

# Population Size, Search Space and Quality of Solution: An Experimental Study

**Ruhul Amin SARKER and M. F. Azam KAZI**

School of Information Technology & Electrical Engineering  
University of New South Wales at the Australian Defence Force Academy  
Northcott Drive, Canberra 2600, Australia  
<ruhul@cs.adfa.edu.au>

**Abstract-** Evolutionary Algorithms (EAs) has attracted increasing attention in recent years, as powerful computational technique, for solving many complex real-world problems. The successful application of Evolutionary algorithms to optimization problems is dependent on the methods and parameters used for the algorithm. In this paper, we investigate the effect of population sizes on the quality of solutions to be obtained, the computational time to be required and the size of search spaces of the problems under consideration. We select a two-stage transportation problem as a test case and also use a well-known conventional optimization technique to compare the solutions. The numerical results are analysed and the interesting findings are discussed.

## 1 Introduction

Over the last two decades, Evolutionary Algorithms (EAs) have shown tremendous success in solving complex real-world problems (Fischer and Leung, 2001 and Barnett et al., 2000). Due to its parallelism and some intelligent properties such as self-organization, adaptation and self-learning, EC has been applied successfully to many problems where the classical approaches are either unavailable or generally lead to unsatisfactory results. In recent years, the interest in EC has been growing dramatically (Yao, 1999 and 2002).

The great success of EAs was first recognized in the 1980s, when extremely complex optimization problems from various disciplines were solved, thus facilitating the undeniable breakthrough of EC as a problem-solving methodology (Goldberg, 1989). EAs are basically heuristic search algorithms. The two major steps in applying any heuristic search algorithm to a particular problem are the specification of the representation and the evaluation (i.e, fitness) function. These two items work like a bridge between the original problem context and the problem-solving framework. When defining an evolutionary algorithm, one needs to choose its core components, such as variation operators (mutation and recombination) that suit the representation, selection mechanisms for selecting parents and survivors, and an initial population. Each of these components may have their own parameters, for instance: the probability of

mutation, the tournament size of selection, or the population size. The values of these parameters greatly contribute to: whether the algorithm will find an acceptable solution, and whether it will find such a solution efficiently. Choosing the right parameter values, however, is a time-consuming task and considerable effort has gone into developing good heuristics for it (Eiben et al, 1999).

In this paper, we will concentrate only on the population size. The question as to how to choose an adequate population size for a particular domain is difficult and has puzzled practitioners for a long time (Quintero and Pierre, 2003; Ahn and Ramakrishna 2002; Harik et al, 1999; Goldberg et al, 1992; and Goldberg and Rudnick, 1988). If the population size is too small, it is not likely that the GAs will find solutions of high quality. However, if the population size is too large, the GAs will unnecessarily waste processing time leading to unacceptably slow convergence. Harik et al (1999) exploited the similarity between the gambler's ruin (one-dimensional random walk) problem and the selection mechanism of GAs for determining an adequate population size that guarantees a solution of the desired (/target) quality. In their work, they assumed that mutation is not a dominant operator. The use of many domain-dependent variables and variable-length chromosomes makes their population-sizing equation unsuitable for applying to other problem domains (Ahn and Ramakrishna 2002). However, the general idea is that the population size may be increased to reach a particular quality of solution.

The integer programming problems are known as hard problems in the optimization literature. Sakawa and Kato (2003) proposed a GA for solving a multidimensional 0-1 knapsack problem and compared the performance with a well-known conventional optimization technique such as branch-and-bound (B&B). The details of the test problems, parameters used for GA, optimization package, computer platform and other information can be found in Sakawa and Kato (2003). As they experimented with fixed population size, GAs provided optimal solutions for smaller problems and the quality of solution deteriorated (slowly) as the size of the problem instance increases. However, the computational time, for GA compared to B&B, is more and more appreciable as long as the size of

the problem instance increases. This is a very favourable point for using EAs in solving hard optimization problems. Sarker et al (2001b) developed a GA for solving a constrained integer-nonlinear programming model arising from a joint product-raw material batch sizing problem. In their experimental study, they consider both real and binary coded GA with a number of different crossover, mutation and penalty function methods. They reported that, for any combination of the parameters and methods, the number of optimal solutions per run had been increased, for such a fixed size model, with the increase of the population sizes. From that study, it was also found that the number of optimal solutions per run had been decreased with the tightness of the constraints (Sarker et al., 2001a & 2001b).

In this paper, we will present an experimental study to show the effect of population sizes on the quality of solutions and the search spaces of the problems under consideration. The number of fitness evaluations and the computational time required will also be analysed. To carry out the experiments, we will briefly introduce a two-stage transportation problem and its mathematical model, and then develop a new GA for solving the model. We will also solve the model using a well-known optimization package to compare the quality of solutions. To the best of our knowledge, no such study for a general optimization problem (like a two-stage transportation problem) has been appeared in the literature. Although our theoretical work on this topic is premature, we will leave it as our future research. The experimental results confirm, the general idea, that the population sizes may need to increase for improving the quality of solutions for a two-stage transportation problem, and it may need further increasing for higher dimensional models. The relationship between the population size and the number of fitness evaluation is not linear. It is relevant to mention here that there is a recent theoretical analysis of the impact of population size on EA's computation time which can be found in He and Yao, 2002.

The organization of the paper is as follows: following the introduction, this paper presents a two-stage transportation problem and its mathematical model in section 2. Section 3 develops genetic algorithms for solving the model developed in section 2. Section 4 reports and analyses the computational results. Finally, the conclusions are provided in section 5.

## 2 A Two-Stage Transportation Problem

Consider a two stage transportation problem where  $n$  sources,  $m$  intermediate /exchange points and  $q$  destinations exist (see Figure 1, next page). The sources have limited supply, the exchange points have limited capacity to handle the supplied goods and the destinations have known demand. All the supplies from the sources are transported to the destinations via the exchange points. The problem is to minimize the overall transportation cost

while satisfying supply, capacity, demand and flow constraints. This problem can easily be formulated as a mathematical programming model as follows.

### 2.1 Notations

We use the following notations in formulating the mathematical model:

- $x1_{ij}$  = The quantity transported from source  $i$  to exchange point  $j$
- $x2_{jk}$  = The quantity transported from exchange point  $j$  to destination  $k$
- $C1_{ij}$  = Unit transportation cost from source  $i$  to exchange point  $j$
- $C2_{jk}$  = Unit transportation cost from exchange point  $j$  to destination  $k$
- $S_i$  = Total supply from source  $i$
- $CP_j$  = Capacity of exchange point  $j$
- $D_k$  = Demand of destination  $k$

### 2.2 Mathematical Model

The mathematical model of the problem can be formulated as follows:

$$\text{Minimize } Z = \sum_i \sum_j C1_{ij} x1_{ij} + \sum_j \sum_k C2_{jk} x2_{jk}$$

Subject to

$$\begin{aligned} \sum_j x1_{ij} &\leq S_i, & \forall i \\ \sum_i x1_{ij} &\leq CP_j, & \forall j \\ \sum_j x2_{jk} &\geq D_k, & \forall k \\ \sum_i x1_{ij} - \sum_k x2_{jk} &\geq 0, & \forall j \\ x1_{ij}, x2_{jk} &\geq 0 \text{ and integer,} & \forall i, j, k \end{aligned}$$

The objective of the problem is to minimize the overall transportation costs. The first, second and third constraints ensure the supply limitation, exchange point capacity and demand requirements respectively. The fourth or the flow constraint indicates that the total supply (in the second stage) from an exchange node  $j$  to all the destinations must be less than or equal to the supply (in the first stage) received by that  $j$  from all sources. This is a standard integer programming model as the variables  $x1_{ij}$  and  $x2_{jk}$  are integer in nature. We assume there exist a number of feasible solutions. For any feasible solution, the first and the last two constraints are binding which makes it a difficult model to be solved by using EAs.

In this paper, we have chosen GAs to solve the above model as they are more popular among the practitioners. The two-stage transportation problem was selected mainly for two reasons: (i) this problem generates many tighter constraints which is not an easy test problem for GA

application and (ii) the corresponding mathematical model can easily be solved using a standard optimization package to compare the solutions.

### 3. Development of GA

First we consider a simple real-coded GA, where the variable values were generated as real/ integer numbers not as binary-string. Usually the binary coding is recognized as the most suitable encoding for any problem solution because it maximizes the number of schemata being searched implicitly (Holland, 1975 and Goldberg, 1989), but there have been many examples in the evolutionary computation literature where alternative representations have resulted in algorithms with greater efficiency and optimization effectiveness when applied to identical problems (see e.g. articles by Back and Schwefel, 1993 and Fogel and Stayton, 1994, among others). Davis (1991) and Michalewicz (1996) comment that for many applications real-values or other representations may be chosen to advantage over binary coding. There does not appear to be any general benefit in maximizing implicit parallelism in evolutionary algorithms, and, therefore, forcing problems to fit into a binary representation may not be recommended. As experimented by Sarker et al (2001a & b), real coded GAs perform better than Binary coded GAs for integer programming problems.

The following characteristics were considered arbitrarily (as of Sarker et al, 2001a & 2001b) in running the Simple GA (SGA):

#### Algorithm 1: Simple GA

- Population size 50
- Initial population was generated randomly within the given range of supply, capacity and requirements.
- Heuristic crossover (as suggested by Sarker et al., 2001a & b) and non-uniform mutations
- Probability of crossover = 1.0, and Probability of mutation = up to 0.10
- Penalty methods for constraints handling: dynamic
- Selection for crossover: First rank all the individuals based on the objective function values. Then select randomly one from the top 10 and two from the rest. Crossover was made between the first one and the better of other two.

The outputs of the algorithm were not satisfactory even for small problems with 3 sources, 2 exchange points and 3 destinations. We run the algorithm 30 times, each time for 300 generations. Most of the time, the algorithm produces too many infeasible solutions (sometimes all) and ended up either with infeasible or suboptimal (very poor feasible) solutions. To overcome this problem, we modified the algorithm to force the feasibility of offspring in each and every generation. We introduce the modified algorithm as Modified GA (MGA).

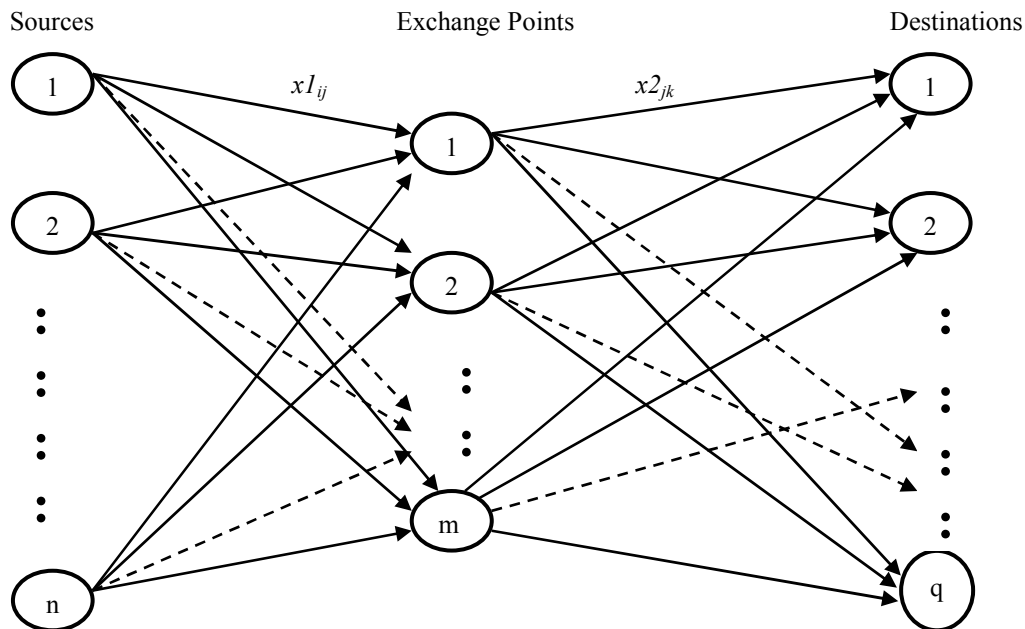


Figure 1: A Two-stage Transportation problem

### *Algorithm 2: Modified GA*

- Population size 50
- Initial population was generated randomly ensuring their feasibility (discarding infeasible solutions).
- Heuristic crossover (as Algorithm 1).
- Probability of crossover = 1.0.
- Constraints handling: ensure feasibility with repairing mechanism.
- Selection for crossover: First rank all the individuals based on the objective function values. Then select randomly one from the top 10 and two from the rest. Crossover was made between the first one and the better of the other two.
- Crossover: Once the parents are selected for crossover, choose a variable randomly (for example, a path or supply amount from a source to an exchange point) and perform crossover. If the variable is within the current range (supply and capacity), we accept and reset the available supply and capacity for other variables. Otherwise, we set them at their extreme values. Once the supply is exhausted from a source, all unassigned paths from that source will be assigned to zero shipments without performing crossover operation. If any positive amount is left in any source after assigning all the paths (originating from that source) except one, then the entire amount will be allocated to that path without performing any crossover if it is less than the remaining exchange point capacity. Otherwise the minimum of available supply and exchange point capacity will be allocated.
- No mutation is used as the usual mutations would destroy the feasible structure of the current problem and we are not interested to design a specialised one at this stage.
- Ensuring Feasibility: After performing crossover following the above procedure, the problem could be infeasible in some cases because of the allocation without crossover as mentioned above and zero allocation after crossover (if the allocation is less than or equal to zero after crossover). These sorts of allocations may violate the supply constraints. Similar violation may occur in the destination side. In such a case, we apply a repair mechanism to ensure the feasibility in the final allocation.
- Repair Mechanism: After crossover, identify all the infeasible individuals. Identify the sources that have supplied less than their available amounts and determine the unassigned amounts

that are available for supply. Generate a list of exchange points which can receive more supplies. Randomly select an exchange point from that list and select a source with unassigned goods, and then assign to the exchange point as much as possible. Update the list of exchange points and sources, and their unassigned supply/capacity. Continue this process until all supplies are exhausted. The same procedure will be applied to the second stage (exchange point – destination) of the problem to remove infeasibility.

## **4 Computational Experiences**

Three instances of a small problem (with 3 sources, 2 trans-shipment points and 3 destinations) were solved and the performances of the above two algorithms (SGA and MGA), in term of quality of solutions, were compared. The algorithms were run for 30 times and were allowed to simulate 300 generations in each run.

From the experimental results, it was found that MGA performs better for a two-stage transportation problem. Most of the constraints in the test problems are tight constraints. As a result, it is very hard for SGA to find a reasonable number of feasible offspring in any generation which contributes to the generation of infeasible offspring in the subsequent generations. Although we allow infeasible offspring to be parents in many situations for better convergence, it does not seem to be effective when dealing with too many tight constraints (Sarker et al., 2001a & 2001b). On the other hand, MGA ensures feasibility in every generation.

We then experimented with different sizes of test problems using the MGA algorithm. The test problems were generated by varying the number of sources, exchange points and destinations as shown in Table 1, and the capacity and demand data were generated arbitrarily random fulfilling the assumptions (i) the total supply equals total demand (balanced problem) and (ii) the total capacity of the exchange points is greater than the total supply. The unbalanced problems can also be solved using the same GA either (i) by making the problem balanced with a dummy source or destination as required or (ii) by allowing a minor change in the algorithm. The number of variables and the number of constraints involved in the mathematical models of the test problems are also shown in Table 1.

Table 1: The test problem details

Prob No.	Number of				
	Sources	Exchange points	Destinations	Variables	Constraints
Prob-1	3	2	3	12	10
Prob-2	4	2	4	16	12
Prob-3	5	3	5	30	16
Prob-4	8	4	8	64	24
Prob-5	10	5	10	100	30
Prob-6	12	7	12	168	38
Prob-7	15	10	15	300	50
Prob-8	20	15	20	600	70

Although the quantity transported (decision variables of the model) requires integer values, the developed model can be treated as a linear program because of its special structure (Hillier and Lieberman, 2001). The model, when solving as linear program, would ensure integer values for the variables, provided supply, capacity and demand data are integer, without having any integer variables or integrality constraint in the model. We have solved the corresponding linear programming model using a commercial optimization package LINDO/LINGO. The linear program's objective function value and the outputs from GA are reported in Table 2. We have experimented on a Pentium4, 1.8GHz, 256MB RAM, IBM PC. The GA 'Overall best' column indicates the best objective function value obtained out of 30 individual runs. The individual GA run was terminated when the best objective function value, obtained from that run, was repeated in five consecutive generations. We have not observed any improvement changing the stopping criteria from five to ten generations.

Table 2: Comparing GA solutionals with optimal solutions

Problem	Size (VxC)	Optimum (LINDO) (a)	GA Overall best (b)	% Differ= 100* (b-a)/a
Prob-1	12x10	32200	32200	0.00
Prob-2	16x12	63500	63500	0.00
Prob-3	30x16	62500	63000	0.80
Prob-4	64x24	147200	148863	1.13
Prob-5	100x30	141700	161800	14.18
Prob-6	168x38	86750	112600	29.80
Prob-7	300x50	91300	125700	37.68

From columns 3 & 4 in Table 2, it is clear that MGA provides optimal solutions for smaller problems (problem # 1 & 2) but the quality of solutions deteriorates with the increase of search spaces (/larger problems). The solutions of the larger problems are not acceptable due to high percentage deviation (see column 5) from the optimal solution. It is more likely to have a better coverage for smaller search space (smaller problems) when we use a fixed population size for all problems.

It was observed that the quality of final solutions is dependent on the quality of the initial population. In order to improve the quality of solutions, we have conducted

further experiments by changing the process of initial generation which would provide a better quality initial population. The process can be described as follows.

- I. Set a list of sources with positive supply and destinations with unfulfilled demand.
- II. Select a source (/destination) at random from the list, and find the cheapest route linked to transshipment points from that source (/destination). Any tie will be broken arbitrarily. Assign the maximum possible amount to that route. Update the involved supply (/demand) and available capacity of transshipment point. If the supply is exhausted (/demand is fulfilled), then the source (/destination) would be removed from the list.
- III. Repeat II until all supplies are exhausted and requirements are fulfilled.

We recognize the MGA with the above change as M<sup>2</sup>GA. As shown in column 2 of Table 3, the results of M<sup>2</sup>GA are little improved (as compared to MGA, column 5 of Table 2) although the quality of solutions is still deteriorating with the increase of the search spaces.

#### 4.1 Population Sizes

We run M<sup>2</sup>GA by varying the population sizes as higher population size may be required for better coverage of the search space in case of larger problems. The percentage deviations of the objective function values obtained, from their respective optimal solutions, for population sizes 50, 500, 1000 and 2000 are presented in Table 3. The percent deviations of the average of best values obtained from 30 runs are also shown in Table 4.

Table 3: Percent gap between the best value obtained in 30 runs and the optimal objective function value

Problem	Population Size			
	50	500	1000	2000
Prob-1	0.00	0.00	0.00	0.00
Prob-2	0.00	0.00	0.00	0.00
Prob-3	0.79	0.00	0.00	0.00
Prob-4	1.06	0.00	0.00	0.00
Prob-5	12.17	5.58	3.32	0.78
Prob-6	28.70	7.44	5.76	1.10
Prob-7	32.37	6.68	8.82	3.61
Prob-8	42.97	12.94	10.35	4.52

From Table 3, it is clear that the GA is able to provide optimal solutions for two smaller problems (Prob-1 & 2) with population size 50, and for Prob-3 & 4 with population size 500. As Table 3, for any given population size, it has shown that the quality of solutions deteriorates with the increase of problem sizes (Prob-3 to 8), and for any problem size, the quality of solutions improves with the increase of population sizes. When the population size

is increased from 50 to 500, that means ten times higher, the quality of solutions improves a lot. The change of population sizes from 500 to 1000 or 1000 to 2000 (that is only two times) shows slow improvement. The improvements are more and more appreciable with the increase of problem sizes.

Table 4: Percent gap between the average of best values obtained from 30 runs and the optimal objective function value

Problem	Population Size			
	50	500	1000	2000
Prob-1	0.00	0.00	0.00	0.00
Prob-2	0.00	0.00	0.00	0.00
Prob-3	1.90	0.45	0.08	0.00
Prob-4	1.65	0.43	0.27	0.24
Prob-5	20.56	8.39	7.42	5.04
Prob-6	40.53	10.94	9.96	6.71
Prob-7	55.47	16.21	14.62	10.69
Prob-8	70.52	28.36	20.08	16.24

The percent improvements, based on the average of best values, are highly correlated with the populations sizes. In Table 2, the improvement for problem 7 is better with population size 500 than 1000. However it is due to one run (out of 30 runs) only as we can see this not the case for average performances (see Table 3). From Tables 2 and 3, one can easily conclude that the quality of solutions is highly dependent on the population size specifically when the search space is larger.

#### 4.2 Computational Time

The average computational times per run, recorded in seconds, are shown in two figures (Figure 2 and 3) due to scaling problems. The relationships seem to be polynomial.

Because of the special structure of the mathematical model, the model can be solved as a linear program. For this reason, we think, it is not fair to compare the computational times of GA with that of linear program. However for the readers, we like to mention here that the time taken by the optimization package is about one second for problems 6, 7 and 8.

As shown in Figure 4, the average number of generations required per run, for any test problem, decreases with the increase of population sizes. That supports the fact that the computational time may follow some sort of polynomial patterns.

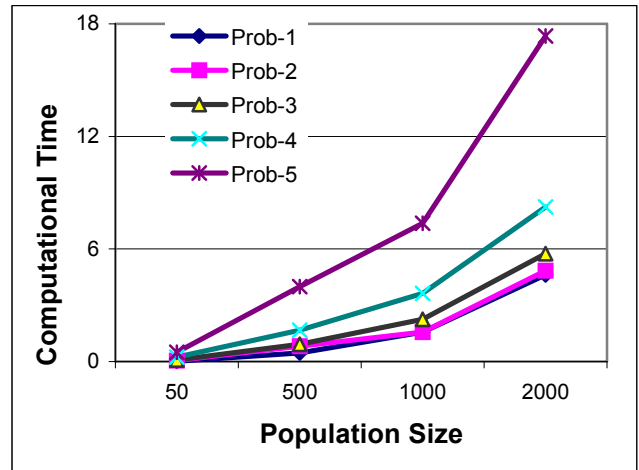


Figure 2: Computational time vs. populations size (Prob-1 to Prob-5)

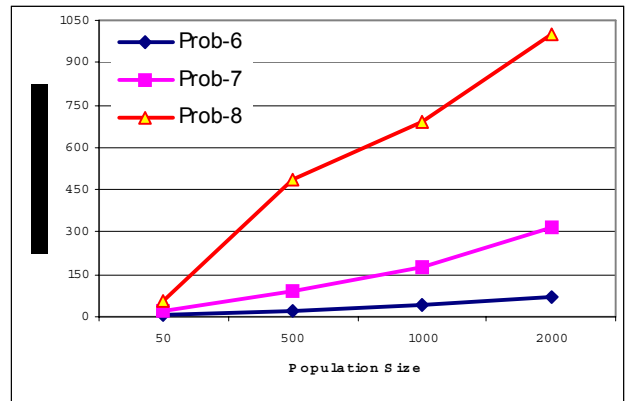


Figure 3: Computational time vs. populations size (Prob-6 to Prob-8)

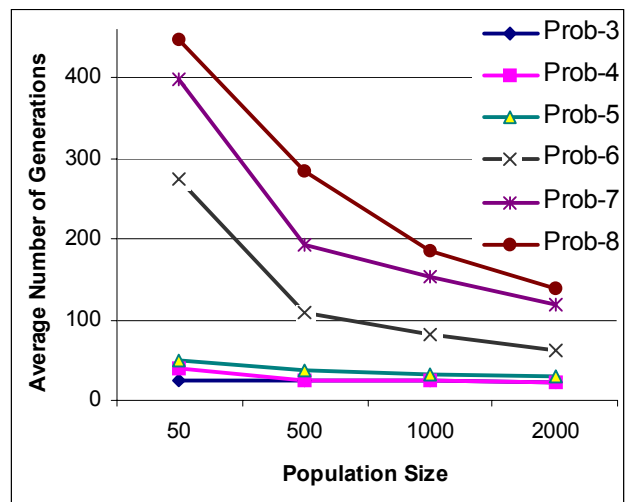


Figure 4: The average number of generations required

#### 4.3 Number of Fitness Evaluations

The average number of fitness evaluated can be calculated as the population size times the average number of

generations required for that population size. The ratios of the number of fitness evaluations (taking the population size 50 as the datum) for different population sizes are given in Table 5.

Table 5: Comparison of the number of fitness evaluations

Problem	Ratio of Population Sizes			
	50:50	500:50	1000:50	2000:50
Prob-3	1.00	9.33	-	-
Prob-4	1.00	6.29	-	-
Prob-5	1.00	7.60	12.68	24.03
Prob-6	1.00	3.96	5.93	9.11
Prob-7	1.00	4.85	7.66	11.88
Prob-8	1.00	6.35	8.31	12.36

Consider the Prob-8 with population size 2000. Although the increase of population size from 50 to 2000 is 40 times higher, the average number of fitness evaluations is only 12.36 times higher which is a favourable point for using higher population size when necessary. We must mention here that the relaxation of stopping criteria (for example allowing more generations) with smaller population sizes does not improve the fitness value.

To ensure the quality within a certain level, the use of population size of 2000 or more for a problem of few hundreds variables and less than 50 constraints would not be a convincing approach to many practitioners. However one should not forget that we have not ensured the best crossover and selection process for solving this problem.

#### 4.4 Overall GA Solutions

The overall GA solutions when run upto population size of 2000 are shown in Table 5.

Table 5: Comparing GA with optimal solutions

Problem	Optimum (LINDO) (a)	GA Overall best (b)	% Differ. 100* (b-a)/a	Corresponding Pop-Size
Prob-1	32200	32200	0	50
Prob-2	63500	63500	0	50
Prob-3	62500	62500	0	500
Prob-4	147200	147200	0	500
Prob-5	141700	142800	0.7763	2000
Prob-6	86750	87700	1.0951	2000
Prob-7	91300	94600	3.6145	2000
Prob-8	171500	181250	5.6851	2000

We believe by changing the selection pressure and other parameters used in running the GA, we would able to improve the performance of the designed algorithm.

## 5 Conclusions

In this paper, we introduced a two-stage transportation problem and formulated its mathematical model. We developed three versions of GA for solving the developed mathematical model and compared their solutions.

We did experiments by changing the problem size and the population size. It is generally known that the quality of solutions improves with the increase of population sizes. This research confirms this general findings once again but for a case of two-stage transportation problem. In addition it shows that it is necessary to increase the population size when the search space is larger. Although it generally dictates that the higher population results in very high number of fitness evaluations (and high computational time), the real number of fitness evaluated (and the computational time required), when increasing the population sizes, is much lower than expected. However, it is an open question how to choose a population, which would ensure a certain quality level, for a given instance of a problem. To answer this question one needs to involve the problem size and complexity, population size and all other methods and parameters involved with the GA to be applied. For unsolved large-scale optimization problems, it is hard to sense whether the algorithm has reached to its optimal solution or not, even though the algorithm converges nicely to a solution (either local or global). As a result, it is a very difficult task to find a general equation for population sizes to be used.

## 6 References

- C. W. Ahn and R. S. Ramakrishna (2002) "A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations", *IEEE Trans on Evolutionary Computation*, 6(6), pp566-579.
- T. Back and H-P. Schwefel (1993) An Overview of Evolutionary Algorithms for Parameter Optimization, *Evolutionary Computation*, 1, 1-24.
- W. Barnett, C. Chiarella, S. Keen, R. Marks, and H. Schnabl (2000) *Complexity and Evolution*, Cambridge University Press.
- L. Davis (1991) *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.
- A. Eiben, R. Hinterding and Z. Michalewicz (1999) Parameter Control in Evolutionary Algorithms, *IEEE Transactions on Evolutionary Computation*, 3(2), pp124-141.
- M. Fischer and Y. Leung (2001) *GeoComputational Modelling Techniques and Applications*, Springer-Verlag, Berlin.

- D. Fogel and L. C. Stayton (1994) On the Effectiveness of Crossover in Simulated Evolutionary Optimization, *ByoSystems*, 32, 171-82.
- D. Goldberg (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA.
- D. Goldberg, K. Deb and J. H. Clark (1992) "Genetic Algorithms, Noise, and the Sizing of Populations", *Complex Systems*, 6, pp333-362.
- D. Goldberg and M. Rudnick (1988) "Genetic Algorithms and the Variance of Fitness", *Complex Systems*, 2, pp265-278.
- G. Harik, E. Canti-Paz, D. Goldberg and B. L. Miller (1999) "The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations", *Evolutionary Computation*, 7(3), pp231-253.
- J. He and X. Yao (2002) "From an Individual to a Population: An Analysis of the First Hitting Time of Population-Based Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, 6(5), pp495-511.
- J. Holland (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, USA.
- F. Hillier and G. Lieberman (2001) *Introduction to Operations Research*, McGraw-Hill, Boston, USA.
- Z. Michalewicz (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, 3<sup>rd</sup> ed., New York, Springer-Verlag.
- M. Sakawa and K. Kato (2003) Genetic Algorithms with Double Strings for 0-1 Programming Problems, *European Journal of Operational Research*, 144, pp581-597.
- R. Sarker, T. Runarsson and C. Newton (2001a) Genetic Algorithms for Solving A Class of Constrained Nonlinear Integer Programs, *International Transaction in Operational Research*, 8(1), pp61-74.
- R. Sarker, T. Runarsson and C. Newton (2001b) A Constrained Multiple Raw Materials Manufacturing Batch Sizing Problem, *International Transaction in Operational Research*, 8(2), pp121-138.
- A. Quintero and S. Pierre (2003) Evolutionary Approach to Optimize the Assignment of Cells to Switches in Personal Communication Networks, *Computer Scmmunications*, 26, pp927-938.
- X. Yao (edited) (1999) *Evolutionary Computation: Theory and Application*, World Scientific Publ. Co., Singapore.
- X. Yao (2002) Evolutionary Computation: A Gentle Introduction, In *Evolutionary Optimization*, R. Sarker, M. Mohammadian and X. Yao (edited), Kluwer, USA, pp27-56