

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 is aimed with 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 is conducted with two algorithms, Simplex LP and Evolutionary, from commercial solver software. The results show that the algorithm outcome is better than the Evolutionary algorithm or similar. In one problem it was able to find better optimal solution than using Simplex LP and similar outcome in remain problems. 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

Different approaches have been proposed to solve integer linear programming (ILP) problems using Genetic Algorithms (GAs) but research on this topic keeps on going due to the difficulty found at the time to try to find optimal solutions to ILP problems [1, 2, 3].

This paper is to report our results when implementing a 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 [1, 2]. In this implementation a test applying systematic neighborhood iterated search is conducted [3]. 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 10 000, number of ages from 1 to 40, crossover level, mutation level and neighbor range. The algorithm is able to run in a progressive way i.e. if at the end of a run it is run 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.

35 For each age selected the algorithm runs the number of generations specified. The algorithm
36 keeps track of every best solution found on each age and also the best of the best that
37 represents the optimal solution, along with a list of all individuals of the population.

38 GAs are an active research topic that has been implemented to solve optimization problems
39 from non-linear programming, task scheduling, computer vision and multi-objective resource
40 allocation problems [2, 4, 5, 6], and of course, integer linear programming [7, 8], among
41 others.

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

48

49 **2. THE GENETIC ALGORITHM MECHANISMS**

50

51 **2.1 Initial population**

52

53 On generation zero population is generated randomly but from age one and above the initial
54 population for that age is generated based on a diversity level: 100% is completely random
55 whereas if this level is set to 50% then 50% will be completely random and the other 50%
56 will be a neighbor from current best solution, inside a maximum neighbor range.

57 On generation zero, of every age, one individual, at random, goes through chromosome
58 repair if it is not feasible. Equation (1) is used to compensate a not feasible individual and
59 make it feasible. It is the same for minimization or maximization but on maximization it
60 decreases the variables values, x_j , if a less than constraint is not met. On the other hand,
61 on minimization, it increases the variables values only when a greater than constraint is not
62 met. The sum of coefficients c_{ij} can be computed at the start of a run to avoid recalculating
63 them every time compensation is request: one time for every age and all individuals that are
64 created applying crossover go through the compensation evaluation.

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

69 Once the initial population is ready the reproduction process is called (crossover and
70 mutation). By default there is a 50-50% chance that a new individual is created using the
71 crossover or mutation mechanisms. But it can be adjusted to a desirable value with the
72 controls provided.

73

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

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 , $j = 1, 2, 3, \dots, m$

74

75 2.2 Crossover

76

77 Crossover is done at a chromosome level. Two individuals are selected from the population
78 at random. At least one chromosome is selected randomly for crossover. A random position
79 in the chromosome is selected and the new individual inherits all the genes from the random
80 position to the right of parent one chromosome followed by all the genes to the left of the
81 random position, from parent two. the procedure is applied to all the chromosomes. There is
82 a 50% chance of switching who is parent one and who is parent two. All individuals created
83 using this mechanism go chromosome compensation using equation (1).

84 Parents are selected randomly from population where the fittest individual has 1/10 chance
85 to be selected for crossover or mutation, while individuals ranked from position 2 to 6 have
86 1/30 chance and remain individuals have 1/60 chance.

87

88 2.3 Mutation

89

90 Two strategies for mutation have been implemented: multiple bit mutation and offset
91 mutation. In offset mutation the chromosomes from current best solution are mutated to
92 another value but inside of range limit from current chromosome value. In this mutation
93 option an integer value is generated randomly from zero up to the range limit of the current
94 chromosome value, an offset, and there is 50-50% chance to go up or down from current
95 chromosome value. The offset is randomly generated for every chromosome selected for
96 mutation. The resulting value of the chromosome is converted to its binary equivalent.

97 2.4 Evaluation

98

99 Individuals are evaluated using the objective function and each time a constraint is not meet
100 a penalty is considered. The size of the penalty is the sum of the quantities by it goes
101 beyond the constraints limits times the number of restrictions that was not able to meet and
102 times a sensibility factor. Penalty is deducted from the objective function value when looking
103 for maxima or added 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 GENETIC ALGORITHM MECHANISMS

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. Figure 1 shows a screen shot of the implementation.

When the find solution button is clicked the algorithm creates a 50 individuals initial population. 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]. Genes are randomly assigned with a zero or a one. All individuals are evaluated and sorted in descending or ascending order depending if the model selector is set to maximize or minimize, respectively.

Objective Function												
x ₁	x ₂	x ₃	x ₄	x ₅	x ₆	x ₇	x ₈	x ₉	x ₁₀	x ₁₁	x ₁₂	
152	61.44	884	23	0	0	0	0	0	0	0	0	

Constraints												
x ₁	x ₂	x ₃	x ₄	x ₅	x ₆	x ₇	x ₈	x ₉	x ₁₀	x ₁₁	x ₁₂	
10.33	3.29	0	2.9	0	0	0	0	0	0	0	0	>= 400
11.44	3.34	100	0.26	0	0	0	0	0	0	0	0	>= 2000
1.04	4.66	0	3.75	0	0	0	0	0	0	0	0	>= 1500
0	1	0	0	0	0	0	0	0	0	0	0	>= 30.5
0	0	0	0	0	0	0	0	0	0	0	0	>= 0
0	0	0	0	0	0	0	0	0	0	0	0	>= 0
0	0	0	0	0	0	0	0	0	0	0	0	>= 0
0	0	0	0	0	0	0	0	0	0	0	0	>= 0
0	0	0	0	0	0	0	0	0	0	0	0	>= 0
0	0	0	0	0	0	0	0	0	0	0	0	>= 0
0	0	0	0	0	0	0	0	0	0	0	0	>= 0
0	0	0	0	0	0	0	0	0	0	0	0	>= 0

Generations	Ages	%Crossover	%Mutation	Range	Diversity			
500	10	10	90	10	90	Clear Tracking	Minimize	Find Solution

F=26181.08 x₁=0 x₂=32 x₃=18 x₄=361 Penalty=0 Mechanism=5 Run End Time=30.549 Found Time=25.075

F=26181.08 x₁=0 x₂=32 x₃=18 x₄=361 Penalty=0 Mechanism=5

F=26181.08 x₁=0 x₂=32 x₃=18 x₄=361 Penalty=0 Mechanism=5

F=26181.08 x₁=0 x₂=32 x₃=18 x₄=361 Penalty=0 Mechanism=5

F=26196.52 x₁=0 x₂=33 x₃=18 x₄=359 Penalty=0 Mechanism=5

F=26196.52 x₁=0 x₂=33 x₃=18 x₄=359 Penalty=0 Mechanism=5

F=26234.96 x₁=0 x₂=34 x₃=18 x₄=358 Penalty=0 Mechanism=5

F=26242.52 x₁=0 x₂=33 x₃=18 x₄=361 Penalty=0 Mechanism=5

F=26242.52 x₁=0 x₂=33 x₃=18 x₄=361 Penalty=0 Mechanism=5

Fig. 1. Implementation display including a partial results list after a run.

Once the initial population is set then it calls the reproduction function where every generation, applying the crossover and mutation mechanisms, 10 new individuals are generated. 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. But, the inequality operators can be change independently as required, non-standard problems can be solved 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 each age a list is displayed showing the final population, fitness value for each individual including variables and their values up to that age. It is possible to change the parameters while the algorithm is running and save the form data using the right click context menu.

The algorithm has been conceived to solve 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 get solved using ILP. All the problems are from [9]. The standard solver in Windows Excel was used to compare the solutions of the ILPGA against two algorithms in the Excel Solver; Simplex LP and Evolutionary [10]. Table 1 list the solutions obtained from the three algorithms. Column A lists the algorithm: SLP is the Simplex LP and E is the Evolutionary, for short, both from the Excel Solver, GA1 is the proposed implementation. Column Z is the objective function value and the other columns are the variables values.

ILPGA is able to find solution equal to SLP but its ability to find multiple solutions where they exist is a good point to ILPGA, not all possible solutions are listed in the Tables 1 and not all the solutions are obtained in just one run but the list of solution helps to deduct that in Problem 1 any combination where x_1 plus x_3 is equal to 50 is a solution to the problem.

Only in problem 9, the ILPGA and SLP outcome was different, and both solutions were different than the one given in [9], 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 he solution given in [9]. The SLP algorithm is based on the Branch and Bound algorithm which is one of the fastest algorithms known up today [10]. It easily outperforms any GA if it is compared just based on time. However, as mentioned before, there are other characteristics like the one to find different solutions where they exist that makes the ILPGA suitable but for sure requires more time than the 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 did take,

on average, 55, 36 and 97 seconds to find the solution of problems 9, 10 and 11, respectively.

186
187
188
189

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.: $x_1 + x_2 + x_3 + x_4 \geq 50$ $3x_1 + x_2 + 2x_3 + x_4 \geq 100$	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.: $x_1 + x_2 \geq 350$; $x_3 + x_4 \geq 300$ $x_1 + x_3 \leq 370$; $x_2 + x_4 \leq 290$	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.: $x_1 + 2x_2 + x_3 + x_4 \leq 50$ $3x_1 + x_2 + 2x_3 + x_4 \leq 100$	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.: $x_1 + 2x_2 + 3x_3 + x_4 \leq 115$ $2x_1 + x_2 + 8x_3 + 5x_4 \leq 200$ $x_1 + x_3 \leq 50$	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.: $x_1 + x_2 \geq 500$; $x_1 + x_2 \leq 500$ $x_3 + x_4 \geq 1000$; $x_3 + x_4 \leq 1000$ $x_1 + x_3 \leq 900$; $x_2 + x_4 \leq 700$	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.: $x_1 + x_2 \leq 25$; $x_3 + x_4 \leq 30$ $x_1 + x_3 \geq 32$; $x_1 + x_3 \leq 32$ $x_2 + x_4 \geq 20$; $x_2 + x_4 \leq 20$	SLP	628	22	0	10	20
	E	828	2	20	30	0
	GA1	628	22	0	10	20
Min. $Z = 30x_1 + 20x_2 + 25x_3 + 22x_4$ s.t.: $x_1 + x_2 \geq 3\ 000$; $x_3 + x_4 \geq 5000$ $x_1 + x_3 \leq 5\ 000$; $x_2 + x_4 \leq 5000$ $2x_1 + 6x_2 + 5x_3 + 4x_4 \leq 40000$	SLP	180400	200	2800	2800	2200
	E	180400	200	2800	2800	2200
	GA1	180400	200	2800	2800	2200
Min. $Z = 30x_1 + 20x_2 + 25x_3 + 22x_4$ s.t.: $x_1 + x_2 \geq 3\ 000$; $x_3 + x_4 \geq 5000$ $x_1 + x_3 \leq 5000$; $x_1 + x_3 \geq 5000$; $x_2 + x_4 \leq 5000$ $x_2 + x_4 \geq 5000$; $2x_1 + 6x_2 + 5x_3 + 4x_4 \leq 40000$	SLP	244400	2200	800	2800	4200
	E	250000	3000	0	2000	5000
	GA1	244400	2200	800	2800	4200
Min. $Z = 152x_1 + 61.44x_2 + 884x_3 + 23x_4$ s.t.: $10.33x_1 + 3.29x_2 + 2.9x_4 \geq 400$ $11.44x_1 + 3.34x_2 + 100x_3 + 0.26x_4 \geq 2000$ $1.04x_1 + 4.66x_2 + 3.75x_4 \geq 1500$; $x_2 \geq 30.5$	SLP	26196.52	0	33	18	359
	E	28623.68	0	322	10	0
	GA1	26181.08	0	32	18	361
Max. $Z = 4.09x_1 + 2.37x_2 + 2.5x_3 + 4.64x_4$ s.t.: $263x_1 + 14x_2 + 448.8x_3 + 21x_4 \leq 2600000$ $492.5x_1 + 36x_2 + 5x_4 \geq 2\ 100000$ $10.33x_1 + 1.62x_2 + 0.85x_4 \geq 60000$ $x_1 + x_2 + x_3 + x_4 \leq 40000$; $x_3 \geq 3125$	SLP	124047	2430	23581	3125	10864
	E	123091	2437	23886	3127	10495
	GA1	124047	2430	23581	3125	10864
Min. $Z = 215x_1 + 884x_2 + 38x_3 + 60x_4$ s.t.: $26x_1 + 1.16x_3 + 10.51x_4 \geq 500000$	SLP	4434922	15188	450	5469	9398
	E	4687827	18987	450	5469	0

$$x_2 + 8.63x_3 + 5.57x_4 \leq 100000$$

$$6.4x_3 \geq 35\,000 \text{ and } x_2 \geq 450$$

GA1 4434922 15188 450 5469 9398

4.1 Time study

Figure 2 illustrates the average time from five runs of 117 problems: from 1 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 also non-standard constraints. All these problems came from [9], some are from examples and others are from the the exercises section. Last four problems were the alternatives of the CAPLOC problem in [11]. This problem was divided into four alternatives: three alternatives with eight variables and ten constraints and one alternative with twelve variables and eleven constraints.

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.

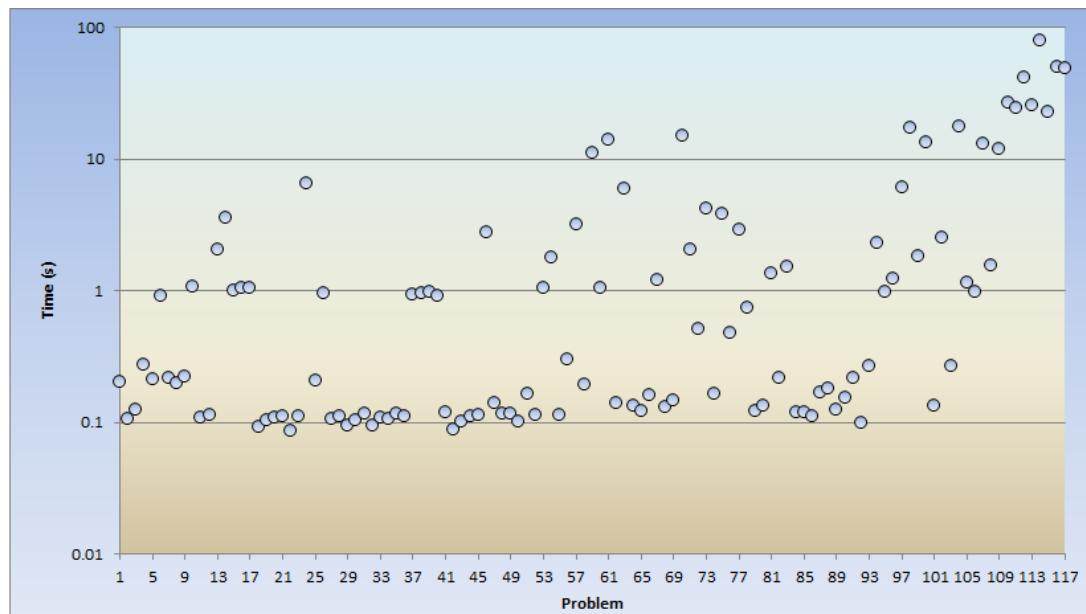


Fig. 2. Time dispersion of average time, from five runs, to get the solution of 117 problems from literature.

4. 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 automatically determine the parameter values to fine tuning the algorithm for a particular set of constraints and the objective function.

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

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

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

229

230

231 REFERENCES

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

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

237 3. Guadalupe Castilla Valdez, Shulamith S. Bastiani Medina. Iterated Local search for the
238 Linear Ordering Problem. International Journal of Combinatorial Optimization Problems and
239 Informatics, Vol. 3, No. 1, 2012, pp. 12-20

240 4. Jasbir Singh, and Gurminder Singh. An Optimized Genetic Approach for Scheduling Task
241 Duplication in Parallel Systems. British Journal of Applied Science & Technology, 10(1): 1-
242 12, 2015.

243 5. Ramesh Rajagopalan. A Genetic Algorithm for Optimizing Background Subtraction
244 Parameters in Computer Vision. British Journal of Applied Science & Technology 4(29):
245 4148-4155, 2014

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

249 7. Zhongsheng Hua and Feihua Huang. A variable-grouping based genetic algorithm for
250 large-scale integer programming. Information Sciences 176 (2006) 2869–2885.

251 8. Huy, P.N.A., San, C.T.B., Triantaphyllou, E.: Solving integer programming problems using
252 genetic algorithms. In: ICEIC: International Conference on Electronics, Informations and
253 Communications, pp. 400–404 (2004)

254 9. Lial, M. L. and Hungerford, T. W. (2003) Linear Programming. In: Mathematics with
255 Applications. 8th ed. Pearson Education. pp. 343-414.

- 256 10. Daniel Fylstra, Leon Lasdon, John Watson and Allan Waren. Design and Use of the
257 Microsoft Excel Solver. INTERFACES, Vol. 28, No. 5, Sept-Oct 1998, pp. 29-55.
- 258 11. LINDO Systems Inc. Lingo user's manual. Accessed 24 September 2015. Available at:
259 <http://www.lindo.com/downloads/PDF/LINGO.pdf>