IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 1, No 3, January 2013
ISSN (Print): 1694-0784 | ISSN (Online): 1694-0814
www.IJCSI.org

Analysis of the impact of parameters values on the Genetic Algorithm for TSP

Avni Rexhepi[1], Adnan Maxhuni[2], Agni Dika[3]

[1,2,3] Faculty of Electrical and Computer Engineering, University of Pristina, Kosovo

Abstract

Genetic algorithms (GAs) are multi-dimensional and stochastic search methods, involving complex interactions among their parameters. Researchers have been trying to understand the mechanics of GA parameter interactions by using various techniques. It still remains an open question for practitioners as to what values of GA parameters (such as population size, choice of GA operators, operator probabilities, and others) to use in an arbitrary problem.
Genetic algorithm (GA) parameters are explored to minimize the time needed to find a solution. The basic GA code stays the same throughout the entire system. Variable parameters include mutation rate, crossover rate, crossover operator, number of generations, population size, etc. When an optimization problem is encoded using genetic algorithms, one must address issues of population size, crossover and mutation operators and probabilities, stopping criteria, selection operator and pressure, and fitness function to be used in order to solve the problem. This paper tests a relationship between (1) size of initial population, (2) mutation probability, and (3) number of generations in runs of Genetic Algorithm for solving the TSP.
TSP is an NP hard problem, so using Genetic Algorithm we can find a solution on reasonable amount of time. In this paper we describe the results of the solution for the Traveling Salesman Problem (TSP) for Kosovo municipalities, using the genetic algorithm, with different settings for the parameters of the Genetic Algorithm [12].

Keywords: *Genetic Algorithms, TSP, Parameter Selection, Initial Population, Crossover, Mutation*

1. Introduction

TSP is a problem, where traveling salesman wants to visit each of a set of cities exactly once, starting from hometown and returning to his hometown. His problem is to find the shortest route for such a trip. TSP has a model character in many branches of Mathematics, Computer Science, Operations Research, etc. Linear programing, heuristics and branch and bound which are main components for the most successful approaches to hard combinatorial optimization problems, were first formulated for the TSP and used to solve practical problem instances in 1954 by Dantzig, Fulkerson and Johnson.
When the theory of NP-completeness developed, the TSP was one of the first problems to be proven NP-hard by Karp in 1972. New algorithmic techniques have first been developed for or at least have been applied to the TSP to show their effectiveness. Such examples are branch and bound, Lagrangean relaxation, Lin-Kernighan type methods, simulated annealing, etc. [3].
Representation model is: Let Kn=(V_n, E_n) be the complete undirected graph with n=|V_n| nodes and m=|En|= $\binom{n}{2}$ edges. An edge e with endpoints *i* and *j* is also denoted by *ij*, or by (*i*,*j*). We denote by R^{En} the space of real vectors whose components are indexed by the elements of E_n. The component of any vector $z \in R^{En}$ indexed by the edge *e=ij* is denoted by z_e, z_{ij}, or *z(i,j)*.
Given an objective function $c \in R^{En}$, that associates a "length" c_e with every edge *e* of K_n, the symmetric traveling salesman problem consists of finding a Hamiltonian cycle such that its *c*-length (the sum of the lengths of its edges) is as small as possible.
Of special interest are the Euclidean instances of the traveling salesman problem. In these instances the nodes defining the problem correspond to points in the two-dimensional plane and the distance between two nodes is the Euclidean distance between their corresponding points.

More generally, instances that satisfy the triangle inequality, i.e., $c_{ij} + c_{jk} \geq c_{ik}$ for all the three distinct *i,j* and *k*, are of particular interest.
For our case, we consider the locations of the cities/municipalities in Kosovo map as nodes of the graph. For to do this, we take their geographic coordinates and then based on that, we calculate their position in our map scaled to a smaller size, for to calculate the real positions and distances, by using the real life values for distances in kilometers between the cities.

2. Genetic Algorithms

Genetic Algorithms (GA) [1,2] are computer algorithms that search for good solutions to a problem within a large number of possible solutions. They were proposed and developed in the 1960s by John Holland, his students, and his colleagues at the University of Michigan. These computational paradigms were inspired by the mechanics of natural evolution, including survival of the fittest, reproduction, and mutation. These mechanics are well suited to resolve a variety of practical problems, including computational problems, in many fields. Some applications of GAs are optimization, automatic programming, machine learning, economics, immune systems, population genetic, and social system.
GAs have been successfully applied to many problems of business, engineering, and science. Because of their operational simplicity and wide applicability, GAs play an important role in computational optimization and operations research [6].
The genetic algorithm transforms a population (set) of individual objects, each with an associated fitness value, into a new generation of the population using the Darwinian principle of reproduction and survival of the fittest and analogs of naturally occurring genetic operations such as crossover (sexual recombination) and mutation. Each individual in the population represents a possible solution to a given problem. The genetic algorithm attempts to find a very good (or best) solution to the problem by genetically breeding the population of individuals over a series of generations.

2.1 Basic elements of GAs

Most GAs methods are based on the following elements: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and random mutation of new offspring. The chromosomes in GAs represent the space of candidate solutions. Possible chromosomes encodings are binary, permutation, value, and tree encodings. GAs require a fitness function which allocates a score to each chromosome in the current population. Thus, it can calculate how well the solutions are coded and how well they solve the problem [2].
The selection process is based on fitness. Chromosomes that are evaluated with higher values (fitter) will most likely be selected to reproduce, whereas, those with low values will be discarded. The fittest chromosomes may be selected several times, however, the number of chromosomes selected to reproduce is equal to the population size, therefore, keeping the size constant for every generation. This phase has an element of randomness just like the survival of organisms in nature. The most used selection methods, are roulette-wheel, rank selection, steady-state selection, and some others. Moreover, to increase the performance of GAs, the selection methods are enhanced by elitism. Elitism is a method, which first copies a few of the top scored chromosomes to the new population and then continues generating the rest of the population. Thus, it prevents losing the few best found solutions.
Crossover is the process of combining the bits of one chromosome with those of another to create an offspring for the next generation that inherits traits of both parents.
Mutation is performed after crossover to prevent falling all solutions in the population into a local optimum of solved problem.
So, general outline of basic GA is:
1. Start: Randomly generate a population of N chromosomes.
2. Fitness: Calculate the fitness of all chromosomes.
3. Create a new population:
    a. Selection: According to the selection method select 2 chromosomes from the population.
    b. Crossover: Perform crossover on the 2 chromosomes selected.
    c. Mutation: Perform mutation on the chromosomes obtained.
4. Replace: Replace the current population with the new population.
5. Test: Test whether the end condition is satisfied. If so, stop. If not, return the best solution in current population and go to Step 2.

Each iteration of this process is called generation.
The genetic algorithm object determines which individuals should survive, which should reproduce, and which should die. It also records statistics and decides how long the evolution should continue. A typical genetic algorithm will run forever, so we must build functions for specifying when the algorithm should terminate. These include terminate-upon generation, in which you specify a certain number of generations for which the algorithm should run, and terminate-upon-convergence, in which you specify a value to which the best-of-generation score should converge. One can customize the termination function to use own stopping criterion and must tell the algorithm

when to stop. Often the number-of generations is used as a stopping measure, but you can use goodness-of-best-solution, convergence-of-population, or any problem-specific criterion if you prefer.
There are some flavors of genetic algorithms. For example, the first is the standard 'simple genetic algorithm' described by Goldberg in his book [2]. This algorithm uses non-overlapping populations and optional elitism. Each generation the algorithm creates an entirely new population of individuals. The second is a 'steady-state genetic algorithm' that uses overlapping populations. In this variation, you can specify how much of the population should be replaced in each generation. The third variation is the 'incremental genetic algorithm', in which each generation consists of only one or two children. The incremental genetic algorithms allow custom replacement methods to define how the new generation should be integrated into the population. So, for example, a newly generated child could replace its parent, replace a random individual in the population, or replace an individual that is most like it. The fourth type is the 'deme' genetic algorithm. This algorithm evolves multiple populations in parallel using a steady-state algorithm. Each generation the algorithm migrates some of the individuals from each population to one of the other populations.
The base genetic algorithm class contains operators and data common to most genetic algorithms.
The genetic algorithm contains the statistics, replacement strategy, and parameters for running the algorithm. The population object, a container for genomes, also contains some statistics as well as selection and scaling operators.
The number of function evaluations is a good way to compare different genetic algorithms with various other search methods[3]. The basic algorithm is as follows:

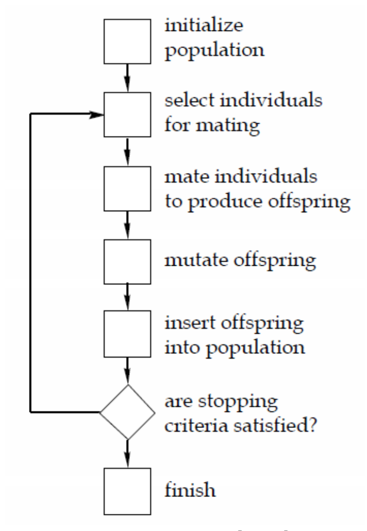
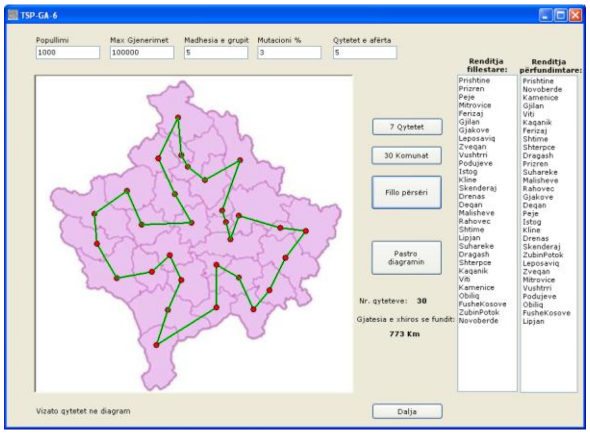

initialize population
select individuals for mating
mate individuals to produce offspring
mutate offspring
insert offspring into population
are stopping criteria satisfied?
finish

Figure 1 – Genetic Algorithm

2.2 TSP

In order to calculate the shortest traveling distance from an initial city, by visiting each one only once and returning to the initial one, we consider the locations as nodes of the graph, in the graph model. In the meantime, the distances between the cities are edges of the graph.
For length of the edges we take inter-city distances in kilometers. We have created a matrix of distances between all the municipalities, where the matrix elements c_{ij} are elements of the square symmetrical matrix, since distance ij is equal to the distance in the other side ji. The diagonal of the matrix will contain zero values, since diagonal elements of the matrix c_{ij}, where $i=j$, will represent the distance of the city to itself, so in fact it will be the traveling distance of zero kilometers, therefore these elements will be equal to $c_{ij}=0$.
By using complete graph in the definition of the TSP, the existence of a feasible solution is guaranteed, while for general graphs deciding the existence of a Hamiltonian cycle is an NP-complete problem. The number of Hamiltonian cycles in K_n, i.e. the size of the set of feasible solutions of the TSP is (n-1)!/2.
The algorithmic treatment of the TSP ensures an approximation algorithm that cannot guarantee to find the optimum, but which is the only available technique to find a good solution to a large problem instances. To assess the quality of a solution, one has to be able to compute a lower bound on the value of the shortest Hamiltonian cycle.
We have built an application in C#, with an image with the small-scaled size of the Kosovo map, with depicted municipality boundaries and locations (Figure 2).
It is possible to select a particular city by clicking on the map and we have also added buttons that make it possible to create locations for the biggest cities and locations for the all municipalities.



Popullimi: 1000
Max Gjenerimet: 100000
Madhesia e grupit: 5
Mutacioni %: 3
Qytetet e afërta: 5

Renditja fillestare:	Renditja përfundimtare:
Prishtine	Prishtine
Prizren	Novoberde
Peje	Kamenice
Mitrovice	Gjilan
Ferizaj	Viti
Gjilan	Kaqanik
Gjakove	Ferizaj
Leposaviq	Shtime
Zveqan	Shterpce
Vushtrri	Dragash
Podujeve	Prizren
Istog	Suhareke
Kline	Malisheve
Skenderaj	Rahovec
Drenas	Gjakove
Deqan	Deqan
Malisheve	Peje
Rahovec	Istog
Shtime	Kline
Lipjan	Drenas
Suhareke	Skenderaj
Dragash	ZubinPotok
Shterpce	Leposaviq
Kaqanik	Zveqan
Viti	Mitrovice
Kamenice	Vushtrri
Obiliq	Podujeve
FusheKosove	Obiliq
ZubinPotok	FusheKosove
Novoberde	Lipjan

Nr. qyteteve: 30
Gjatesia e xhiros se fundit: 773 Km
Vizato qytetet ne diagram

Figure 2 – Screenshot of the application

We used the geographical coordinates of the cities, to calculate their positions. By running the application, we can calculate the shortest traveling distance between the selected cities, by finding a solution of the TSP using a genetic algorithm.

User can set up the values for initial population, maximal number of generations, size of the group, percentage of mutations and number of close cities/locations (used by the algorithm, while finding closest locations).

3. Simulation and Results

We analyzed the results of different cases with different values for the parameters of initial population, probability of mutation and number of generations.

Firstly, we have set the value of the maximal number of generations to 10,000 and we calculated the fitness for the cases where the size of the initial population is: 1000, 5000 and 10000 and for each of these cases, we calculated the results for the mutation rate of: 1%, 3%, 5% and 10% (as in Figures: 3, 4 and 5).

Then, we compared the results for each value of the mutation rate (1%, 3%, 5% and 10%), with different values of the initial population parameter values (1000, 5000 and 10000, as in Figures: 6, 7, 8 and 9).

We repeated this for the values of maximal number of generations of 50,000 and 100,000, too (see Appendix).

In each figure, the y-axis is the value of the fitness and the x-axis is the maximal number of generations, which serves as the interruption parameter for the genetic algorithm.

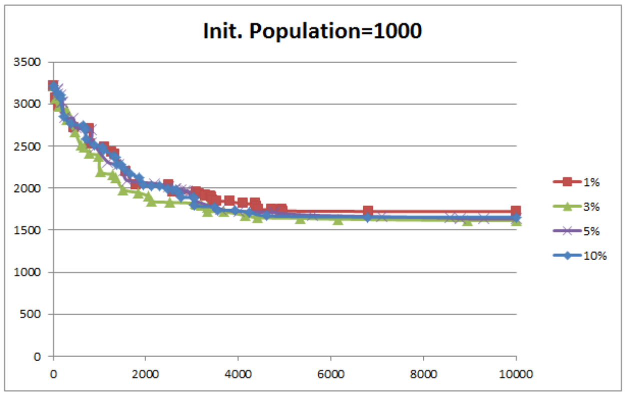


Figure 3 – Initial population size: 1000; pm=1%, 3%, 5% and 10%.

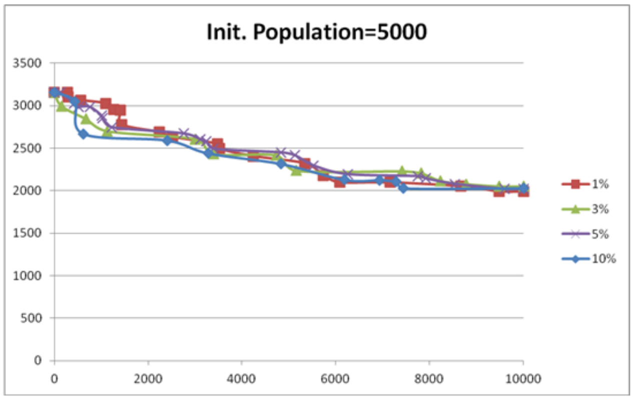


Figure 4 – Initial population size: 5000; pm=1%, 3%, 5% and 10%.

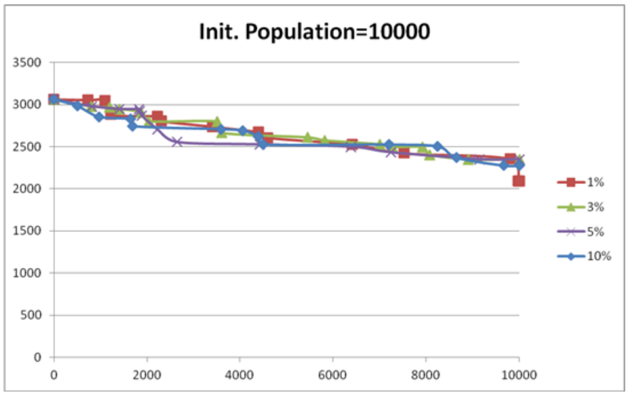


Figure 5 – Initial population size: 10000; pm=1%, 3%, 5% and 10%.

Now we present the results for the cases when, for some particular probability of mutation, we take different values for the parameter of the Initial Population. In Figure 6 we show the changes in fitness for different values of the number of initial population, having the same probability of mutation (pm=1%). Similarly, for other pm values (3%, 5% and 10%), in Figure 7, 8 and 9.

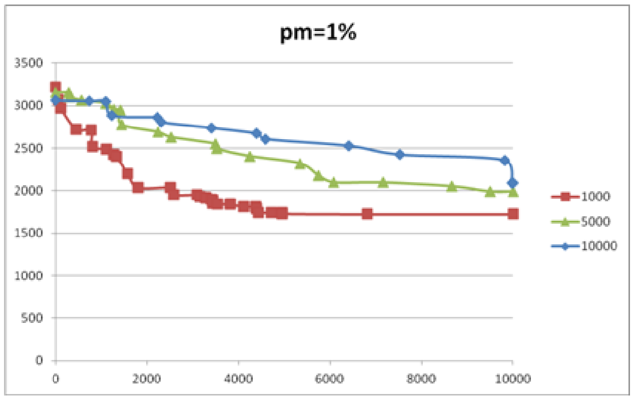


Figure 6 – pm=1%,; Initial population sizes: 1000, 5000 and 10000.

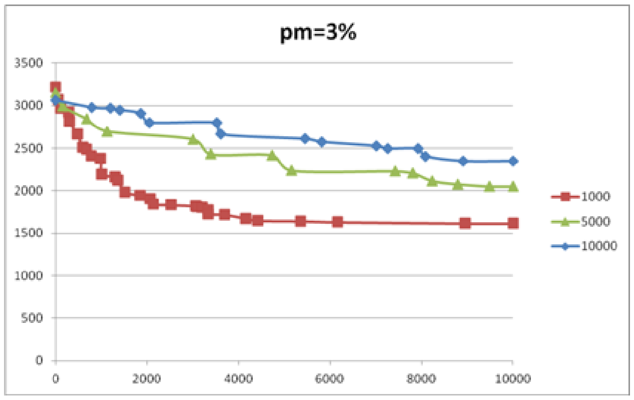


Figure 7 - pm=3%,; Initial population sizes: 1000, 5000 and 10000

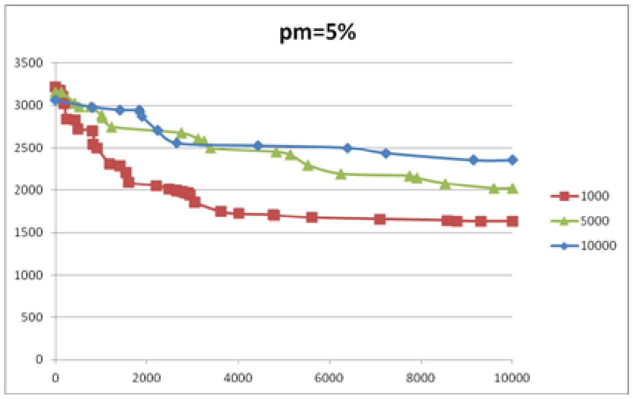


Figure 8 - pm=5%,; Initial population sizes: 1000, 5000 and 10000

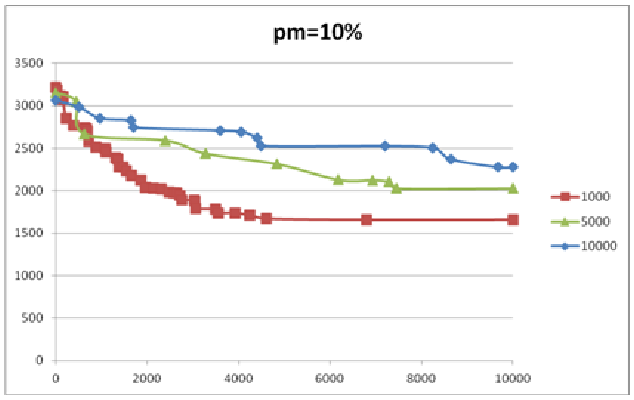


Figure 9 - pm=10%,; Initial population sizes: 1000, 5000 and 10000

4. Conclusions

Since the use of correct population size is a crucial factor for successful GA applications, more efforts need to be spent in finding correct population sizing estimates for particular problems. Our results show that there is a slight or no difference in fitness results for the case of TSP with GA, for different values of the initial population size, with all tested values of mutation parameter. So increasing the mutation rate doesn't contribute to a better results.

There is no use of increasing the value of the probability of mutation, since it doesn't pay-off as it increases the execution time/processor time and doesn't contribute to a much better fitness result.

Considering the effect of changing the initial population size for fixed mutation rate, we see that the effect of the mutation is important for small initial populations, since it contributes to promoting new solutions in the solution space.

For each percentage of the mutation, results show that the bigger the value of the initial population, the less is the fitness value improved.

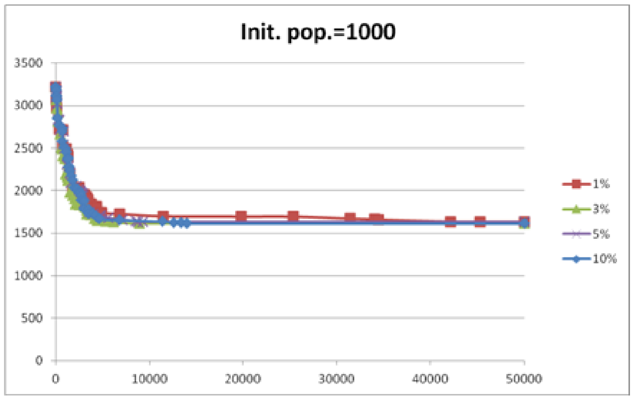
The effect of mutation is especially noticed in the first generations of the cases with small value of the intimal population (when there is no enough number of good solutions). Otherwise, when the initial population is of higher value, it means that there is a higher diversity of solutions and so the mutation doesn't bring much new and better solutions.

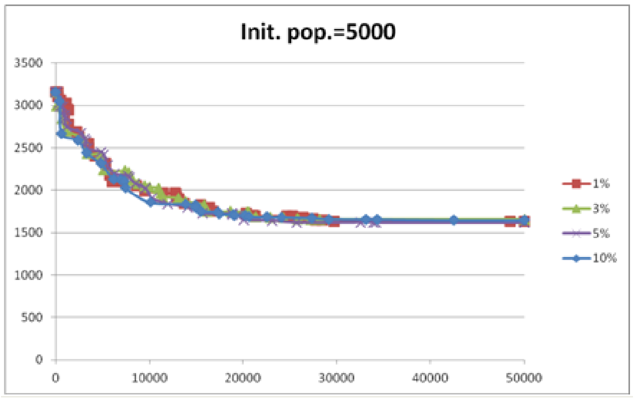
When the population is large, the diversity in the initial random population is large and the best solution in the population is expected to be close to the optimal solution.

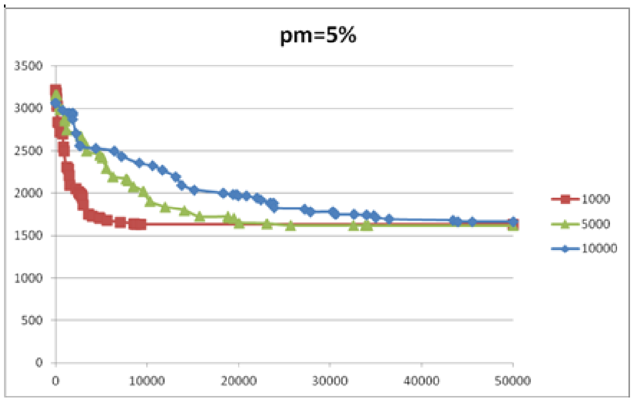
Appendix

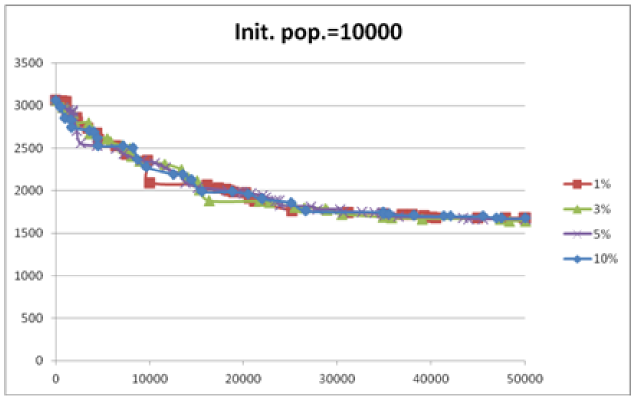
Here we will present the results for the cases when the value of the parameter of maximal number of generations is: 50,000 and 100,000.

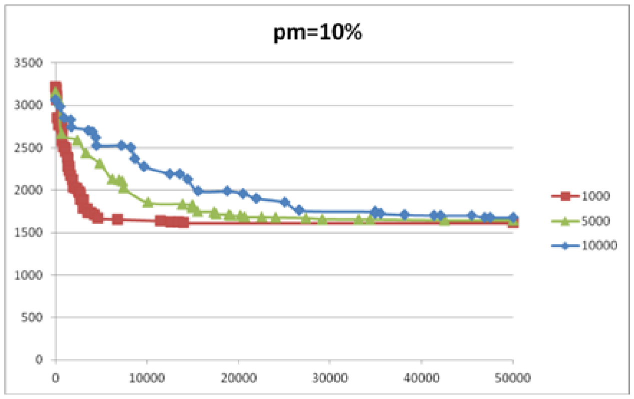
For each value of the number of generations, we present the results for cases with values of the initial population (1000, 5000 and 10000) for different values of the probability of mutation (1%, 3%, 5% and 10%) and then cases with same probabilities of mutation for different values of initial population (1000, 5000 and 10000) .

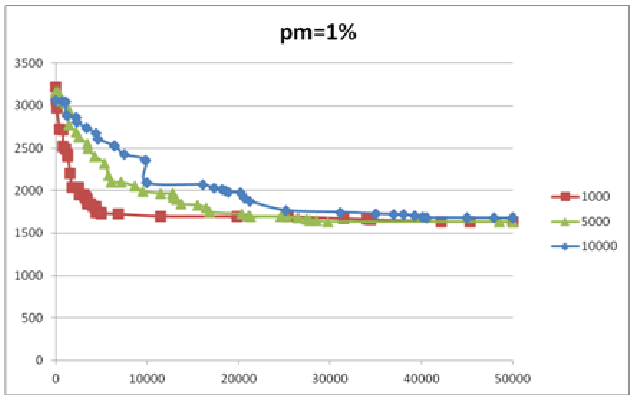


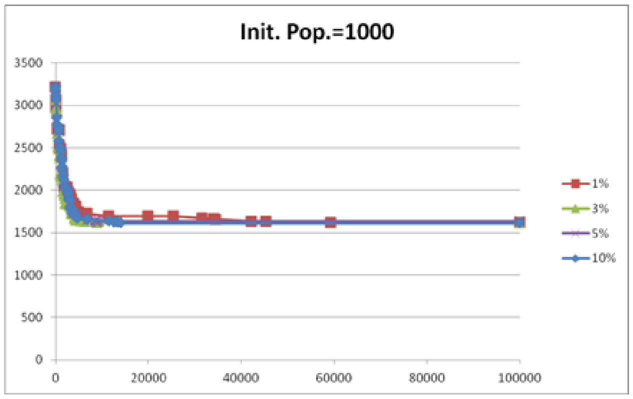

Init. pop.=5000
1%
3%
5%
10%

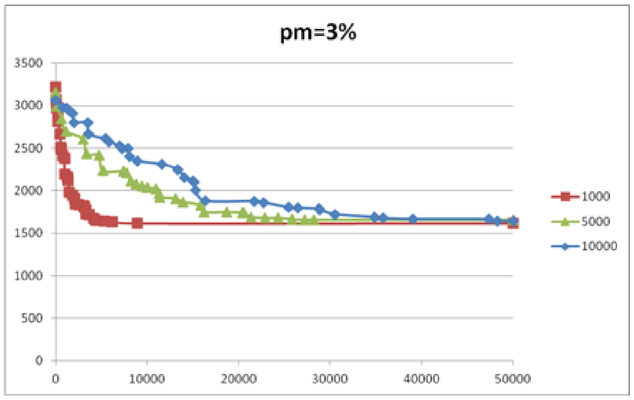

pm=5%
1000
5000
10000

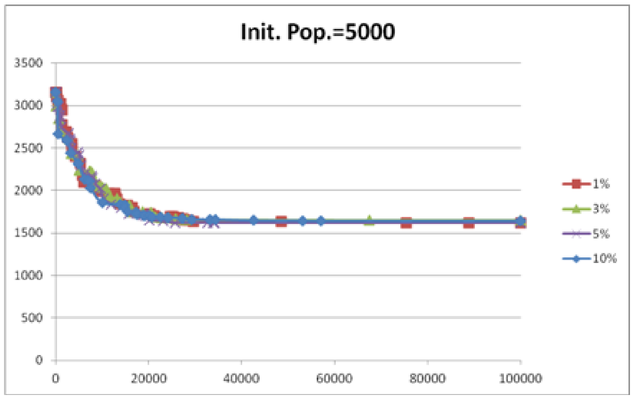

Init. pop.=10000
1%
3%
5%
10%

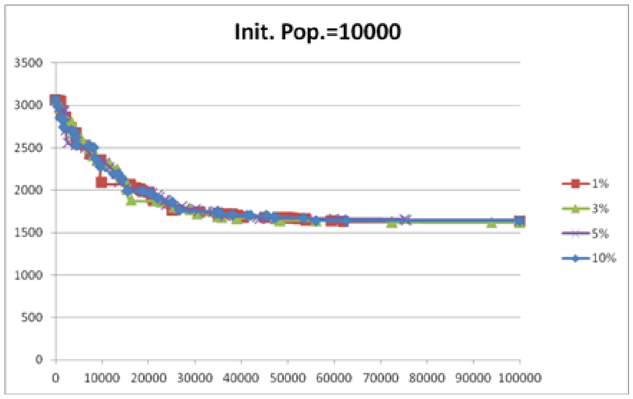

pm=10%
1000
5000
10000

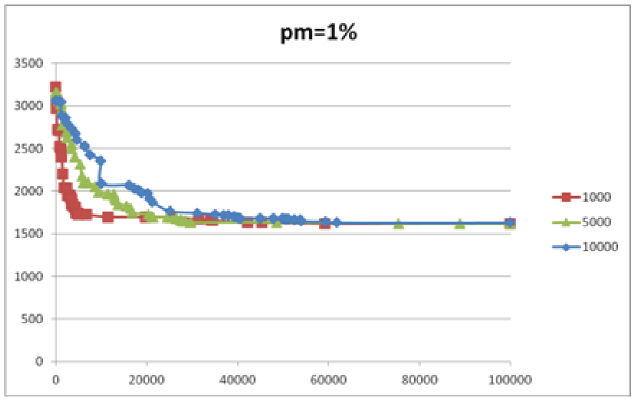

pm=1%
1000
5000
10000

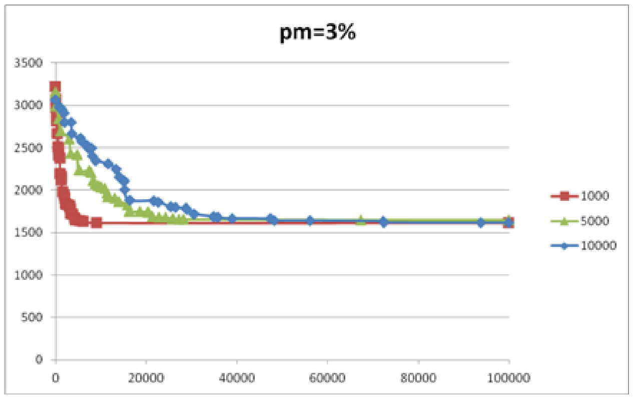

Init. Pop.=1000
1%
3%
5%
10%

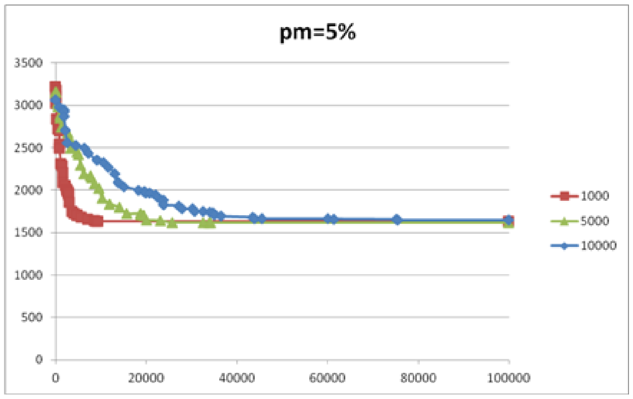

pm=3%
1000
5000
10000


Init. Pop.=5000
1%
3%
5%
10%









References

[1] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[2] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley, 1989.

[3] Michael Junger, Gerhard Reinelt, Giovanni Rinaldi, *The Traveling Salesman Problem*, M.O. Ball et all, Eds. Handbooks in OR & Ms, Vol. 7, Elsevier Science, B.V. 1997

[4] T.-L. Yu, D. E. Goldberg, and Y.-P. Chen, “A genetic algorithm design inspired by organizational theory: A pilot study of a dependency structure matrix driven genetic algorithm,” IlliGAL Report No. 2003007, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 2003.

[5] K. Sastry, D. E. Goldberg, and G. Kendall, “Genetic algorithms: A tutorial,” in Introductory Tutorials in Optimization, Search and Decision Support Methodologies, ch. 4, pp. 97–125, Springer, 2005.

[6] Martin Pelikan, Genetic Algorithms, MEDAL Report No. 2010007, 2010.

[7] R.Sivaraj, T.Ravichandran Computer Engineering and Intelligent Systems, www.iiste.org, ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online) Vol. 3, No.1, 2012.

[8] K. Deb, S. Agrawal, Understanding Interactions among Genetic Algorithm Parameters, KanGAL Report Number 1999003.

[9] E.M. Khalilzad , S.Hosseini, ISSN (Online): Recovery of Faulty Cluster Head Sensors by Using Genetic Algorithm (RFGA), 1694-0814, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 4, No 1, July 2012.

[10] D. Anand, Feature Extraction for Collaborative Filtering: A Genetic Programming Approach, ISSN (Online): 1694-0814, www.IJCSI.org, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 1, September 2012.

[11] Ahmed Azouaoui, Ahlam Berkani and Pr. Mostafa Belkasmi, An Efficient Soft Decoder of Block Codes Based on Compact Genetic Algorithm, ISSN (Online): 1694-0814 www.IJCSI.org. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2, September 2012.

[12] A. Rexhepi, A. Dika, A. Maxhuni, Solving TSP using Genetic Algorithm – Case of Kosova, WSEAS 6th WSEAS European Computing Conference (ECC '12), Prague, Czech Republic, 2012.

Avni Rexhepi, MSc. (DB: 12/12/1969). Master of Science in computer science (2004) - University of Prishtina. Teaching and research Assistant in the University of Prishtina, since 1996.

Adnan Maxhuni, MSc. (DB: 31/12/1967). Master of Science in computer science (2005) - University of Prishtina. Teaching and research Assistant in the University of Prishtina, since 1993.

Agni Dika, Prof. Dr. (DB: 21/02/1950). PhD in computer sciences (1989) - University of Zagreb, Croatia. Ordinary Professor at the Faculty of Electrical and Computer Engineering, in the University of Prishtina.