Short Research Article

A Genetic Algorithm with Neighborhood Search to Solve Integer and Linear Programming Problems

.

ABSTRACT

In this paper, a metaheuristic algorithm that combines genetic and neighbor search algorithms is proposed to solve integer linear programming problems. The individuals of the population are binary coded into a sequence of chromosomes (variables). Initially, chromosome length is five bits (genes) but if required they grow, up to 21 genes per chromosome, when looking for optima. The algorithm includes a test based on systematic neighborhood search to decide if it continues or stops. The algorithm is able to solve maximal or minimal integer linear programming problems in standard or non-standard form and linear programming problems with a simple adaptation. A comparative study was conducted with three algorithms; LINGO, Simplex LP and Evolutionary, these last two algorithms are from commercial solver in Excel spreadsheet software. The results show that the algorithm was able to find similar solution than LINGO and Simplex LP and better than the Evolutionary. A time study using problems from literature with two, three, four, eight and twelve variables is included.

Keywords: Optimization, Metaheuristic, Maximization, Minimization, Genetic Algorithm

1. INTRODUCTION

In this paper, a first effort to deploy a piece of software capable of finding solution to integer linear programming (ILP) and linear programming (LP) is presented and discussed. The test examples are limited to simple examples that range from two up to four variables and up to six constraints. The implementation is based on a variation of the classical Genetic Algorithm (GA) as first discussed in [1]. Research on this topic keeps on going due to the difficulty found when looking for optimal solutions to ILP problems [2, 3, 4].

This paper reports the results when implementing a variation of GA with a set of control parameters to search for optima with normal or aggressive GA mechanism, not only to escape from local optima but also to focus the generation of solutions. Furthermore, we deal with one of the questions when applying the GA: how many generations should the GA must run? The answer could be as much as possible, but it is a vague response. Usually, implementations use a time limit some others terminate after a certain number of generations where no improving has been found or all individual chromosomes are identical [2, 3]. In this implementation a test applying a systematic neighborhood iterated search is conducted [4]. If one better solution is found in the neighborhood of the current solution then the algorithm is triggered again or ends, otherwise.

The implementation includes six control parameters: the model definition: maximization or minimization, number of generations from 100 to 10000, number of ages from 1 to 40, crossover level, mutation level, neighbor range and diversity level. The algorithm is able to

run in a progressive way i.e. if at the end of a run it runs again it will use the results from last run to continue searching for optima and it is where the history tracking control fits, if a completely new run is required.

For each age selected, the algorithm runs the number of generations specified. The algorithm keeps track of fittest individual found on each age and the fittest individual of all ages will be the solution, the best solution, an approximation to the optimal one if not the optimal, along with a list of the fittest individuals of all ages.

GAs are an active research topic that has been implemented to solve optimization problems from non-linear programming, task scheduling, computer vision and multi-objective resource allocation problems [3, 5, 6, 7], and of course, integer linear programming [8, 9], among others. The mixing of GA with neighborhood search is not new, it has been used to solve problems related to resource scheduling, machine cell formation and traveling salesman problem, among others [10,11,12],

Section 2 is about the variation of the GA mechanisms and how they were implemented. Section 3 describes de implementation software. Section 4 presents a comparative study with two other algorithms from commercial software, Microsoft Excel Solver. Section 5 includes a time study with problems ranging from two to twelve variables, maximization and minimization in standard and non-standard from literature. Finally, on Section 6 there are conclusions and recommendations for further research.

2. THE GENETIC ALGORITHM MECHANISMS

In this section, the mechanisms as implemented are discussed. For further explanation of the classical mechanisms of the GA they can be found in [1].

2.1 Initial population

On generation zero, population is generated randomly but from age one and above the initial population for that, is generated based on a diversity level. A level of 100% means that population is generated at random. If this level is set to 50%, for example, there is a 50% chance that individuals will be generated at random and the other 50% will be a neighbor from current best solution, at random but inside a maximum neighbor range.

On generation zero, one individual goes through chromosome repair if it is not feasible [13]. The algorithm uses Equation (1) to compensate a not feasible individual and make it feasible. It is the same for minimization or maximization but on maximization, it decreases the variables values, x_i, if a less than constraint is not met. On the other hand, on minimization, it increases the variables values only when a greater than constraint is not met. The algorithm computes the sum of coefficients c_{ii} at the start of a run. It has been observed that in this implementation the use of equation (1) helps to accelerate the convergence of the algorithm.

Initial population is size 50 and it remains the same through all the optima searching process. Every generation ten individuals are generated applying crossover and/or mutation and they replace ten individuals of the population, the ten individuals with lower fitness evaluation.

Once the initial population is ready the reproduction process is called (crossover and mutation). By default there is a 50-50% chance that a new individual is created using the

crossover or mutation mechanisms. But it can be adjusted to a desirable value with the controls provided.

$$x_j = x_j + \frac{C_i - \sum_{j=1}^{n} c_{ij} x_j}{\sum_{j=1}^{m} c_{ij}} \quad \forall i = 1,2,3,...,n \tag{1}$$

$where:$

$n =$ number of constraints

$m =$ number of variables

$c_{ij} =$ coefficient of constraint i variable j

$C_i =$ constraint i critical value

$x_j =$ variable j, $\quad j = 1,2,3,...,\mathrm{m}$

2.2 Crossover

The algorithm selects two individuals from the population at random and at least one chromosome goes through crossover. A random position in the chromosome is selected and the new individual inherits all the genes from the random position to the right or left of one parent followed by all the genes to the left or right of the random position, from the other parent. There is a 50% chance that the procedure includes all the chromosomes and 50% that it uses a subset of chromosomes. There is a 50% chance of switching who is parent one and who is parent two. Fig. 1 illustrates the crossover process.

Parents are selected randomly from population where the fittest individual has 1/10 chance to be selected for crossover or mutation, while individuals ranked from position 2 to 6 have 1/30 chance and remain individuals have 1/60 chance.

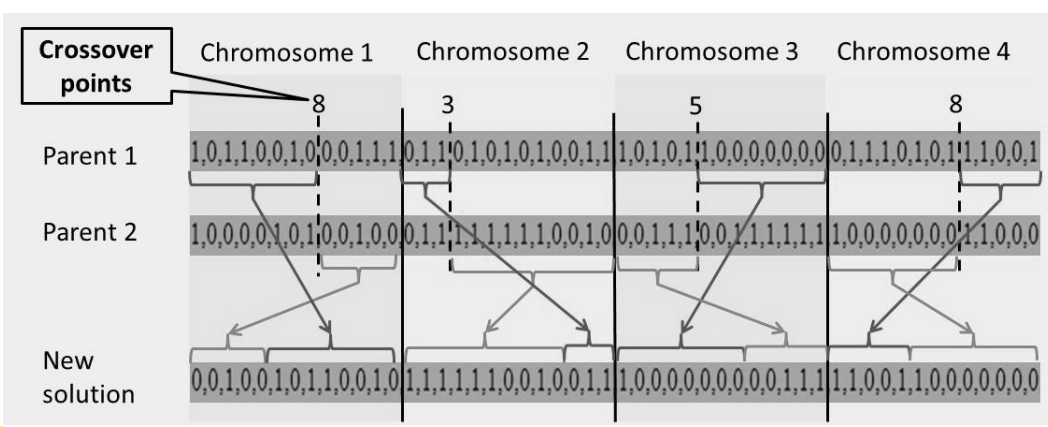
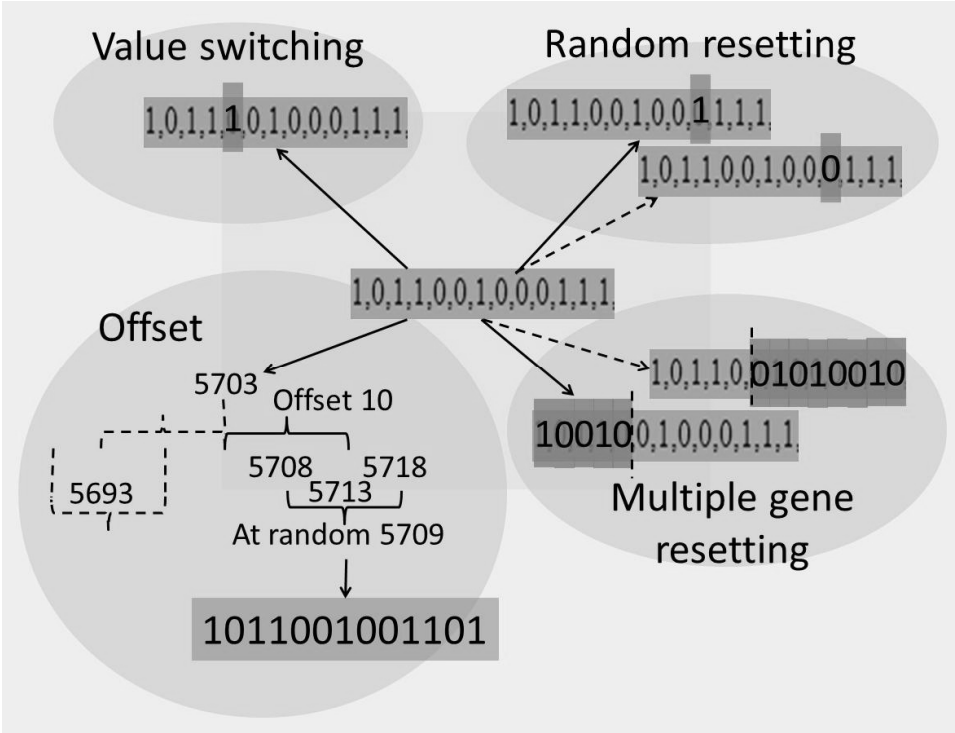


Fig. 1. Example of crossover mechanism with 4 chromosomes and 13 genes.

2.3 Mutation

There are four mutation strategies with equal chance to modify a chromosome: one gene mutation with value switching, one gene mutation with random value setting, multiple gene mutation and offset mutation. In offset mutation, the algorithm mutates the chromosomes from current best solution to another value but inside of range limit from current chromosome value. In this mutation option, the algorithm generates an integer value randomly from zero up to a range limit, an offset, and there is 50-50% chance to go up or down from current chromosome value. The algorithm converts the resulting value of the chromosome to its binary equivalent. Fig. 2 illustrates all four strategies for mutation; the dash lines indicate alternatives due to 50% chance. The illustration depicts a chromosome of 13 genes length at the center of the illustration, from where new chromosomes could be generated using one of



the four possible strategies. The algorithm assigns same probability to all mutation strategies.

Fig. 2. Mutation strategies sample.

2.4 Evaluation

Individuals are evaluated using the objective function and each time a constraint is not met a penalty is considered. The size of the penalty is the sum of the quantities by it goes beyond the constraints limits times the number of restrictions that was not able to meet and times a sensibility factor. The algorithm deducts a penalty from the objective function value when looking for maxima or adds to it when looking for minimal.

The highest value from the objective function coefficients and the constraints critical values is set as the sensibility but if it is lower than 100 then sensibility is set to 100 and 10 000 if it is greater than 10 000.

3. THE IMPLEMENTATION

The implementation runs on any web browser with HTML5 and JavaScript support. All the tests were run under Linux Ubuntu platform using Firefox web browser. At present time, the implementation supports up to twelve variables and eleven constraints. All the runs were done using a laptop 3.9 GB, Intel® Core™ 2 Duo CPU T7250 @ 2.00GHz × 2, Ge Force 9200M, 32-bit OS. Fig. 3 shows a screen shot of the implementation. The implementation is available to install on Android devices at Google Play under the name MathGO.

After clicking the find solution button, the algorithm creates 50 individuals Each individual is set to have equal number of chromosomes as the number of variables in the objective function. All chromosomes are set to five genes length but they will grow if required, small chromosome length reduces the searching space [6]. The algorithm assigns at random a zero or a one to every gene. All individuals go to evaluation process and, if model selector is set to maximize, the algorithm sorts the population in descending order whereas, if selector is set to minimize, the algorithm sorts the population in ascending order.

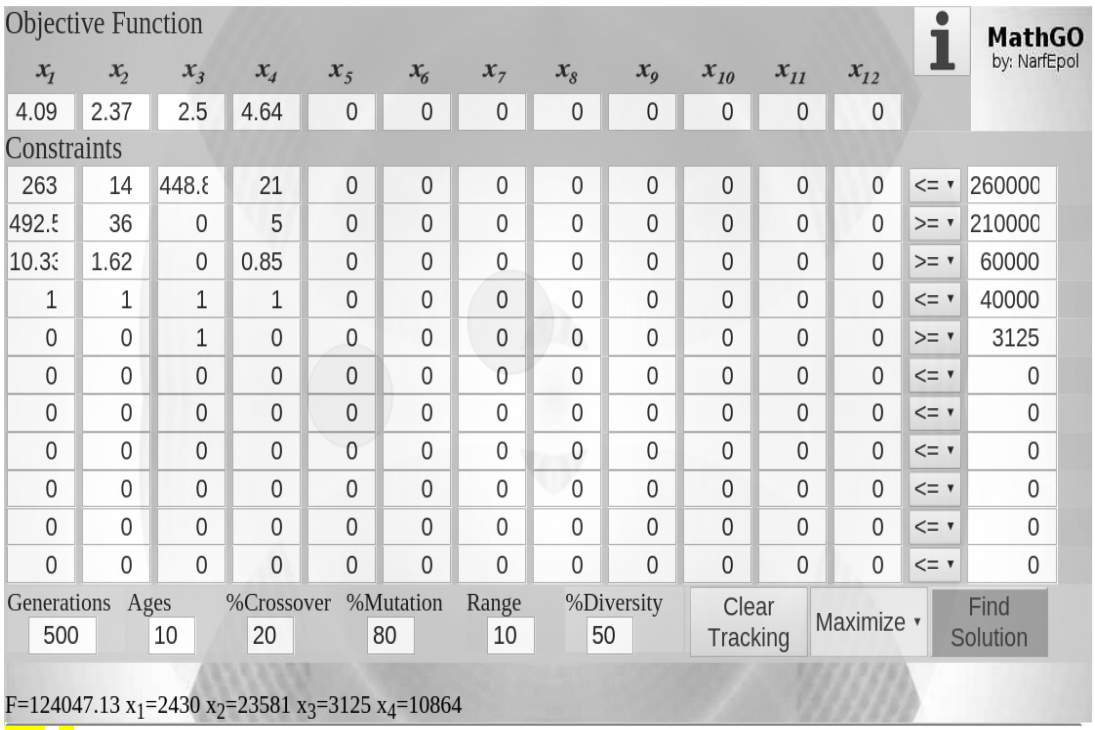


Fig. 3. Implementation display including a partial results list after a run.

The algorithm generates ten new individuals on every generation applying the crossover and mutation mechanisms. New individuals replace the last 10 individuals of the population and sorting reorders the population. The best solution is at the top of the list.

The implementation displays a coefficient matrix where there are input fields and control buttons. There is one button to select the model: maximize or minimize. When selecting maximize or minimize model the operator of the constraints will change automatically from less than to greater than, or vice versa. However, the inequality operators can be change independently as required. The algorithm is able to solve non-standard problems without the

need to convert them into standard form. Below the constraints matrix there are five fields to define the number of generation, ages, crossover, mutation and neighbor range.

Once all the coefficients, critical constraint values, and algorithm parameters are set, clicking on the find solution button will trigger the GA will start the search for optima. At the end of every age, the algorithm displays a list of the population, the fitness value for each individual, variables and their values up to that age. It is possible to change the parameters values while the algorithm is running or stop the running, adjust parameters or change coefficients and continue the run.

At present moment, the algorithm solves ILP problems but it can be used to solve LP problems without the need to change the implementation. If variables are required to get one, two or more decimal point precision just moving the decimal point of the constraint limits, adding zeros to the right or move the decimal point one place for each digit after the decimal point required. The variables values will still be integers but just move the decimal point to the left the same number of places as in the critical constraints values to get a final solution, with decimal point, the same for the objective function value. All the problems in the set used to test the algorithm where a continuous solution was required were solved using this adaptation.

4. RESULTS AND DISCUSSION

4.1 Comparative study

A comparative analysis with commercial software was conducted. Eleven four variables problems were used in this comparative study, three of the problems (9, 10 and 11) are originally LP but were solved adapting them to be solved using ILP. All the problems are from [14]. LINGO and the standard solver in Windows Excel were used to compare the solutions of the proposed algorithm (ILPGA). The two algorithms in the Excel Solver are Simplex LP and Evolutionary [15]. There is another algorithm in the Excel solver, the GRG nonlinear but it was not used on this comparative study. Table 1 lists the solutions obtained from the four algorithms. Column A lists the algorithms: Lingo that uses branch and bound, SLP is the Simplex LP and E is the Evolutionary, for short, both from Microsoft Excel Solver, GA1 is the proposed implementation based on a variation of the genetic algorithm and neighbor search. Column Z is the objective function value and the other columns are the variables values outcome after running the algorithm.

ILPGA is able to find equal solution than LINGO and SLP but it takes longer. There are problems were alternative variables values result in same solution and the ILPG is able to list those alternatives. Table 1 lists only two alternatives for ILPGA on problem 1 and 4. The list of alternatives helps to deduct that any combination where x_1 plus x_3 is equal to 50 is a solution, in case of problem 1.

Only in problem 9, the ILPGA and SLP outcome was different, but probably SLP requires a different setting than the one used in this comparative study. Both solutions were different than the one given in [14], it seems that ILPGA is a better solution because it is lower than the one found using SLP and it does not violates any constraints as the solution given in [14]. The SLP algorithm is based on the branch and bound algorithm, which is one of the fastest algorithms known up today [15].

Lingo also uses branch and bound and the outcome is the optimal solution. Branch and bound easily outperforms any GA if it is compared just based on time and especially in this set of simple problems being used here. However, as mentioned before, there are other

characteristics like the one to find different solutions where they exist, flexibility, and easy to code, among others, that makes GAs appropriate but for sure requires more time and runs than Lingo or SLP.

In the comparative, the SLP did not take more than 7 seconds to find a solution in the worse scenarios of Problems 9, 10 and 11. However, in Problem 9 the SLP was not able to come out with a better solution even after running it multiple times. The ILPGA was able and it did take, on average, 55, 36 and 97 seconds to find the solution of problems 9, 10 and 11, respectively.

Table 1. Comparative study outcome.

Model	A	Z	x_1	x_2	x_3	x_4
Min. $Z = x_1 + 2x_2 + x_3 + 5x_4$ s.t. $x_1 + x_2 + x_3 + x_4 \geq 50$ $3x_1 + x_2 + 2x_3 + x_4 \geq 100$	Lingo	50	50	0	0	0
	SLP	50	50	0	0	0
	E	63	13	0	50	0
	GA1	50	50	0	0	0
	GA1	50	2	0	48	0
Min. $Z = 0.25x_1 + 0.23x_2 + 0.22x_3 + 0.21x_4$ s.t. $x_1 + x_2 \geq 350$ $x_3 + x_4 \geq 300$ $x_1 + x_3 \leq 370$ $x_2 + x_4 \leq 290$	Lingo	147.7	60	290	300	0
	SLP	147.7	60	290	300	0
	E	147.7	60	290	300	0
	GA1	147.7	60	290	300	0
Max. $Z = x_1 + 2x_2 + x_3 + 5x_4$ s.t. $x_1 + 2x_2 + x_3 + x_4 \leq 50$ $3x_1 + x_2 + 2x_3 + x_4 \leq 100$	Lingo	250	0	0	0	50
	SLP	250	0	0	0	50
	E	230	5	0	0	45
	GA1	250	0	0	0	50
Max. $Z = x_1 + x_2 + 4x_3 + 5x_4$ s.t. $x_1 + 2x_2 + 3x_3 + x_4 \leq 115$ $2x_1 + x_2 + 8x_3 + 5x_4 \leq 200$ $x_1 + x_3 \leq 50$	Lingo	200	0	0	0	40
	SLP	200	0	0	0	40
	E	196	4	37	0	31
	GA1	200	0	40	0	32
	GA1	200	0	0	0	40
Min. $Z = 1.2x_1 + 2.1x_2 + 1.8x_3 + 1.5x_4$ s.t. $x_1 + x_2 = 500$; $x_3 + x_4 = 1000$; $x_1 + x_3 \leq 900$; $x_2 + x_4 \leq 700$;	Lingo	2190	5000	0	300	700
	SLP	2190	500	0	300	700
	E	2790	0	500	800	200
	GA1	2190	500	0	300	700
Min. $Z = 14x_1 + 22x_2 + 12x_3 + 10x_4$ s.t. $x_1 + x_2 \leq 25$; $x_3 + x_4 \leq 30$ $x_1 + x_3 = 32$; $x_2 + x_4 = 20$;	Lingo	628	22	0	10	20
	SLP	628	22	0	10	20
	E	828	2	20	30	0
	GA1	628	22	0	10	20
Min. $Z = 30x_1 + 20x_2 + 25x_3 + 22x_4$ $x_1 + x_2 \geq 3\,000$; $x_3 + x_4 \leq 5\,000$	Lingo	180400	200	2800	2800	2200
	SLP	180400	200	2800	2800	2200

Model	A	Z	x_1	x_2	x_3	x_4
	E	180400	200	2800	2800	2200
	GA1	180400	200	2800	2800	2200
Min. $Z = 30x_1 + 20x_2 + 25x_3 + 22x_4$ $x_1 + x_2 \geq 3\,000$; $x_3 + x_4 \geq 5\,000$ $x_1 + x_3 = 5\,000$; $x_2 + x_4 = 5\,000$; $2x_1 + 6x_2 + 5x_3 + 4x_4 \leq 40\,000$	Lingo	244400	2200	800	2800	4200
	SLP	244400	2200	800	2800	4200
	E	250000	3000	0	2000	5000
	GA1	244400	2200	800	2800	4200
Min. $Z = 152x_1 + 61.44x_2 + 884x_3 + 23x_4$ $10.33x_1 + 3.29x_2 + 2.9\,x_4 \geq 400$ $11.44x_1 + 3.34x_2 + 100\,x_3 + 0.26x_4 \geq 2000$ $1.04x_1 + 4.66x_2 + 3.75x_4 \geq 1500$; $x_2 \geq 30.5$	Lingo	26181.08	0	32	18	361
	SLP	26196.52	0	33	18	359
	E	28623.68	0	322	10	0
	GA1	26181.08	0	32	18	361
Max. $Z = 4.09x_1 + 2.37x_2 + 2.5x_3 + 4.64x_4$ $263x_1 + 14x_2 + 448.8x_3 + 21x_4 \leq 2600000$ $492.5x_1 + 36x_2 + 5x_4 \geq 2100000$ $10.33x_1 + 1.62x_2 + 0.85x_4 \geq 60000$ $x_1 + x_2 + x_3 + x_4 \leq 40000$; $x_3 \geq 3125$	Lingo	124047	2430	23581	3125	10864
	SLP	124047	2430	23581	3125	10864
	E	123091	2437	23886	3127	10495
	GA1	124047	2430	23581	3125	10864
Min. $Z = 215x_1 + 884x_2 + 38x_3 + 60x_4$ $26x_1 + 1.16x_3 + 10.51x_4 \geq 500000$ $x_2 + 8.63x_3 + 5.57x_4 \leq 100000$ $6.4x_3 \geq 35\,000$ and $x_2 \geq 450$	Lingo	4434922	15188	450	5469	9398
	SLP	4434922	15188	450	5469	9398
	E	4687827	18987	450	5469	0
	GA1	4434922	15188	450	5469	9398

4.2 Time study

Fig. 4 illustrates the average time from five runs of 117 problems: from one to 59 were problems with two variables, from 60 to 102 were problems with three variables, from 103 to 113 was the set of problems with four variables, the ones used in the comparative study. Problems range from finding maxima and minima with standard and non-standard constraints. All these problems came from [14], some are from examples and others are from the exercises section. Last four problems were the alternatives of the CAPLOC problem in [16]. This problem was divided into four alternatives: three alternatives with eight variables and ten constraints and one alternative with twelve variables and eleven constraints.

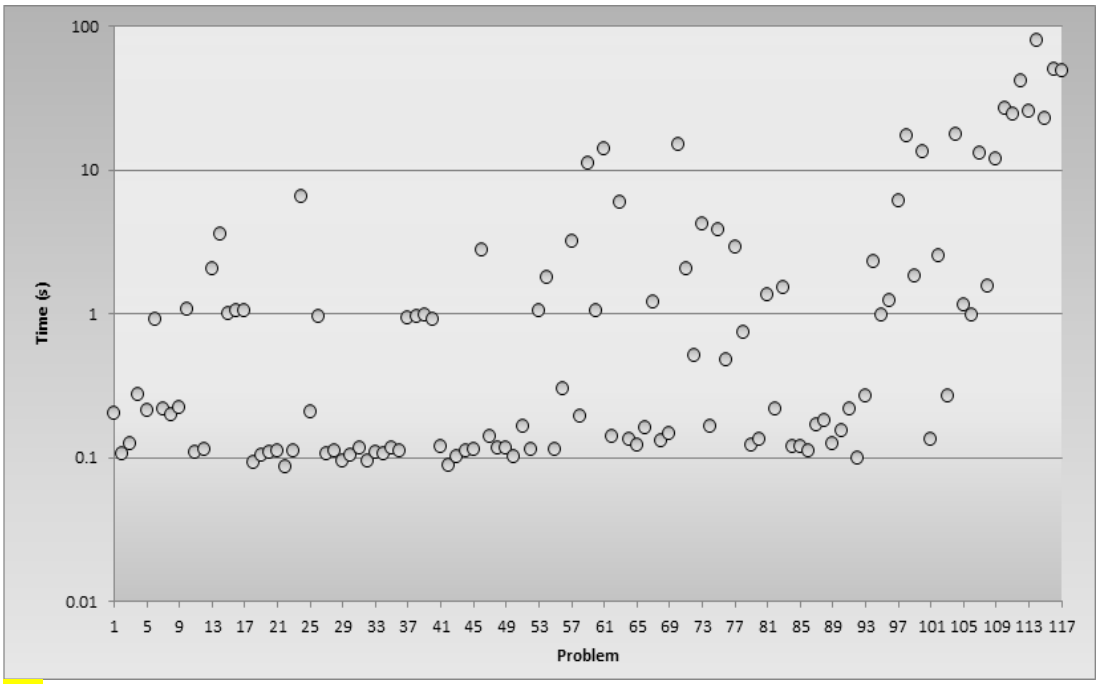


Fig. 4. Time dispersion of average time, from five runs, to get the solution of 117 simple "toy problems" from literature.

There were problems where no maxima exists, in these cases variables keep increasing their value until they reach the maximum limit number of genes and the algorithm terminates the search for optima with a message indicating that not maxima exist and the upper number of genes has been reached. The algorithm was set to run using 500 generations and 10 ages.

5. CONCLUSION

GA1 algorithm performed better than the Evolutionary implementation in Excel Standard Solver and was able to find equal solutions than the Simplex LP. In one problem, GA1 was able to come up with a better solution than the Simplex LP.

More research is required to determine the parameter values to fine tuning the algorithm for a particular set of constraints and the objective function.

More research will be conducted taking into account bigger problems but the implementation until now is able to solve academic problems with up to twelve variables and eleven constraints.

The test to find if the algorithm keeps running requires more research because as the number of variables increase the number of iterations required during testing grows exponentially with it. At twelve variables, testing was set to look in ±1 range and this reduces the power of the test to find a neighbor solution. For now, in cases like this, the alternative is to run one more time or many more times as desirable in hope of finding a better solution.

Another line of research is the improving of equation (1) to better compensate infeasible solutions in the early generations to help accelerate the convergence of the algorithm.

REFERENCES

1. John H. Holland. Genetic Algorithms. Scientific American. July 1992:66-72.

2. Nada Al Sallami, and Sarmad Al Aloussi. A Genetic Algorithm in Green Cloud Computing. British Journal of Applied Science & Technology. 2015;7(2):179-185.

3. Faiz A. El-Qorashy, Hossam A. Nabwey, and A. A. Mousa. A Combined Genetic Algorithms-local Search Engine (GAs–LCE) in Constrained Nonlinear Programming. British Journal of Applied Science & Technology. 2015;8(3):324-333.

4. Guadalupe Castilla Valdez, Shulamith S. Bastiani Medina. Iterated Local search for the Linear Ordering Problem. International Journal of Combinatorial Optimization Problems and Informatics. 2015;3(1):12-20.

5. Jasbir Singh, and Gurvinder Singh. An Optimized Genetic Approach for Scheduling Task Duplication in Parallel Systems. British Journal of Applied Science & Technology. 2015; 10(1):1-12.

6. Ramesh Rajagopalan. A Genetic Algorithm for Optimizing Background Subtraction Parameters in Computer Vision. British Journal of Applied Science & Technology. 2014; 4(29): 4148-4155.

7. Mai A. Farag, M. A. El-Shorbagy, I. M. El-Desoky, A. A. El-Sawy and A. A. Mousa. Genetic Algorithm Based on K-means-Clustering Technique for Multi-objective Resource Allocation Problems. British Journal of Applied Science & Technology. 2015;8(1):80-96.

8. Zhongsheng Hua and Feihua Huang. A variable-grouping based genetic algorithm for large-scale integer programming. Information Sciences. 2006;176:2869-2885.

9. Huy, P.N.A., San, C.T.B., Triantaphyllou, E.: Solving integer programming problems using genetic algorithms. In: ICEIC: International Conference on Electronics, Informations and Commumications. 2004; 400-404.

10. Sepehr Proon, Mingzhou Jin. A Genetic Algorithm with Neighborhood Search for the Resource-ConstrainedProject Scheduling Problem. Naval Research Logistics. 2011;58(2):73-82.

11. Bouazza Elbenani, Jacques A. Ferland and Jonathan Bellemare. Genetic Algorithm and Large Neighbourhood Search to Solve the Cell Formation Problem. CIRRELET. 2010;(39):1-17.

12. Keivan Borna and Vahid Haji Hashemi. An Improved Genetic Algorithm with a Local Optimization Strategy and an Extra Mutation Level for Solving Traveling Salesman Problem. Int. J. of Comp. Sci, Eng. And Inf. Technology. 2014;4(4):47-53.

13. Steven Orla Kimbrough,Gary J. Koehler, Ming Lu, David Harlan Wood. On a Feasible–Infeasible Two-Population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. European Journal of Operational Research. 2008;190:310-327.

14. Lial, M. L. and Hungerford, T. W. Linear Programming. In: Mathematics with Applications. 8th ed. Pearson Education. 2003.

15. Daniel Fylstra, Leon Lasdon, John Watson and Allan Waren. Design and Use of the Microsoft Excel Solver. INTERFACES. 1998;28(5): 29-55.

16. LINDO Systems Inc. Lingo user's manual. Accessed 24 September 2015. Available: http://www.lindo.com/downloads/PDF/LINGO.pdf