# Incorporating gender and age in genetic algorithms to solve the indexing problem

Diptesh Ghosh

**W.P. No. 2016-03-32**
March 2016



**INDIAN INSTITUTE OF MANAGEMENT**
**AHMEDABAD – 380015**
**INDIA**

# Incorporating gender and age in genetic algorithms to solve the indexing problem

Diptesh Ghosh

**Abstract**

In this paper we propose new genetic algorithms for the tool indexing problem. Genetic algorithms are said to be nature-inspired, in that they are modeled after the natural process of genetic evolution. The evolution process that they model is asexual in which individuals can potentially live forever. In this paper, we propose a genetic algorithm in which solutions are of two genders, reproduction happens by a combination of solutions with different genders, and each solution has a finite life. We compare our genetic algorithms with the best known genetic algorithm for the tool indexing problem and report our computational experience.
**Keywords:** Genetic algorithm, permutation problem, crossover, mutation

## 1 Introduction

The tool indexing problem is an important problem in automated manufacturing. In this kind of manufacturing, a computer numerically controlled (CNC) machine processes a job by performing several operations on it using a number of tools. Since the actual times for performing the operations times are fixed, the processing time of the job can be reduced by reducing the tool change time.

In a CNC machine, toola are changed as follows. The machine has a tool changer that picks up a tool from a specific position called the index position and attaches it to the tool holding mechanism of the machine to perform an operation on a job. Once the operation is complete, the tool changer replaces the tool at the index position. The index position is itself a position in a tool magazine. This tool magazine can be thought of as a disk with slots at equal intervals along its circumference. These slots are either empty or can contain tools. In this paper, we assume that only one copy of each tool can be present in the tool magazine. The tool magazine brings a tool to the index position by rotating either clockwise or anti-clockwise so that the slot with the desired tool is positioned at the index position. The tool changing time can be measured in terms of the total amount of rotation that the tool magazine has to perform to supply the tool changer with all the tools it requires. This rotation is measured in units of "operations", the amount of rotation required to move one slot to the position occupied by its adjacent slot in any direction. The total number of operations required to complete the processing of a job given a tool arrangement is called the cost of the arrangement, and the indexing problem is one of finding an assignment of tools to slots so that the total tool changing time, and hence the total processing time for processing a given job is minimized.

The tool indexing problem is one that has been widely studied in the literature, and the reader is referred to Ghosh (2016a) for a review of the literature. The tool indexing problem, being a computationally hard problem, has often been solved through metaheuristics. In particular genetic algorithms have often been used to solve this problem. Dereli et al. (1998) and Dereli and Filiz (2000) proposed the first genetic algorithm to solve the indexing problem. Their genetic algorithm was non-standard, and recently a standard permutation based genetic algorithm proposed in Ghosh (2016c) has been seen to outperform this algorith. The algorithm in Ghosh (2016c) is a standard genetic algorithm. In this paper, we propose to other genetic algorithms incorporating concepts of gender and age of solutions to develop a better genetic algorithm for the indexing problem.

The remainder of the paper is organized as follows. In the next section we present a description of two genetic algorithms that we propose in this paper. We present the results of our computational experiments with these algorithms in Section 3, and summarize our paper in Section 4.

## 2 Modifications to Genetic Algorithms

The genetic algorithm proposed in Ghosh (2016c) was a distinct improvement on the genetic algorithm proposed in Dereli et al. (1998) and Dereli and Filiz (2000). It was a conventional genetic algorithm in which a new generation was created from an old generation using reproduction, crossover, and mutation operators. The crossover and mutation operators used in OURGA were the alternating edge (AEX) crossover operator and the inversion mutation operator (see e.g., Grefenstette et al. 1985, Larranaga et al. 1999). OURGA is a hybrid genetic algorithm since it periodically exposed solutions to an exchange neighborhood based local search.

In OURGA as in most standard implementations of genetic algorithms, an individual can potentially crossover with any other individual. This genetic algorithm therefore represents a population whose individuals are hermaphrodites. In most higher biological organisms, the population is divided into genders (strictly speaking, into sexes, since sex is a biological construct while gender is a social construct), and crossover of genetic material occurs among individuals of opposite genders. Genetically also, males and females are different from each other in that the number of mutations in the genetic material of males is much more than in females (see e.g., Haldane 1947, Conrad et al. 2011). We model this in the first genetic algorithm that we propose. In this algorithm, called GEN-GA, each generation is divided into two populations, one comprising males, and the other comprising females. Each individual in either population maintains two strings of genetic material, $X$ and $mateX$. The $X$ string defines the individual and does not change over generations. We use this string to compute the cost of the solution represented by the individual. The $mateX$ string is the one that the individual offers to the crossover process. This string starts off by being identical to the $X$ string for the individual, but changes from one generation to the next through mutation. Thus, in our algorithm, the string coding the individual solution is different from the solution that the individual offers for crossover.

The best, i.e., least cost males and females in one generation are directly copied to the next in GEN-GA using the usual reproduction operator. Crossovers occurs between individuals from the male population and ones from the female population. The individuals to be crossed over are chosen through tournament selection using their costs (as computed from their $X$ strings). Once they are chosen, they submit their $mateX$ strings to be crossed over to generate new solution strings. These strings then form the $X$ strings of two individuals (the children resulting from the crossover). One of the two children is added to the male population at random, and the other to the female population. The $mateX$ strings in these individuals are copies of their $X$ strings. Once a new generation is formed, the $mateX$ string of each individual in the new generation is subjected to inversion mutation with a probability that depends on the individual's gender. Based on findings from the literature in biology, we maintain the mutation rate in the male population at a value higher than that in the female population. A pseudocode of the GEN-GA algorithm is given below. In the pseudocode GA-Iter refers to the current iteration of the genetic algorithm, Gen-Size refers to size of the population of each gender in a generation, Max-Gen refers to the maximum number of iterations allowed, Mutate-Prob-Male and Mutate-Prob-Female refer to the mutation probabilities in male and female solutions, and Nr-Reproduce refers to the number of solutions of each gender copied from the current generation to the next generation using the reproduction operation. CURRENTGENMALE and CURRENTGENFEMALE refer to the collections of male and female individuals in the current generation, MATEPOOLMALE and MATEPOOLFEMALE refer to the male and female individuals chosen to mate and produce the next generation, and NEXTGENMALE and NEXTGENFEMALE refer to the collections of male and female individuals in the next generation.

**Algorithm GEN-GA**

1. begin
2. set Best-Sol to $\infty$;
3. set CURRENTGENMALE to a set of Gen-size individuals;
4. set CURRENTGENFEMALE to a set of Gen-size individuals;
   (each individual has a $X$ string and a $mateX$ string; these are identical for each individual)
5. for (GA-Iter from 1 to Max-Gen) begin
6. if (least cost solution coded in any $X$ string in CURRENTGENMALE is better than Best-Sol)
7. set Best-Sol ⟵least cost solution coded in the $X$ string;
8. if (least cost solution coded in any $X$ string in CURRENTGENFEMALE is better than Best-Sol)
9. set Best-Sol ⟵least cost solution coded in the $X$ string;
10. copy the best Nr-Reproduce solutions in CURRENTGENMALE to NEXTGENMALE;
11. copy the best Nr-Reproduce solutions in CURRENTGENFEMALE to NEXTGENFEMALE;
12. set MATEPOOLMALE and MATEPOOLFEMALE to $\emptyset$;
13. for (Count from 1 to (Gen-Size−Nr-Reproduce)) begin
14. obtain solution $P_1$ using tournament selection from CURRENTGENMALE;
15. obtain solution $P_2$ using tournament selection from CURRENTGENFEMALE;
16. add $P_1$ to MATEPOOLMALE and $P_2$ to MATEPOOLFEMALE;
17. end;
18. while (MATEPOOLMALE $\neq \emptyset$) do begin
19. remove a solution $P_1$ from MATEPOOLMALE and solution $P_2$ from MATEPOOLFEMALE;
20. apply the AEX crossover on the $mateX$ strings in $P1$ and $P2$ to obtain two child strings. Create two solutions $C1$ and $C2$. Copy one of the strings to both the $X$ and $mateX$ strings of $C1$ and the other string to both the $X$ and $mateX$ strings of $C2$.
21. add $C_1$ to NEXTGENMALE and $C_2$ to NEXTGENFEMALE;
22. end;
23. for (each solution $P$ in NEXTGENMALE) apply inversion operation to the $mateX$ string in $P$ with probability Mutate-Prob-Male;
24. for (each solution $P$ in NEXTGENFEMALE) apply inversion operation to the $mateX$ string in $P$ with probability Mutate-Prob-Female;
25. set CURRENTGENMALE and CURRENTGENFEMALE to $\emptyset$;
26. copy all solutions from NEXTGENMALE to CURRENTGENMALE and from NEXTGENFEMALE to CURRENTGENFEMALE;
27. end;
28. output Best-Sol;
29. end.

In a genetic algorithm, each individual is potentially immortal, in the sense that if a solution is good (though not optimal), it can be copied using the reproduction operator until the genetic

algorithm terminates. Such immortality is sometimes detrimental to the performance of the genetic algorithm, since it can reduce the diversity in the population (see e.g., Ghosh 2012, for more details on this apect.). This possibility is present, both in OURGA and GEN-GA. In our second genetic algorithm, called AGE-GEN-GA we restrict each individual to be active for a limited period only. We do this by maintaining a lifespan for each individual created. The lifespan is the maximum number of genetic algorithm iterations that an individual will be active in. This lifespan is a randomly generated number within a range. Each time a genetic algorithm iteration is complete, the lifespan of each solution in both the male and the female populations decrease by one. If the lifespan becomes zero, then the individual has lasted long enough, and is not copied into the next generation. The pseudocode for the AGE-GEN-GA algorithm is given below. The variables and collections used in this algorithm are the same as was used in EN-GA. The only addition is the lifespan variable for each individual that stores the remaining lifespan of the individual.

**Algorithm AGE-GEN-GA**

1. begin
2. set Best-Sol to $\infty$;
3. set CURRENTGENMALE to a set of Gen-size individuals;
4. set CURRENTGENFEMALE to a set of Gen-size individuals;
   (each individual has a $X$ string and a $mateX$ string; these are identical for each individual. Each also has an lifespan, assigned randomly to it.)
5. for (GA-Iter from 1 to Max-Gen) begin
6. if (least cost solution coded in any $X$ string in CURRENTGENMALE is better than Best-Sol)
7. set Best-Sol ⟵least cost solution coded in the $X$ string;
8. if (least cost solution coded in any $X$ string in CURRENTGENFEMALE is better than Best-Sol)
9. set Best-Sol ⟵least cost solution coded in the $X$ string;
10. copy the solutions in CURRENTGENMALE to NEXTGENMALE;
11. copy the solutions in CURRENTGENFEMALE to NEXTGENFEMALE;
12. set MATEPOOLMALE and MATEPOOLFEMALE to $\emptyset$;
13. for (Count from 1 to Gen-Size) begin
14. obtain solution $P_1$ using tournament selection from CURRENTGENMALE;
15. obtain solution $P_2$ using tournament selection from CURRENTGENFEMALE;
16. add $P_1$ to MATEPOOLMALE and $P_2$ to MATEPOOLFEMALE;
17. end;
18. while (MATEPOOLMALE $\neq \emptyset$) do begin
19. remove a solution $P_1$ from MATEPOOLMALE and solution $P_2$ from MATEPOOLFEMALE;
20. apply the AEX crossover on the $mateX$ strings in $P1$ and $P2$ to obtain two child strings. Create two solutions $C1$ and $C2$. Copy one of the strings to both the $X$ and $mateX$ strings of $C1$ and the other string to both the $X$ and $mateX$ strings of $C2$.
21. add $C_1$ to NEXTGENMALE and $C_2$ to NEXTGENFEMALE;
22. end;
23. sort the solutions in NEXTGENMALE and NEXTGENFEMALE in non-decreasing order of the costs of the solutions obtained using the $X$ string;
24. Discard all solutions in NEXTGENMALE except the first Gen-Size solutions with a positive lifespan;

```
25.             Discard all solutions in NEXTGENFEMALE except the first Gen-Size solutions
                with a positive lifespan;
26.             for (each solution P in NEXTGENMALE) apply inversion operation to the
                mateX string in P with probability Mutate-Prob-Male;
27.             for (each solution P in NEXTGENFEMALE) apply inversion operation to the
                mateX string in P with probability Mutate-Prob-Female;
28.             set CURRENTGENMALE and CURRENTGENFEMALE to ∅;
29.             copy all solutions from NEXTGENMALE to CURRENTGENMALE
                and from NEXTGENFEMALE to CURRENTGENFEMALE;
30.             reduce the lifespan values of all solutions in CURRENTGENMALE and
                CURRENTGENFEMALE by 1;
31.         end;
32.         output Best-Sol;
33. end.
```

In the next section we describe our computational experience with the three algorithms mentioned in this section.

# 3 Computational Experiments

We coded OURGA, the genetic algorithm of Ghosh (2016c), GEN-GA and AGE-GEN-GA in C, and performed our experiments on them on a machine with Intel i-5-2500 64-bit processor at 3.30 GHz with 4GB RAM. OURGA was coded without the local search component in order to obtain a fair comparison of the genetic algorithms. All the algorithms were implemented in a multi-start manner, with each algorithm being allowed 20 starts, and each start was allowed to run for 500 generations. The results that we propose are the best results obtained over all 20 starts, and the times reported are the total times required to run all 20 starts.

The size of each generation in OURGA (Ghosh 2016c) was taken to be 100. Of these 20 were directly copied from the previous generation using the reproduction operator, and the other 80 were obtained by the AEX crossover operation. The probability of mutation was taken as 0.22 (same as in Ghosh (2016c)). We used a generation size of 100 for our genetic algorithms too. In GEN-GA, 20 of the 100 were copied from the previous generation by the reproduction operator, and the remaining 80 were obtained through crossover. In AGE-GEN-GA, the generation size was 100, the intermediate generation size was 200, of which 100 were the solutions in the last generation, and another 100 were obtained through crossover.

We performed some initial experiments to obtain the best mutation probabilities for the two genders. To do this, we used 25 randomly generated instances. These instances were in five sets with five instances in each set. The numbers of tools and slots in the five sets are given below.

| Set | Tools | Slots |
|---|---|---|
| Set 1 | 30 | 45 |
| Set 2 | 45 | 60 |
| Set 3 | 60 | 75 |
| Set 4 | 75 | 90 |
| Set 5 | 90 | 120 |

We used different combinations of mutation probabilities in male and female solutions to solve these sets. The mutation probabilities in female solutions were varied from 0.05 to 0.30 in steps of 0.05, while the mutation probabilities in male solutions were varied from 0.1 to 0.5 in steps of 0.1. The mutation probabilities in male solutions were kept higher than that in females. The results from our preliminary experiments with GEN-GA and AGE-GEN-GA are given in Tables 1 and 2 respectively. Solution times are not reported in these tables because the execution times required by GEN-GA and AGE-GEN-GA were found to be linear functions only of the number of slots in the instance. The solution costs reported for each set in these tables are averages of solution costs for all five instances in the set. We have italicized the best mutation rate combination for each set.

From the results we see that there is no set of mutation parameters that performs well for all the sets, nor is there any clear trend visible in the results. Thus, to find out a "best" pair of mutation parameters for each genetic algorithm, we found out the percentage suboptimality of each mutation parameter pair for each set. We then averaged the suboptimality values over the five sets, and chose that pair of mutation values for which this average suboptimality was the least. Using this method, we found out that the performance of GEN-GA was best when the male and female mutation probabilities were 0.5 and 0.1 respectively, and the performance of AGE-GEN-GA was best when the male and female mutation probabilities were 0.5 and 0.05 respectively. We used these values in our main experiments.

Our main experiments used data from benchmark instances. These benchmark instances are known in the literature as Anjos instances and sko instances, and have been used to computationally compare the performance of algorithms for the single row plant layout problem. Each benchmark instance contains a symmetrical matrix of frequency values. We use these frequencies to denote the number of times a tool change operation exchanged the particular pair of tools during the processing of a job. The dimension of the matrix is taken as the number of tools used to process the job, and the number of slots is decided by us. The Anjos instances (Anjos et al. 2005) comprise four sets with 5 instances in each set. The number of tools in instances of each of these five sets are 60, 70, 75, and 80 respectively. We have defined the number of slots for these instances to be 100 for each instance. The sko instances (Anjos and Yen 2009) have seven instances. These instances have 42, 49, 56, 64, 72, 81, and 100 tools respectively. There are 60 slots in the first three instances and 100 slots in the other four.

Table 3 reports the costs of the best solutions obtained by the three genetic algorithms in each of the benchmark instance. The first column indicates the name of the instance, and the next three columns report the costs of the best solutions obtained by the specific algorithm for the instance. We have indicated the best among the three costs using italics in the table. We see that apart from three instances, GEN-GA outputs the best solutions for these instances. It also takes the least amount of execution times among the three, and AGE-GEN-GA takes the longest execution times among the three. Thus, from these experiments, it is clear that GEN-GA is the best among the three genetic algorithms, so that we can conclude that coding solutions as individuals with different genders, which have different mutation rates improves the performance of genetic algorithms on the tool indexing problem. Interestingly, the effect of adding a lifespan to individuals does not have a positive effect on the quality of solutions obtained by the genetic algorithm.

# 4 Summary

In this paper we propose two genetic algorithms to solve the tool indexing problem. These algorithms are modifications of the standard genetic algorithm in that they segregate the individuals in a population into two genders and ensure that crossover happens between individuals of opposite gender. This simulates reproduction in most advanced species. Also, based on observations from the biological world, the mutation probabilities in the individuals depend on their gender. In one

of the genetic algorithms, each solution is given a randomly assigned lifespan, corresponding to the lifespans of individuals in the biological world.

We perform experiments comparing the quality of solutions obtained by regular genetic algorithms and the proposed algorithms on benchmark instances. We see that the segregation of solutions into male and female genders produce better solutions than the standard genetic algorithms. However, from our experiments, we are unable to see any advantage in assigning lifespans to solutions.

# References

M.F. Anjos, A. Kennings, and A. Vannelli. A Semidefinite Optimization Approach for the Single-Row Layout Problem with Unequal Dimensions. Discrete Optimization 2 (2005) pp. 113–122.

M.F. Anjos and G. Yen. Provably Near-Optimal Solutions for Very Large Single-Row Facility Layout Problems. Optimization Methods and Software 24 (2009) pp. 805–817 10.1080/00207543.2015.1055351

D.F. Conrad, J.E.M. Keebler, M.A. DePristo, S.J. Lindsay, Y. Zhang, F. Casals, Y. Idaghdour, C.L. Hartl, C. Torroja, K..V. Garimella, M. Zilversmit, R. Cartwright, G.A. Rouleau, M. Daly, E.A. Stone, M.E. Hurles, and P. Awadalla for the 1000 Genomes Project. Variation in genome-wide mutation rates within and between human families. Nature Genetics 43 (2011) pp. 712–714.

T. Dereli, A. Baykasoğlu, N.N.Z. Gindy, and İ.H. Filiz. Determination of Optimal Turret Index Positions by Genetic Algorithms. Proceedings of 2nd International Symposium on Intelligent Manufacturing Systems. (1998) pp. 743–750. Turkey.

T. Dereli and İ.H. Filiz. Allocating Optimal Index Positions on Tool Magazines Using Genetic Algorithms. Robotics and Autonomous Systems 33 (2000) pp. 155–167.

D. Ghosh. A diversification operator for genetic algorithms. Opsearch. 49 (2012) pp. 299–313.

D. Ghosh. Allocating Tools to Index Positions in Tool Magazines using Tabu Search. Working Paper 2016-02-06. IIM Ahmedabad. (2016a)

D. Ghosh. A New Genetic Algorithm for the Tool Indexing Problem. Working Paper 2016-03-17. IIM Ahmedabad. (2016c)

J.J. Grefenstette, R. Gopal, B.J. Rosmaita, D. van Gucht. Genetic Algorithms for the Traveling Salesman Problem. In: Proceedings of the 1st International Conference on Genetic Algorithms (ed. J.J. Grefenstette). Lawrence Erlbaum Associates, Mahwah NJ (1985) pp. 160–168.

J.B.S. Haldane. The mutation rate of the gene for haemophilia, and its segregation ratios in males and females. Annals of Eugenics 13 (1947) pp. 262–271.

P. Larranaga, C.M.H. Kuipers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operators. Artificial Intelligence Review 13 (1999) pp. 129-170.

Table 1: Preliminary experiments using GEN-GA

Set 1

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 104020.8 | 103564.4 | 103338.4 | *102650.2* | 103574.4 |
| 0.10 | | 103223.6 | 103927.0 | 103268.6 | 102716.8 |
| 0.15 | | 103234.2 | 103259.0 | 103331.2 | 103099.4 |
| 0.20 | | | 103028.4 | 103512.6 | 103237.2 |
| 0.25 | | | 103747.8 | 103640.0 | 103548.8 |
| 0.30 | | | | 104024.8 | 103529.0 |

Set 2

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 342789.0 | 342766.2 | 342681.6 | 341814.6 | 342305.6 |
| 0.10 | | 343089.4 | 342176.0 | 341560.4 | 340983.2 |
| 0.15 | | 343000.4 | 342106.8 | 342533.8 | 341996.6 |
| 0.20 | | | 342870.4 | 342436.8 | 342039.8 |
| 0.25 | | | 342116.6 | 342456.2 | 342284.8 |
| 0.30 | | | | 342067.8 | *340121.4* |

Set 3

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 793182.4 | 792215.8 | 790227.4 | 790128.0 | 789188.2 |
| 0.10 | | 790287.6 | 789212.2 | 788392.2 | *787405.8* |
| 0.15 | | 788911.0 | 789646.8 | 787489.0 | 789003.4 |
| 0.20 | | | 788815.6 | 790237.8 | 790108.0 |
| 0.25 | | | 788908.8 | 788809.8 | 788826.6 |
| 0.30 | | | | 790242.2 | 789348.0 |

Set 4

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 1508118.8 | 1509639.2 | 1507309.6 | 1505531.2 | 1504561.6 |
| 0.10 | | 1508148.0 | 1506153.0 | 1503150.4 | 1501541.4 |
| 0.15 | | 1502197.4 | 1506083.8 | 1502266.4 | 1502694.0 |
| 0.20 | | | 1504493.0 | 1502498.4 | 1503746.4 |
| 0.25 | | | 1505200.4 | 1502078.8 | 1503116.6 |
| 0.30 | | | | *1496737.4* | 1503707.2 |

Set 5

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 2799474.8 | 2797276.0 | 2795771.6 | 2787067.6 | 2788489.2 |
| 0.10 | | 2799298.4 | 2788009.8 | 2787605.8 | 2782776.2 |
| 0.15 | | 2789632.8 | 2791340.4 | 2789714.8 | 2785324.2 |
| 0.20 | | | 2786613.0 | *2781656.6* | 2792613.4 |
| 0.25 | | | 2790868.0 | 2787547.8 | 2785489.4 |
| 0.30 | | | | 2790758.2 | 2793907.6 |

Table 2: Preliminary experiments using AGE-GEN-GA

Set 1

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 106933.8 | 105750.2 | 106783.2 | 106488.8 | *105190.2* |
| 0.10 | | 106753.4 | 107276.4 | 106799.8 | 106471.6 |
| 0.15 | | 106342.2 | 106315.0 | 106191.8 | 106359.8 |
| 0.20 | | | 105599.6 | 105838.4 | 106259.2 |
| 0.25 | | | 106495.2 | 105910.0 | 105703.6 |
| 0.30 | | | | 105957.2 | 105750.2 |

Set 2

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 352083.0 | 348281.2 | 351593.0 | 348268.4 | *345393.0* |
| 0.10 | | 348967.4 | 349380.6 | 349717.6 | 347289.2 |
| 0.15 | | 349890.6 | 348316.6 | 345502.4 | 347208.4 |
| 0.20 | | | 346602.8 | 346897.0 | 346159.0 |
| 0.25 | | | 347419.2 | 347164.0 | 347507.2 |
| 0.30 | | | | 348099.4 | 348281.2 |

Set 3

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 802994.8 | 797678.4 | 800266.2 | 797890.8 | 797401.2 |
| 0.10 | | 800756.2 | 799416.4 | 800112.6 | 797955.6 |
| 0.15 | | 801108.6 | 798667.6 | 796062.6 | 796889.0 |
| 0.20 | | | 798607.6 | 801088.4 | 796666.4 |
| 0.25 | | | 799186.0 | *795739.8* | 796066.8 |
| 0.30 | | | | 797893.6 | 797678.4 |

Set 4

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 1521175.4 | 1514128.6 | 1519796.4 | 1519089.8 | 1517536.4 |
| 0.10 | | 1522348.8 | 1518076.6 | 1516935.0 | 1515601.0 |
| 0.15 | | 1521308.2 | 1518929.0 | 1514209.6 | 1514154.8 |
| 0.20 | | | 1519084.8 | 1515997.6 | 1514854.0 |
| 0.25 | | | 1517299.2 | *1513536.4* | 1514222.6 |
| 0.30 | | | | 1516415.8 | 1514128.6 |

Set 5

| | Mutation rates Male | | | | |
|---|---|---|---|---|---|
| Female | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| 0.05 | 2880900.0 | 2857687.2 | 2859529.6 | *2845641.8* | 2861103.8 |
| 0.10 | | 2850695.2 | 2857456.0 | 2863854.0 | 2852064.4 |
| 0.15 | | 2861244.4 | 2865694.6 | 2855057.0 | 2857050.2 |
| 0.20 | | | 2865697.4 | 2854411.4 | 2858422.6 |
| 0.25 | | | 2870937.0 | 2855326.6 | 2860468.8 |
| 0.30 | | | | 2860967.4 | 2857687.2 |

Table 3: Costs of best solutions for benchmark instances

| Instance | OURGA | GEN-GA | AGE-GEN-GA |
|---|---|---|---|
| Anjos instances | | | |
| Anjos-60-01 | 74414 | *69668* | 73845 |
| Anjos-60-02 | 43553 | *39851* | 43961 |
| Anjos-60-03 | 33518 | *33118* | 34002 |
| Anjos-60-04 | 18852 | *17150* | 19314 |
| Anjos-60-05 | 22535 | *21805* | 23342 |
| Anjos-70-01 | 58953 | 58481 | *57055* |
| Anjos-70-02 | 69724 | *67602* | 68443 |
| Anjos-70-03 | 56166 | *55105* | 56929 |
| Anjos-70-04 | 37766 | *36687* | 37475 |
| Anjos-70-05 | 174908 | *170699* | 175273 |
| Anjos-75-01 | 84547 | *83337* | 85790 |
| Anjos-75-02 | 143041 | *139421* | 142622 |
| Anjos-75-03 | 51092 | *49077* | 51078 |
| Anjos-75-04 | 132191 | *131220* | 134305 |
| Anjos-75-05 | 59480 | *58961* | 60329 |
| Anjos-80-01 | 71279 | *69884* | 70943 |
| Anjos-80-02 | 66472 | *65760* | 66764 |
| Anjos-80-03 | 115416 | *115410* | 117817 |
| Anjos-80-04 | 122301 | *120539* | 124216 |
| Anjos-80-05 | 45995 | *45449* | 45953 |
| sko instances | | | |
| sko-42 | 30311 | *30274* | 30871 |
| sko-49 | 43372 | *42908* | 43086 |
| sko-56 | *60476* | 60644 | 60608 |
| sko-64 | 119499 | *114162* | 121644 |
| sko-72 | 161859 | *157073* | 162586 |
| sko-81 | 215063 | *212681* | 215197 |
| sko-100 | *325670* | 327348 | 326019 |