it is not even covering the graph. Finding only a cover set with pure natural selection and random mutations

is difficult for a simple GA, but the aim is to find the minimal cover set, so the task is even more complicated. Fortunately, there are easy methods to create a cover set from an imperfect cover set.

The simplest, and fastest one is to check every edge. If an edge is not incident to any vertex in  $C$  (unstable edges), then add one of the vertices of this edge to  $C$ . Some nodes may be redundant, these unnecessary nodes can be removed [4].

We analyze the use of a greedy method, it checks every vertex that is not in  $C$  first (unselected vertices), but adds them to  $C$  only if they have the highest number of unstable edges incident to them. In the next cycle, it checks every unselected vertices again, until there are no more unstable edges. This method is much slower, because it runs for every genome in every generation, but our tests showed that the algorithm gets significantly better with the greedy error correction.

This correction works if the genome selects too few vertices. It is possible to correct those genomes that selects too many. Invert correction: if the neighbors of a selected vertex  $X$  are all selected vertices, then  $X$  can be removed from  $C$ .

Combining these ideas, it is possible to further improve these methods. At the last cycle of the greedy correction, there are no unselected vertices with multiple unstable edges. In this cycle, the choice of which vertex of an unstable edge should be selected is arbitrary. In some case however, this choice has an impact on the final solution, because of the final invert correction.

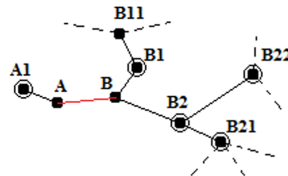


Figure 3: A subgraph with the last cycle of the greedy algorithm where each vertex has a maximum of one unstable edge

Take the subgraph shown in Figure 3.

Between vertices  $A$  and  $B$ , there is an unstable edge. Since the greedy correction is at its last cycle, the algorithm can be sure that there are no vertices with more than one unstable edges. Suppose the algorithm first discovers vertex  $A$ . Vertex  $A$  has one unstable edge. The other vertex of this unstable edge is vertex  $B$ . It is granted that  $B$  has no other unstable edges, only this one.

Let us introduce a new concept, the compatible neighbor.

**Definition 1** *Definition of compatible neighbor: if  $B$  and  $B_2$  are neighbor vertices in a graph,  $B_2$  is selected, and beyond  $B$ ,  $B_2$  has only selected neighbors, then  $B_2$  is a compatible neighbor of  $B$ .*

The algorithm should decide like this:

If  $B$  has more compatible neighbors than  $A$ , then select  $B$ , else select  $A$ .

Compatible neighbors are good, because the invert correction will deselect them if all their neighbors are selected. Since  $A$  has one compatible neighbor ( $A_1$ ), and  $B$  has one compatible neighbor too ( $B_2$ ), in this case, the choice is truly arbitrary, the algorithm will select  $A$ .

Lamarckian inheritance is the idea that characteristics developed during an organism's lifetime (the greedy error correction), can be inherited. The question is, should the algorithm follow this idea, and pass through the genes reflecting the solution after the correction, or not.

Since the greedy error correction is not a stochastic method, it creates the same phenotypes for a given genotype. If the child inherits every gene of a parent's genotype perfectly, it will develop the same phenotype as the parent did. Furthermore, it is possible that the algorithm creates the same phenotype for different genotypes. That is why, using the Lamarckian inheritance could lead to a less varied gene pool, which is not desirable.

## 5 Genetic Algorithm for partitioning

The first step for both the genetic algorithm and the random method is to determine the number of partitions. 10 partitions for a graph with 10 vertices would result the same as there would not be groups at all, but little number of partitions for a big graph is not ideal either. We use the following function:

$$|P| = (|V|^{0.7})/3$$

The GA for partitioning has to be fast, so the genome structure is not as straightforward as the genome structure of the main GA.

Number of genes =  $|P|$ . Every gene denotes the base vertex of the corresponding partition. In the initial population, every gene gets a random value  $x$ , where  $0 \leq x < |V|$ .

The fitness function first simulates a vertex-conqueror game.

Rules of the vertex-conqueror game are:

Initially, every partition has one vertex, their base vertex, based on the genome. Every partition has a set of conquered vertices ( $CV$ ), initially it is their base vertices. If a partition has a base vertex already taken, then the new partition conquers the base vertex of the previous partition, and the previous partition leaves the game.

In every round, each vertex of the  $CV$  of each partition expands, meaning that they conquer every neighbor vertices, except those already conquered by any partition.

When no partition can expand anymore, the first partition conquers every non-conquered vertices if there are any left (may happen in case of not connected graph). The game ends here: the phenotype is ready.

The fitness value of a phenotype is as follows:

Let  $g$  be the number of neighbor vertices that belong to one group, and  $w$  is the number of neighbor vertices that belong to different groups.

Let  $a = \lfloor g/(g + w) \rfloor$ , this is a number between 0 and 1, as  $g$  and  $w$  are both positive numbers.

Let  $cmax = \max(|CV|)$ , the size of the largest partition.

Let  $cmin = \min(|CV|)$ .

Let  $b = cmin/cmax$ , a number between 0 and 1, as  $cmin \leq cmax$ , and both are positive numbers.

The fitness value is:  $(a * b * b) * 10000$ .

For every new generation, the GA for partitioning uses elitism, fitness proportionate selection, one-point crossover, and mutation, we talk about these as we detail them in the next session.

## 6 Next generation of the minimum vertex cover algorithm

The first  $x$  genomes of the next generation is  $x$  genomes of the first population with the best  $F$  values. This is called elitism, it ensures that the best solutions will always stay in the population, with elitism, there is no chance of losing quality due to mutation.

The next genome of the next population is super-child  $S$ .

For every  $i = 1, 2, \dots, m$ , where  $m = |P|$ , let genome  $B$  be the genome of the previous generation with the best value  $I_i$ . The fitness function for  $I_i$  is the number of selected vertices on partition  $P_i$ .  $K_j^S = K_j^B$  for every  $j = P_{i_1}, P_{i_2}, \dots, P_{i_q}$ , where  $K^S$  is the set  $K$  of the super-child,  $K^B$  is the set  $K$  of  $B$ ,  $P_{i_x}$  is the  $x$ th member of  $P_i$ , and  $q = |P_i|$ .

In other words, the super-child inherits the best genes of the population for every interval, from one, two, or more parents.

The rest of the population are created as follows:

- Two parent are selected from the previous population, using fitness proportionate selection:
  - Let  $A$  be the arithmetic mean of the  $F$  values of the population.
  - If value  $F$  of a genome is better than  $A$ , then this genome will be a parent with a probability depending on  $F - A$
- Let  $O_1$  and  $O_2$  two genomes generated from the parents.  $O_1$  completely inherits the genes of the first parent, or partially inherits the genes of the first parent to a random point  $x$ . If  $O_1$  inherited partially, then it inherits the missing genes beyond  $x$  from the other parent.  $O_2$  inherits the remaining genes of the parents. (One-point crossover).
- Each genes of  $O_1$  and  $O_2$  has a chance to mutate. If a gene in  $K$  mutates, then its value changes from false to true, or from true to false. For example:

$K = [0, 0, 0, 1, 1]$ . If the mutation chance is 100%, then after the mutation  $K$  is  $[1, 1, 1, 0, 0]$ .

The new generation has the same size of population as the previous one had. The old generation dies, the next generation takes their place.

Naturally, after the next generation is ready, the algorithm updates the  $F$  and  $I$  values of each genome.

## 7 Example

For the sake of this example, let's assume that there is a population of 3 solutions for a graph, and the partitions of the graph is known ( $[1, 2, 3, 4], [5, 6, 7]$ ). The order of the graph vertices is shown on Figure 4. The first generation is also known.

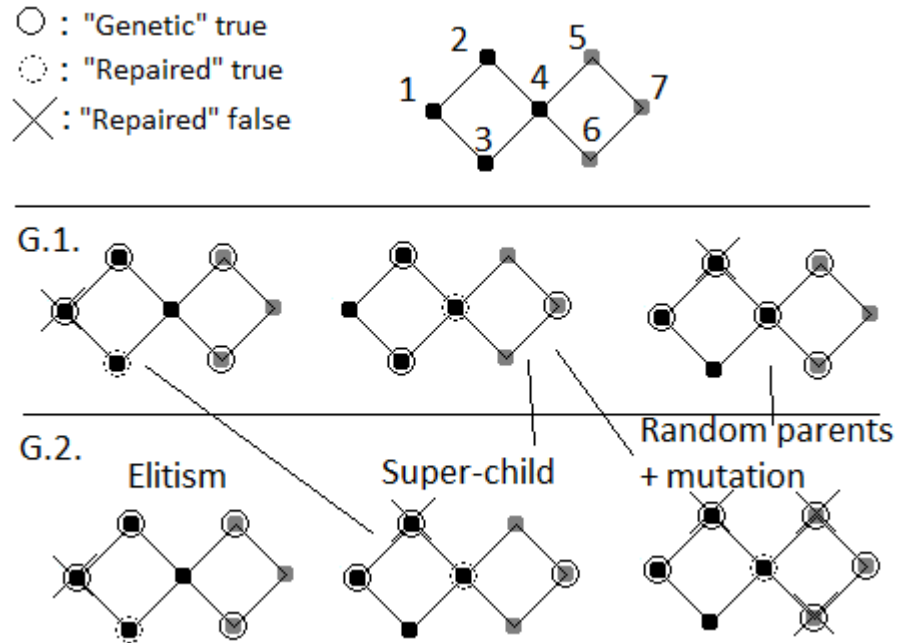


Figure 4: A simplified illustration of the algorithm

To repair the first solution, the algorithm has to complete the graph cover. The algorithm is greedy, but there is only one edge without cover vertices. The edge is between vertex 3 and 4. Selecting vertex 3 would result one unnecessary cover vertex, selecting vertex 4 would result one unnecessary cover vertex too, the algorithm selects vertex 3. Vertex 1 becomes unnecessary. Therefore this cover has 4 vertices.

Solution 2 has edges without cover vertices too. Since the algorithm is greedy, vertex 4 is selected, because it will cover 2 uncovered edges. Solution 3 is a cover set, but it has one unnecessary vertex.

The final cover set size of each solution in the first generation is 4. But on the black interval solution 1 and 3 are better, and on the grey interval solution 2 is better.

The first solution of the new generation is the first best solution of the previous generation. Since they all had the same fitness value, solution 1 is the elite.

The second solution is the super child, with the best intervals. It inherits the genes of the black interval of solution 1, and the genes of the grey interval of solution 2. Vertex 4 is selected because it will cover 3 uncovered edges. Vertex 2 is unnecessary. Therefore this cover has 3 vertices.

The third solution inherits genes from two parents randomly, then mutates. It has one edge without cover vertices. The algorithm selects vertex 4, because it will result 3 unnecessary vertices. This cover has 3 vertices too.

## 8 Experimental Results

We compare our algorithm with a fast greedy algorithm, a normal GA with simple error correction, and TVCA. TVCA of Ashay Dharwadker is a great algorithm for the vertex cover problem [10]. It always finds the cover set with size of a given number  $k$ , if there is any. TVCA first fills  $C$  (the cover set) with the members of  $V$  (vertices of the graph). The algorithm then finds the elements it can take out from  $C$ . But this is not the only difference between the two algorithms: TVCA is a non-stochastic algorithm.

Our algorithm (MGA) is fast and accurate in almost any cases, except the Witzel Graph [10], where almost every node is connected to every node, making the intervals meaningless.

Graph 1 (Vertices: 300, Edges: 354, M.V.C. Size: 132)

| Algorithm | Time 1 | Size 1 | Time 2 | Size2 |
|-----------|--------|--------|--------|-------|
| TVCA      | 6      | 132    |        |       |
| Greedy    | 0      | 135    |        |       |
| GA        | 1      | 132    |        |       |
| MGA       | 1      | 132    |        |       |

Graph 2 (Vertices: 880, Edges: 1030, M.V.C. Size: 431 or less)

| Algorithm | Time 1 | Size 1 | Time 2 | Size2 |
|-----------|--------|--------|--------|-------|
| TVCA      | 600    | 431    |        |       |
| Greedy    | 0      | 441    |        |       |
| GA        | 10     | 436    | 120    | 433   |
| MGA       | 5      | 433    | 20     | 432   |

Graph 3 (Vertices: 1000, Edges: 2000, M.V.C. Size: 558 or less)

| Algorithm | Time 1 | Size 1 | Time 2 | Size2 |
|-----------|--------|--------|--------|-------|
| TVCA      | 480    | 559    |        |       |
| Greedy    | 0      | 570    |        |       |
| GA        | 10     | 564    | 100    | 560   |
| MGA       | 5      | 563    | 20     | 559   |

Witzel Graph (Vertices: 450, Edges: 17827, M.V.C. Size: 420)

| Algorithm | Time 1 | Size 1 | Time 2 | Size2 |
|-----------|--------|--------|--------|-------|
| TVCA      | 60     | 420    |        |       |
| Greedy    | 0      | 428    |        |       |
| GA        | 40     | 425    |        |       |
| MGA       | 50     | 425    |        |       |

## 9 Conclusions

Our modified GA always yields at least the same solution as a greedy algorithm, but most of the time our algorithm is better. The reason is that the first generation has a blank genome, so the greedy error correction is the same as a greedy algorithm. The modified GA is a big improvement for the GA. In case of big graphs, it converges much faster with the interval fitness method which can be viewed as a problem reduction approach. At the same time, the algorithm focuses on the whole problem as well. With the error correction, almost all member of the population has a better fitness, and the genetic variation is much better too. But the biggest advantage of the error correction is obviously the fact that it mixes the fast and efficient greedy method with the heuristic GA, keeping the advantage of both approach. In case of a difficult graph, the basic GA can not approach the efficiency of a simple greedy algorithm, not even with a lot of time given.

With greedy error correction, it is possible that the genome of the optimal solution is blank, meaning every element of set  $K$  has false value. Since the greedy error correction step itself is a good method for the problem as well, it is a good thing if the GA cooperates with the greedy method, by only selecting genes that improve the fitness. For this, the GA has to start with a blank genome. Because in the first step each genome is blank, there is no need to calculate the fitness of each genome in the first generation, one is enough.

GAs tend to converge towards local optimum. To avoid this, it is worth to experiment with multiple populations. Since members of different populations have no chance to mix their genetic information, multiple populations develop independently, and it is possible that each population converges towards different local optimums. It is also means that the different populations can use different processors.

An other idea to solve the problem of local optimum is to restart every generation after a given time or generation. Obviously the algorithm has to save the best solution, but the new population must not remember that genome, it has to start with blank genomes.

## References

- [1] Shaohua Li, Jianxin Wang, Jianer Chen and Zhijian Wang, *An Approximation Algorithm for Minimum Vertex Cover on General Graphs*, Proceedings of the Third International Symposium on Electronic Commerce and Security Workshops(ISECS 10) Guangzhou, P. R. China, 29-31, July 2010, pp. 249-252 (2010)
- [2] Jack Coldridge and Martyn Amos, *Genetic Algorithms and the Art of Zen*, Bio-Inspired Computing: Theories and Applications (BIC-TA) (2010)
- [3] Benedek Nagy, Sándor Vályi, *Interval-valued computations and their connection with PSPACE*, Theoretical Computer Science - TCS 394/3, 208-222. (2008)
- [4] Martin Pelikan, Rajiv Kalapala, Alexander K. Hartmann. *Hybrid Evolutionary Algorithms on Minimum Vertex Cover for Random Graphs*, GECCO '07 Proceedings of the 9th annual conference on Genetic and evolutionary computation ACM New York, NY, USA 2007 (2007)
- [5] Benedek Nagy, *A general fuzzy logic using intervals*, 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest, Hungary, 613-624. (2005)
- [6] Ketan Kotecha, Nilesh Gambhava, *A Hybrid Genetic Algorithm for Minimum Vertex Cover Problem*, In Prasad, B. (Ed.), The First Indian International Conference on Artificial Intelligence (2003)
- [7] Ali Karci, Ahmet Arslan, *Bidirectional evolutionary heuristic for the minimum vertex-cover problem*, Computers and Electrical Engineering, Volume 29, Number 1, January 2003 , pp. 111-120(10), Publisher: Elsevier (2003)
- [8] Álmos Attila, Dr. Horváth Gábor, Dr. Várkonyiné dr. Kóczy Annamária, Győri Sándor, *Genetikus algoritmusok*, Typotex Kiadó (2003)
- [9] Goldberg, David E, *Genetic Algorithms in Search Optimization and Machine Learning.*, Publisher: Addison Wesley. (1989)
- [10] [http : //www.dharwadker.org/vertex\\_cover/](http://www.dharwadker.org/vertex_cover/)

## 10 Saját munka

Nagy Benedek intervallumos ötletét próbáltam ki a minimális lefedő csúcshalmaz algoritmusom javítására. Az algoritmus további javításán gondolkozva rájöttem, hogy a genetikus algoritmust segíteni kell egy javító-eljárással, így kifejlesztettem a mohó-javítást is a problémához. Később találtam egy cikket [4], melyben a szerzők hasonló ötlettel

álltak elő, de ők úgy ítélték meg, hogy elég egy egyszerűbb javító további fejlesztések nem javítanak jelentősen az eredményt, viszont számításigényesebbé válna az algoritmus. A tesztjeim során azonban nekem úgy tűnt, hogy érdemes bonyolítani a javító-algoritmust, mert egyértelműen javul a jó megoldások megtalálásának gyorsasága.

Az intervallumok meghatározására több ötletem is volt, az egyik közülük a genetikus algoritmus használata, tehát ehhez a problémához is készítettem egy genetikus algoritmust. A témavezetőm javaslatára elkezdtem keresni mások által készített algoritmusokat a problémához, ekkor találtam meg a [4, 10] munkákat, és elkészítettem egy mohó algoritmust is.

Az algoritmusommal később a párhuzamos programozást is teszteltem, amivel sikerült gyorsabb eredményt elérni.

## 11 Köszönetnyilvánítás

Köszönetet szeretnék mondani Nagy Benedeknek, hogy ötleteivel, támogatásával segítette a munkámat.