MIDDLE EAST TECHNICAL UNIVERSITY

Maximal Covering Location Problem with Genetic Algorithm

Lecturer: Prof.Dr. Nur Evin Ozdemirel

Sırma Karakaya
Orhan Toprakman

9.1.2019

MIDDLE EAST TECHNICAL UNIVERSITY

1. Introduction

Facility location models have been a fundamental research area among the practitioners and scholars. Brandeau and Chiu (1989), Revelle and Eiselt (2005) and and Daskin (2008) provide a detailed introduction of facility location problems and explains each of its application areas. One of the famous facility location problems is the covering location problems (CLP) as it has wide area of applications in public and private sectors, including locating warehouses, sensors in wireless networks, fire stations, pump stations, schools, emergency centers, and etc. In addition, this model can be applied to data abstraction to portfolio formation. (Aytuğ and Saydam (2002), Shahanaghi and Ghezavati (2008)).

CLPs are often utilized when there is no enough budget, or human resources to cover all the demands. It seeks to cover a set of demands while one or more objectives are to be optimized. The CLPs are often categorized as Set Covering Location Problem (SCLP) and Maximal Covering Location Problem (MCLP). While SCLPs aim to cover all the demands with the least number of facilities, a classical MCLP aims to locate a given number of facilities so that maximum amount of demands is covered. A facility covers a demand point if the distance between the demand nodes and the facility is less than a pre-determined coverage radius (Aytuğ and Saydam (2002)). The MCLP was first introduced by Church and ReVelle (1974) on a network and since then different extensions of the MCLP were introduced by different studies. (Aytuğ and Saydam (2002), Shahanaghi K., & Ghezavati (2008), Lee and Lee(2010), Atta et al. (2017), Farahani et al. (2014)) Due to the application areas of the problem, MCLP is often needed to be solved for large instances but it is difficult to solve them in a reasonable amount of time. Therefore, utilizing heuristics becomes inevitable. In literature, variants of heuristics for different extensions of MCLP are studied.

In this study we propose a modified version of genetic algorithm (GA) to solve the MCLP for large instances. GA is known to be one of the powerful tools that is able to solve NP-hard problems in reasonable computational times while yielding near-optimal solutions. It has 5 main steps: 1) generation of initial population, 2) parent selection, 3) crossover operation, 4) mutation operation, 5) new population selection. We aim to improve the performance of GA by adding some local search strategy to the heuristic. One of the main criticisms for GAs is although it performs well in large and complex search area, it is not good enough in fine-tuning the solutions that are near to optimal solution (Atta et al. (2017)). They suggest that incorporating local search to GA may improve the performance of it. Inspiring from this suggestion we consider making the repair procedure according to some local search which we call it as Max_Rep and explain it in details in §3.2.1

The outline of this study is as follows. In §2 we first provide the mathematical formulation of the MCLP and the related literature. In §3 we explain the proposed GA. In §4 we provide the results of computational analysis and in §5 we provide general remarks obtained through the study and discuss future research directions.

2. Problem Definition and Literature Review

The MCLP locates p facilities to maximize the number of covered demands. The model allows some nodes to be uncovered if the necessary number of facilities to cover all nodes exceeds p. For the formulation of MCLP, the following notations are used:

I: set of demand nodes
J: set of candidate sites
i: index of demand nodes, i∈ I
j: index of candidate site, j∈ J
h_i: demand at demand node i

Decision variables:

x_j: x_j equals 1 if we locate candidate site j and 0 otherwise.
z_i : z_i equals 1 if node i is covered and 0 otherwise.

The MCLP problem can be modeled as an integer problem as follows:

$$\text{Max} \quad \sum_{i \in I} h_i z_i \tag{1}$$

$$s.t \quad \sum_{j \in J} x_j = p \tag{2}$$

$$z_i - \sum_{j \in N_i} x_j \leq 0 \quad \forall i \in I \tag{3}$$

$$x_j \in \{0,1\} \quad \forall j \in J \tag{4}$$

$$z_i \in \{0,1\} \quad \forall i \in I \tag{5}$$

The objective function (1) maximizes the number of covered demands. Constraint (2) ensures that exactly p facilities are located. Constraint (3) links the location and coverage variables. It allows z_i to equal 1 only when or more facilities that can cover node i is located. Note that in (3) N_i corresponds to the set of candidate sites that may cover demand node i. D^c, and d_{ij} correspond to the coverage distance and distance between node i and j, respectively, and then $N^i = \{j \in J: d_{ij} \leq D^c\}$. Constraints (4) and (5) are the binary restriction constraints. To solve MCLP, linear programming can be considered, by relaxing the binary restrictions. Studies applying LP solution obtain 80% of time all integer solutions. If the solution obtained through LP is not all integers, applying branch and bound algorithm is generally used to produce integer solutions. However, computational results reported on these problems is restricted to small and medium scale problems. (Galvao and Revelle (1996)).

Because the MCLP is NP-hard, many researchers propose various heuristics to solve it for large instances. Church and Revelle (1974) is one of the pioneering studies that introduce algorithms to solve MCLP. The authors introduce Greedy Adding and Greedy Adding with Substitution algorithms. Computational experiments are done with 55 demand nodes, with different p values. Weaver and Church (1984), Galvao and Revelle (1996), Loreana and Pereira (2002), and Senne et al. (2010) develop Lagrangian heuristic for the MCLP. The experimental results of Weaver and Church (1984) are inconclusive. Lorena and Pereira

(2002) are unable to measure their solution quality due to non-continuous nature of the cost parameter. In the paper of Galvao and Revelle (1996) upper bounds are obtained by vertex addition and lower bounds are obtained through a sub-gradient optimization algorithm. With the proposed algorithm the authors are able to find optimal solutions for 55-vertex network problem. For a 150-vertex problem the average duality gap is found to be 4.39%, while the maximum gap is 7.66%. Senne et al. (2010) introduce a decomposition heuristic based on Lagrangian relaxation to obtain the upper bounds for the optimal solution. The authors suggest that the upper bounds obtained through decomposition are better than the ones obtained through the integer restriction relaxation.

Adenso-Diaz and Rodrigues (1997), Lee and Lee (2010), propose Tabu-search algorithms for the MCLP. Adenso-Diaz and Rodrigues consider MCLP in the context of locating ambulances in rural regions. Their experimental setting consists of 213 demand nodes and p is in the range of [16, 25]. The paper does not provide any computational comparison with the other heuristics or the optimal solutions but the authors claim that their proposed TS algorithm offers good results in short times. Lee and Lee (2010) consider a variant of MCLP; hierarchical covering location problem. For the experimental setting; they set the number of nodes as {20, 50,100,150}. The algorithm is able to find solutions on average 3.12% away from the optimal solution. When n=150, in the worst case the solution is 5.82% away from the optimal solution. The authors claim that the algorithm often gives good solutions within reasonable computational time.

Farahani et al. (2014) also consider hierarchical covering location problem and propose artificial bee colony (ABC) algorithm. Their findings indicate that for small instances (problems with up to 500 candidate locations), ABC finds solutions with at most 1% deviation from the optimal solution, and the computation times are significantly small (the greatest computation time is not more than 6 minutes). They also test the algorithm for large instances (problems with up to 1600 candidate locations) and show that the algorithm works fast even for such large instances. The solution qualities cannot be compared with any optimal solution or lower bound. Zarandi et al. (2012) consider another extension of MCLP; the dynamic one and propose simulated annealing (SA) heuristic to solve the problem. Their proposed algorithm is capable of solving problems with up to 2500 demand nodes and 200 potential facilities. The experiments show that the proposed algorithm finds solutions with errors less than one percent.

Jaramillo et al. (2002) study genetic algorithm (GA) for several facility location problems including the MCLP. For experiments two data sets are used. The first one is obtained from the study of Daskin (1995) which consists of 88 cities, and the second set is obtained by modifying the one from the literature which is a 150-node network. Their results indicate that GA performs better than Lagrangian heuristic for both problem sets. They also consider GA followed by substitution. The substitution procedure takes a solution and attempts to improve it using a greedy heuristic. Findings indicate that GA with substitution performs the best among pure GA, Lagrangian heuristic and Lagrangian heuristic with substitution. However, GA with substitution and pure GA are computationally expensive compared to Lagrangian heuristic. Atta et al. (2017) propose GA for the MCLP with local refinement. Local refinement strategy is adopted during the initial generations to select the facility locations. After mutation, each chromosome is locally updated. First they cluster customers with respect to assignments to the facilities. Each cluster is formed around a facility. Then according to the total weighted sum of distances of the facilities within

a cluster, each facility location is updated. The proposed heuristic is demonstrated on several benchmark data sets. The results show that GA with local refinement outperforms the existing approaches in most of the cases. Aytuğ and Saydam (2001), Shahanaghi and Ghezavati (2008), Zarandi et al. (2011), Yoon and Kim (2013) are also some studies that propose GA for variants of the MCLP.

3. The Proposed Solution Procedure

GA is first proposed by Holland (1974). It is a search algorithm analogous to natural selection's survival of the fittest (Atta et al. (2017) and Aytuğ and Saydam (2002)). From the mid-70s to date, various types of GA have been employed in problems such as vehicle routing, scheduling, and facility location problems. The main idea behind GA is to improve generations (set of solutions) by applying reproduction mechanisms such as crossover and mutation. In each iteration, a set of parents are selected from the generation and based on their fitness value they are selected to produce offspring (new solutions). Then based on selection strategy, the population is updated. The procedure continues until a termination criterion is satisfied. In this paper we use GA to solve the maximal covering location problem. Instead of using conventional GA we consider GA with local refinement to improve the performance of the algorithm. In the following, first we provide pseudocode of the algorithm and then explain it.

Pseudocode for the proposed GA

```
while(iteration < number of iteration)  //Satisfied Vector Calculation
      for(i=0; i< number of nodes;i++)
            for(j=0;j<number of nodes; j++)
            if((chromosome[i]==1) && (distance[i][j]< range));
                  satisfied_vector[j]=1;
                  =0, otherwise ;
            endif
            endfor
      endfor
      for(i=0;i<number of nodes;i++)      //Fitness Value Calculation
            fitness=satisfied_vector[i]*node_demand;
      endfor

      for(i=0;i<number of nodes;i++)  //Probability Calculation
            prob[i]=(fitness[i]-fitness(min))/(fitness(max)-fitness(min));
      endfor
      crossover();
      repair();

      randomly swap two elements (1 and 0) of chromosome; //Mutation Operator

      for(i=0;i<number of parents;i+=2)
      //Replace parents with offsprings with probability
            for(j=0;j<2;j++)
                  if((offspring[i].fitness>parent[j].fitness)&&(random<replace
                  ment_probability))
                        replace (offspring[i],parent[j]);
                  endif
            endfor
      endfor
```

3.1 Representation Schemes

We introduce the chromosome as a binary string, where "1" corresponds to an open depot and "0" corresponds to closed depots. Our chromosome size is |J|, where J is the set of the candidate sites. We also keep a satisfied vector, again as a binary string. Its size is |I| where I is the set of demand points. The vector keeps the set of demand points. If a demand point is covered by an open facility, it takes value "1" and "0" otherwise. Note that the satisfied vector takes the values according to Constraint (3) explained in §2. We keep the satisfied vector to be able to calculate the fitness function of the solutions. Crossover and mutation operations are applied only on the chromosome, not on the satisfied vector. Below, Figure [1] shows the chromosome and satisfied vector structure for an instance where |J|=10, |I|=10 and p=4. Chromosome representation shows that at sites {1, 5, 6, 8} facilities are opened. The satisfied vector representation shows that demand points {1, 2, 4, 5, 6, 8} are covered by the open facilities.

3.2 Crossover Operators

For the crossover operation we consider both 1 point and 2-points crossovers. For 1-point crossover, we select the point randomly between 1 to j. For 2-points we choose two crossover points for the first and the second parts of the chromosomes, which we denote them by n_1 and n_2, respectively. Both n_1 and n_2 are randomly selected from 1 to j. The first parts of the chromosomes belonging to the parents are divided by the point n_1 and the first n_1 parts of the chromosomes are exchanged between the parents. In a similar way, the second part of the chromosome is divided by point n_2 and then the remaining j- n_2 parts are exchanged between the parents. We consider both making crossover with 2 parents and 3 parents. For the 2 parents both crossover operations are considered, for the 3 parents we only consider 2-points crossover. Below, Figure [2] and [3] show the considered crossover operators for both 2 and 3 parents settings, respectively. Note that when we have 3 parents, we obtain 6 offspring.

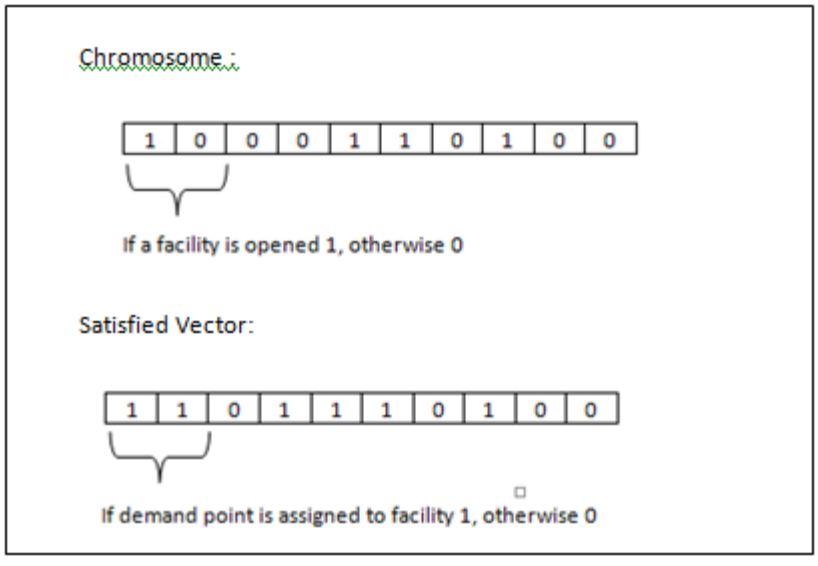


Figure [1]. Chromosome and Satisfied Vector representation

When we observe the offspring from Figures [2] and [3] we observe that after the crossover some offspring correspond to infeasible solutions. The number of open facilities becomes either smaller or larger than the predetermined number. In order to avoid infeasibility, we need to have a repair operation on the offspring. In the following we explain the considered repair operations.

3.2.1 Repair Operators

We consider three repairing operations which we introduce them as randomly repairing (Rand_Rep), greedy repairing (Gre_Rep), and maximum improving repairing (Max_Rep). Note that independently from the repairing operator, we always update the satisfied vector after the repairing, so that the fitness values are also updated.

Random Repairing Operator: When the crossover results in less number of open facilities than the predetermined number, the repair operator randomly opens new facilities until the predetermined number of facilities are opened. Similarly, when the crossover results in higher number of open facilities than the predetermined number of facilities, the repair operator closes facilities randomly until the predetermined level is reached. The operator does not take into account the demand coverage levels with an open facility. Here we present the pseudocode of the operator:

Pseudocode for the random repair operator

```
Rand_Rep (chromosome x)
//If total facility is under/over p then randomly assign 1/0 respectively to
satisfy total facility=p

      for (int i = 0; i < number of nodes; i++)
            sum += x[i];
      endfor

      if (sum < p)
            while (sum < p)
                  if (x.[rand_num] == 0)
                        x.[rand_num] = 1;
                        sum += 1;
                  endif
            endwhile
      endif

      if (sum > num_facility)
            while (sum > num_facility)
                  if (x.[random] == 1)
                        x.[random] = 0;
                        sum -= 1;
                  endif
            endwhile
      endif
return x;
```

Greedy Repairing Operator: When the crossover results in less number of open facilities than the predetermined number, for each closed facility, the operator calculates the maximum demand that can be satisfied from that facility. It calculates the maximum amount of demands that can be satisfied by

that facility by considering what happens if only that closed depot is opened and the others are closed. After calculating the maximum demands that can be satisfied by each closed facility, it opens facilities in the order of decreasing amounts of satisfied demand, until the predetermined number is reached. When the number of facilities is more than the pre-determined number of facilities, the operator closes the facilities in the order of increasing amounts of satisfied demand (i.e. first close the one that satisfies the least amount, then the one with higher and so on).

Pseudocode for greedy random repair operator

```
order(); //order nodes according to satisfaction portion if facility is opened
at that node

Gre_Rep(chromosome x)
//Assign missing facilities according to satisfaction portion

      for (int i = 0; i < number of nodes; i++)
            sum += x[i];
      endfor

      while (sum < p) {
            for (int i = 0; i < number of nodes; i++)
                  if ((x[i] == 0) && (random < repairing_probability))

                        x[i] = 1;
                        sum++;
                  endif
            endfor
      endwhile

      while (sum > p)
            for (int i = number of nodes; i > -1; i--)
                  if ((x[i] == 1) && (random < repairing_probability))
                        x = 0;
                        sum--;
                  endif
            endfor
      endwhile
return x;
```

Maximum Improving Repairing Operator: If the number of open facilities is less than the predetermined number of facilities, the operator evaluates each closed depot based on the improvement amount on the total fitness value and opens depot each time considering which improves fitness value most. (i.e. first, opens the one which improves the fitness function most, and then again calculates fitness of the current solution and opens which improves most and so on). If the facilities are more than the predetermined number, then the operator closes facilities each time which reduces fitness value smallest. Note that while the greedy repairing algorithm calculates the improvement, in terms of what happens if the closed facility is opened and all the others are closed, maximum improving repairing operator calculates the marginal improvement of opening a closed facility.

Pseudocode for maximum improving repair operator

```
Max_Rep(chromosome x) //Assign missing facilities which improves best
      current_fitness = x.fitness;
      chromosome temp;
      temp = x;
      int deviation[num_node];

      for (int i = 0; i < number of nodes; i++)
            sum += x[i];
      endfor

      if (sum < p) {
            while (sum < p)
                  for (int i = 0; i < number of nodes; i++)
                        if (temp[i] == 0)
                              temp[i] = 1;
                              calculate_satisfied_vector(temp);
                              calculate_fitness(temp);
                              deviation[i] = temp.fitness - current_fitness;
                        endif
                  temp = x;
                  endfor

                  find maximum deviation[i];
                  x[maximum deviation index] = 1;
                  sum++;
            endwhile
      endif

      if (sum > p) {
            while (sum > p)
                  for (int i = 0; i < number of nodes; i++)
                        if (temp[i] == 1)
                              temp[i] = 0;
                              calculate_satisfied_vector(temp);
                              calculate_fitness(temp);
                              deviation[i] = current_fitness - temp.fitness;
                        endif
                        temp = x;
                  endfor
                  find minimum deviation[i];
                  x[minimum deviation index] = 0;
                  sum--;
            endwhile
      endif
return x;
```

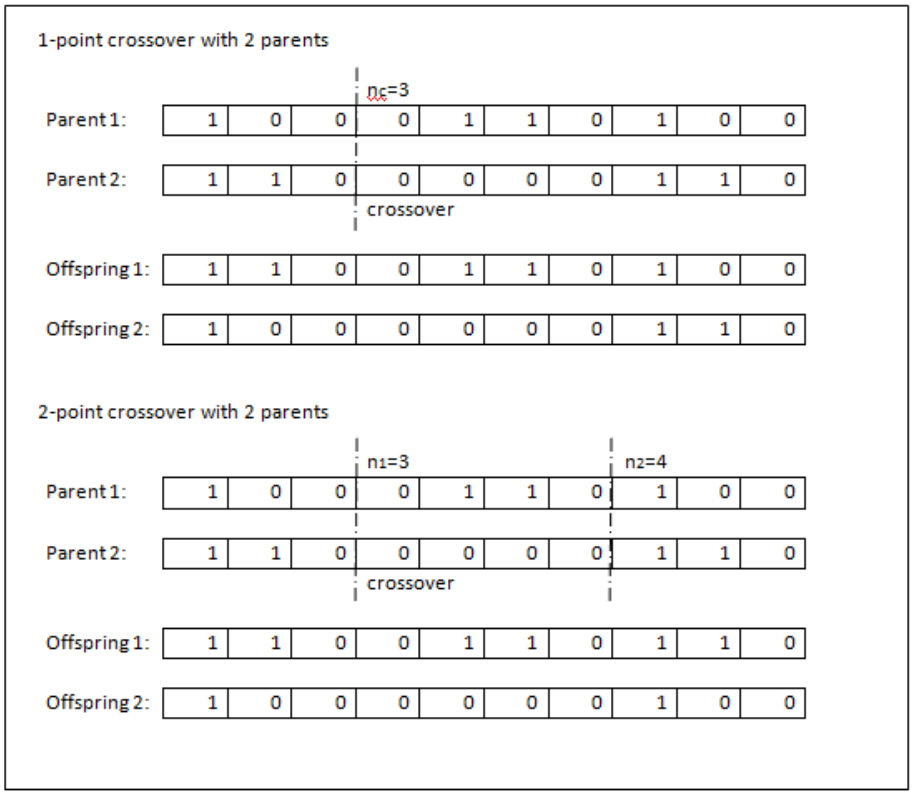


Figure [2]. Crossover operations with 2 parents

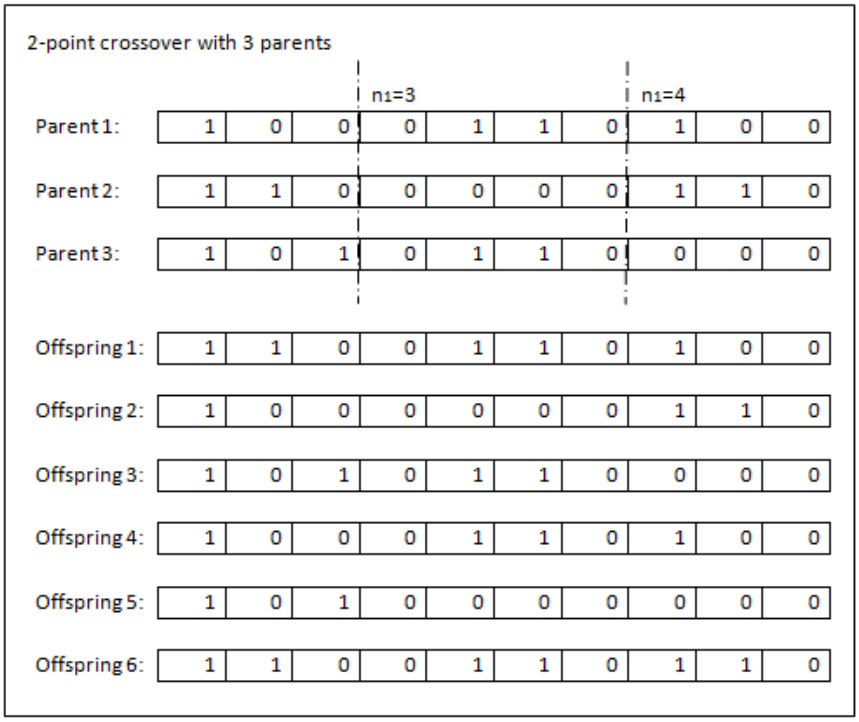


Figure [3]. Crossover operations with 3 parents

3.3 Mutation Operator

To avoid the early convergence near a local optimal solution, the mutation needs to be operated on the chromosomes with probability p_m. We randomly select an open facility and make it closed and randomly select a closed facility and make it opened. The mutation does not cause infeasible solutions because we always open and close the same number of facilities. After the mutation, we always update the satisfied vector. Below, Figure [4] shows the mutation operator for an instance.

Mutation Operator: (Open facility 3 and close facility 2)

Offspring 1:	1	1	0	0	1	1	0	1	0	0
Satisfied Vector:	1	1	0	0	1	1	0	1	0	0
Offspring 1:	1	0	1	0	1	1	0	1	0	0
Satisfied Vector:	1	1	1	0	1	1	0	1	0	1

Figure [4]. Mutation on an offspring

3.4 Initial Population, Fitness Function, Parent Selection, and Stopping Criteria

Initial Population: To generate initial population, genes are generated randomly. In the proposed algorithm, to generate the initial population we determine the number of facilities to open and generate solutions accordingly. Then according to the open facilities, we observe the distances between the open facilities and the demand points, and based on the coverage radius we obtain the satisfied vectors for each generated solution. As we obtain solutions with p many open facilities, all the generated solutions are feasible.

Fitness Function: The fitness function is calculated using the objective function of the mathematical model.

Parent Selection: At the beginning of the algorithm parent selection process is done according to Roulette Wheel Selection (RWS). Based on the fitness values, probability of selecting each chromosome is calculated according to (6). Then all the probabilities are normalized. In (6) prob[i] denotes the probability of selecting chromosome i, fitness [i] denote the fitness value of the chromosome i and $fitness_{min(max)}$ corresponds to the minimum (maximum) fitness values in the population.

$$prob[i] = \frac{fitness[i] - fitness_{min}}{fitness_{max} - fitness_{min}} \qquad (6)$$

When we obtain offspring, we replace parents with better offspring with a replacement probability. Note that since we replace the parents with better offspring with some probability, sometimes we may

continue with worse solutions in the generation. It is a better approach to avoid from the early convergence.

Stopping Criteria: To terminate from the algorithm we consider two stopping criteria: (i) algorithm reaches the maximum number of iterations; (ii) the best solution is not changed for a reasonable amount of time (if significantly higher than the average computation times).

4. Experimental Results

In this section we first introduce the test problems and then provide the results of the parameter tuning. In the last part, we provide the performance of our heuristic, with respect to the other heuristic proposed in different studies. For the algorithms C++ programming language, and problem solutions C++ interfacing with callable library in ILOG CPLEX version 12.8 is utilized. The related codes are illustrated in Appendix E. All the experiments have been run on a computer with 4GB RAM, and Intel Core 2 Duo 2.00 GHz processor, under Windows 7. For the sake of simplicity, we provide Table [1] that shows the notations for the parameters used in this section.

4.1 Test Problems

The performance of the proposed algorithm is tested with both real-world data and random data sets. The real-world data are obtained from geo-referenced database of São José dos Campos' city, Brazil. The data sets, SJC324, SJC402, SJC500, SJC708, and SJC818 consist of 324, 402, 500, 708, and 818 vertices, respectively. Lorena and Pereira (2002) and Atta et al. (2017) also use these data so we are able to compare our heuristic performance with their proposed heuristic on these test problems. The data sets are available in the following website: http://www.lac.inpe.br/~lorena/instancias.html. We also generate random data sets to observe our heuristic performance for larger instances. To generate data with 1800 nodes we follow the same procedure with Zarandi et al. (2012) and Atta et al. (2017). The locations of the nodes are randomly generated using a uniform distribution between 0 and 30 for x and y coordinates. The distances between the nodes are defined as the Euclidean distances. Populations on the nodes are generated using a uniform distribution between 0 and 200. We call this data instance as SO1800.

I	Iteration number
N	Population size
N_i	Number of parents to be selected
p_m	Mutation probability
p_{rep}	Replacement probability
S	Coverage radius
p	Number of facilities to be opened

Table [1]. Notations for the parameters used in §4

.**4.2 Parameter Tuning**

For our algorithm we need to set the stopping criteria (number of iterations (I), mutation probability (p_m), population size (N), parent number to be selected (N_i), and replacement probability (p_{rep}). Note that in our algorithm we do not have a crossover probability. Any selected parents go to crossover operation but we replace the parents with better offspring with some replacement probability. Therefore, it should be selected appropriately to obtain the best performance from the algorithm. To tune the parameters (N, N_i, p_{rep}) we test the algorithm on test instances SJC324, SJC500, and SJC 818 with p=5 and S= 800. Each problem is solved 10 times and then the worst, average and the best solutions are identified for each parameter setting. As mentioned in §3 we consider 3 different crossover and 3 different repair operators for the algorithm. We consider all the combinations and therefore obtain 9 different algorithm settings. Table [2] shows the problem descriptions and the related abbreviations for these settings. For all the algorithm settings, we try to find the best parameter settings so that during deciding the best algorithm setting, we make fair comparisons within each setting.

For the mutation probability we utilize from the literature. Mahmoudi and Shahanaghi (2013) consider the mutation probability as 0.4, Zarandi et al. (2012) set the probability to 0.25 and Atta et al. (2017) set the value to 0.01. We consider mutation probabilities as {0.01,0.05,0.5} with different parameter settings and observe that there is no much difference between 0.01 and 0.05, and using 0.5 provides worse results on the average when problem sets SJC324, SJC500 and SJC818 are solved 10 times. Therefore, we set the mutation probability to 0.05 for all the runs. In order to decide the number of iterations we test each algorithm setting on problem instance SJC818. Our findings indicate that the minimum necessary number of iterations for the convergence is 50 for Rand_Rep(2-2) , N=150, N_i =30, p_{rep} 0.4, p_m =0.05, P=5, S=800, and the maximum necessary number is 500 with Max_Rep(2-1) with the same parameter settings. However, with this iteration number the computation time significantly increases and there are no significant solution differences between the one found at I=300 and I=500. Figure[1] and [2] show the solution values and the computation time with respective to iteration number, respectively. Therefore, we set I=300 for all the runs.

Rand_Rep(2-1)	GA with 2 parents 1 point crossover and random repairing
Rand_Rep(2-2)	GA with 2 parents 2 points crossover and random repairing
Rand_Rep(3-2)	GA with 3 parents 2 points crossover and random repairing
Gre_Rep(2-1)	GA with 2 parents 1 point crossover and greedy repairing
Gre_Rep(2-2)	GA with 2 parents 2 points crossover and greedy repairing
Gre_Rep(3-2)	GA with 3 parents 2 points crossover and greedy repairing
Max_Rep(2-1)	GA with 2 parents 1 point crossover maximum improving repairing
Max_Rep(2-2)	GA with 2 parents 2 points crossover and maximum improving repairing
Max_Rep(3-2)	GA with 3 parents 2 points crossover and maximum improving repairing

Table [2]. Algorithm settings and the abbreviations

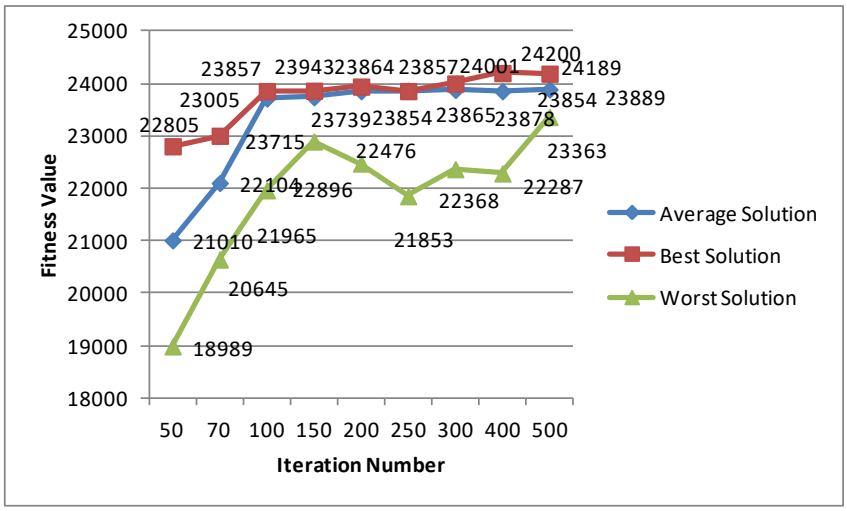


Figure [1]. Fitness function values with respect to iteration number for Max_Rep(2-2) N=150, N_i =30, p_{rep} 0.4, p_m =0.05 , P=5, S=800

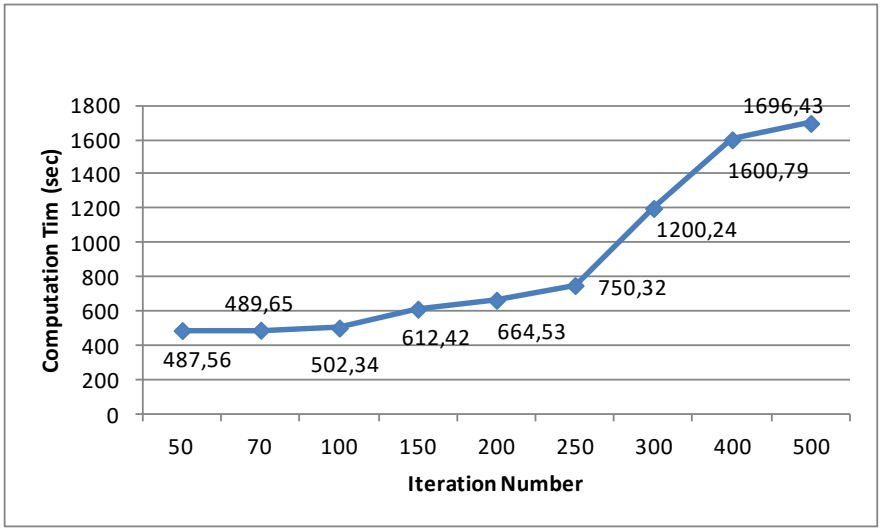


Figure [2]. Computation time vs. iteration number for Max_Rep(2-2) N=150, N_i =30, p_{rep} 0.4, p_m =0.05 , P=5, S=800

For the other parameters; we consider N= {50,150}, N_i= {10, 30}, and p_{rep}= {0.4, 0.7}. The results for 10 runs for each instance are provided in Tables [3], [4] and [5] in Appendix A. We also provide the summary of the results (the worst, average and the best solutions and the gap between the optimal and the average) in Tables [6], [7] and [8] in Appendix B. We analyze the results with the statistic software Minitab 17. The detailed analysis is provided in Appendix C. In order to decide which algorithm setting is the best, we first decide the best parameter settings for all the problem sets. We make it to ensure that we compare different algorithm settings in a fair way and be able to choose the best algorithm setting.

After identifying the best parameter settings for each problem set under each algorithm setting, we solve each problem again for 10 times with the best parameter settings and apply ANOVA test for each problem. ANOVA test enables us to analyze whether there are statistically significant differences among the three algorithm settings or not. For each problem set, we provide the ANOVA results in Appendix D. To measure the performance of each setting, we compare the average results obtained from each 10 runs. We do not measure the computation time as a performance measure, because we know that there is open door to improve our coding, computation times for each setting may significantly change if we improve it. Our findings indicate that for SJC324 the three algorithm settings are not statistically different (p=0.850 is found to be greater than α=0.05), which shows that in terms of the performance all the settings yield similar performances. We also apply Tukey's statistical test to compare the pairwise means. For SJC500 we find that there are significant differences among the three algorithm settings (p=0.03 < α=0.05). For SJC818, ANOVA test shows that the three algorithm settings are significantly different from each other (p=0.02 < α=0.05). Tukeys' test results show that there is significant difference between Max_Rep and Gre_Rep. Max_Rep yields a higher average than the Gre_Rep and therefore performs better. That is why for all the instances, we decide to use Max_Rep operator instead of Gre_Rep. We also consider Rand_Rep and observe the performances of these settings with respect to other heuristics proposed in other studies.

4.3 Computational Results

We test our algorithm with two settings, GA with Max_Rep and GA with Rand_Rep. Datasets SJC324, SJC402, SJC500, SJC708, SJC818 and SO1800 are used with different n (number of demand points), p (number of facility) and S (range) values. And also, optimal results are calculated with CPLEX. In each test whenever algorithm reaches the optimal value, it stops. If it does not reach the optimal value then it stops in case improvement is not obtained in a reasonable time. We also compare the algorithm performance with the results that were obtained by Lorena and Pereira (2002) and Atta et al. (2017). Results are presented at Tables [9] [10],[11],[12],[13], and [14] on pages 19-21. Optimal results are marked bold. We test our algorithm with totally 57 datasets. According to the results; Max_Rep finds optimal values of 39 datasets, on the other hand Rand_Rep finds optimal values of 18 datasets. We also present the pairwise comparison of algorithms. Numbers are calculated such that way; first, solution quality is compared. It may be noted that coverage percentage is important as facilities are built for only once, therefore, we compare coverage percentage first. If one of the pairs has better quality then we select that algorithm as dominating without considering computation time. If both of them have the same quality, then we compare computation times. Also, pairwise coverage percentage difference are presented in Figure[3-8] as stack bar chart for SJC 708 and SJC818 comparing Max_Rep with benchmark heuristics.

Number of Domination	Rand_Rep	Max_Rep	Lorena et.al (2002)	Atta et.al (2017) GA w/o Ref	Atta et.al (2017) GA with Ref
Rand_Rep	-	6	4	1	0
Max_Rep	49	-	5	8	9
Lorena et.al (2002)	51	50	-	19	14
Atta et.al (2017) GA w/o Ref	54	47	36	-	1
Atta et.al (2017) GA with Ref	55	46	41	54	-
Total Optimal	18	38	53	45	53

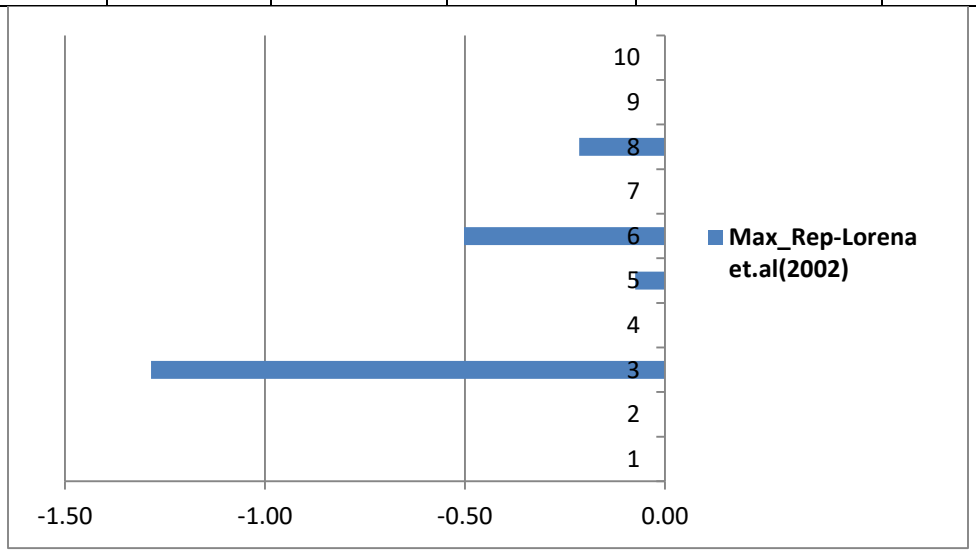


Figure [3]. Difference of Percentage Coverage in SJC708 (Max_Rep-Lorena et.al(2002))

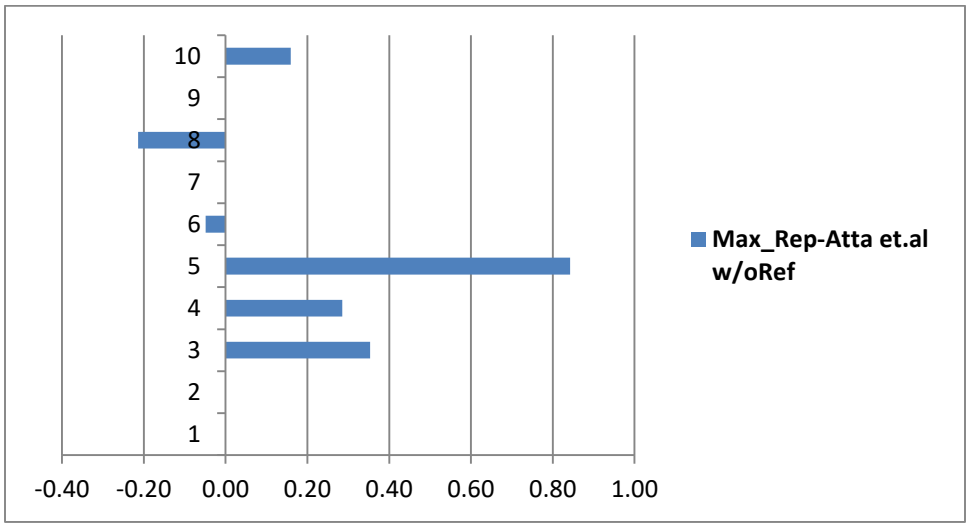


Figure [4]. Difference of Percentage Coverage in SJC708 (Max_Rep- Atta et.al(2017) GA w/o Ref)

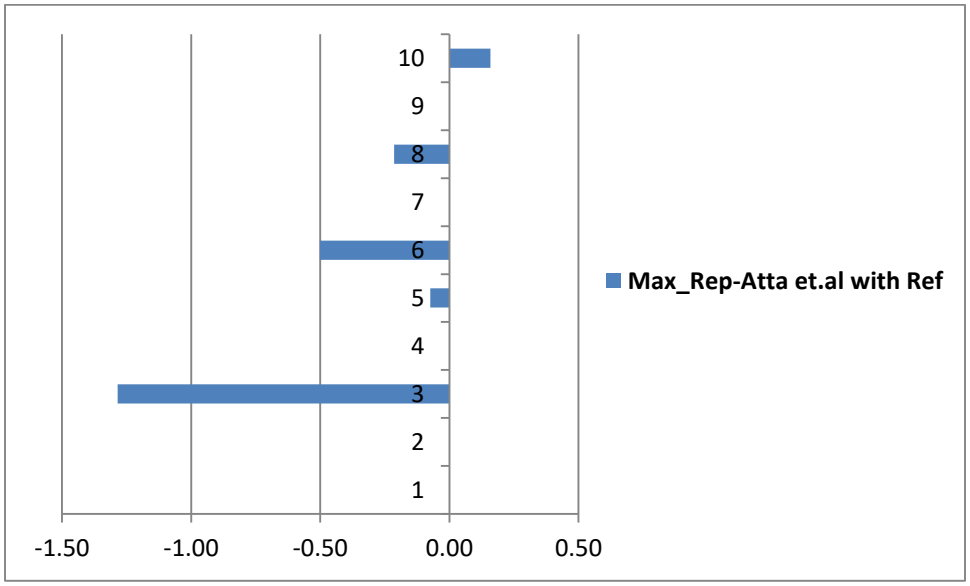


Figure [5]. Difference of Percentage Coverage in SJC708 (Max_Rep-Atta et al(2017) GA with Ref)

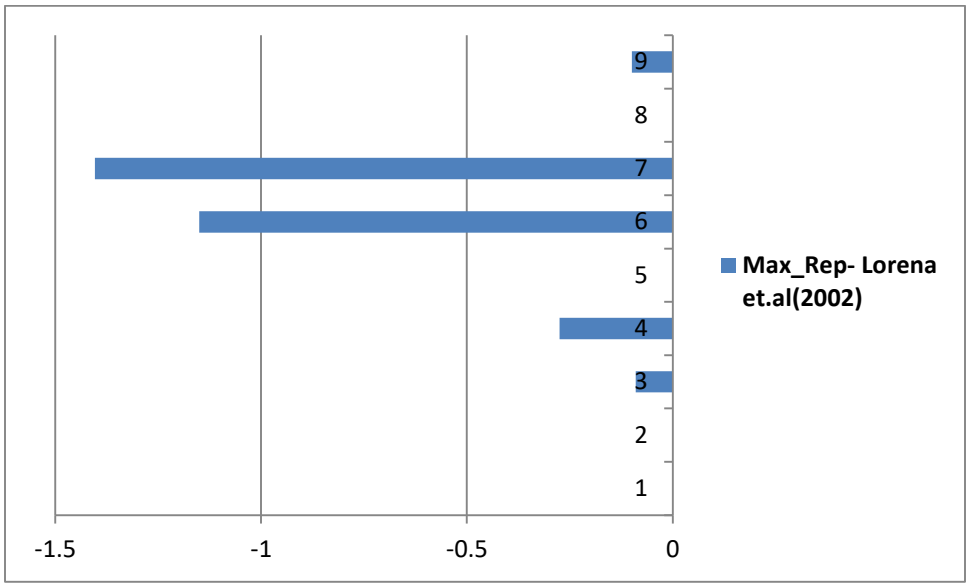


Figure [6]. Difference of Percentage Coverage in SJC818 (Max_Rep-Lorena et.al(2002))

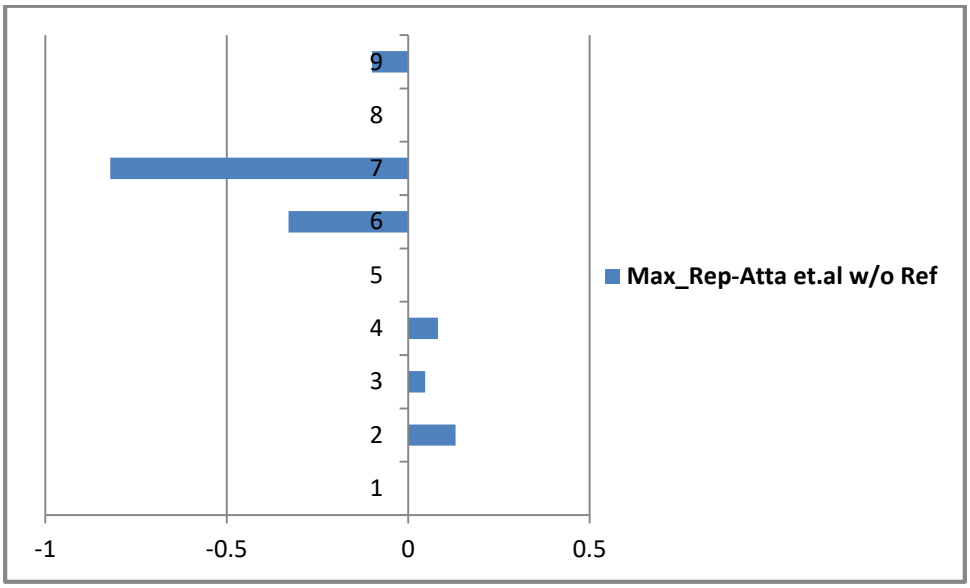


Figure [7]. Difference of Percentage Coverage in SJC818 (Max_Rep- Atta et.al(2017) GA w/o Ref)

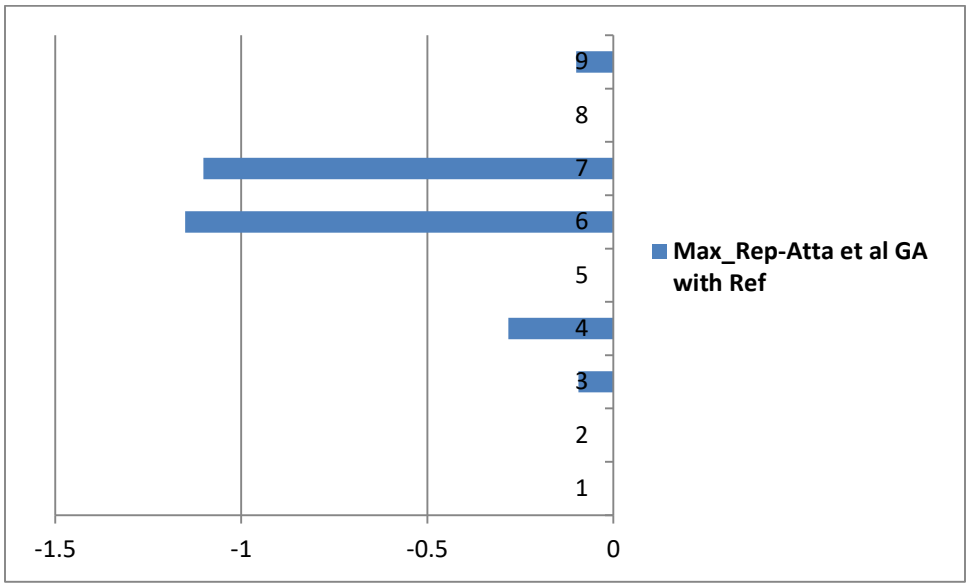


Figure [8]. Difference of Percentage Coverage in SJC818 (Max_Rep-Atta et al(2017) GA with Ref)

5. Conclusion

In this project, MCLP has been considered for real world and large-scale datasets and genetic algorithm with two different mating (2 parents-3 parents) and three different repairing functions (Rand_Rep-Gre_Rep and Max_Rep) is proposed.

In terms of the heuristic search capabilities;

- Both Rand_Rep and Max_Rep satisfy reachability.
- It is quite possible to have infeasible solutions after crossover therefore repairing functions Max_Rep and Rand_Rep are produced. Max_Rep has better exploitation capability, as it searches maximum marginal improvement. However, computing effort gets larger, hence some improvements are needed. Quality of the code also affects the computation time.
- Rand_Rep also finds good solutions quickly however randomness does not allow it to find optimal value mostly.
- Exploration capabilities are derived from the genetic algorithm itself. Searching on a population and introducing replacement probability (replacing parents and offspring) support the exploration. Early convergence is also intended to avoid using replacement probability and mutation operator.
- We investigate that using 3 parents 2 points do not improve the performance, and even sometimes worsens for the considered problem instances. 2 parents-1 point may improve the performance but 2 parents-2 points more often improve it.

As future directions; GA with Max_Rep can be extended. Currently, although it gives satisfactory results, it is computationally expensive. As a future direction we propose to improve the code quality and also improve the proposed maximum improvement repair operator.

References

Adenso-Diaz, B., & Rodriguez F. (1997). A simple search heuristic for the MCLP: application to the location of ambulance bases in a rural region. *Omega*, 25(2), 181-187. https://doi.org/10.1016/S0305-0483(96)00058-8

Atta, S., Sinha Mahapatra, P. R., & Mukhopadhyay, A. (2017). Solving maximal covering location problem using genetic algorithm with local refinement. *Soft Computing,22*(12), 3891-3906. doi:10.1007/s00500-017-2598-3

Aytug, H., & Saydam, C. (2002). Solving large-scale maximum expected covering location problems by genetic algorithms: A comparative study. *European Journal of Operational Research,141*(3), 480-494. doi:10.1016/s0377-2217(01)00260-0

Brandeau, M. L., & Chiu, S. S. (1989). An overview of representative problems in location research. *Management Science,35*(6), 645-674. doi:10.1287/mnsc.35.6.645

Church, R., & Revelle, C. (1974). The maximal covering location problem. *Papers of the Regional Science Association,32*(1), 101-118. doi:10.1007/bf01942293

Daskin, M.S. (1995). Network and discrete location: models, algorithms and applications. New York, NY:Wiley.

Daskin, M. S. (2008). What you should know about location modeling. *Naval Research Logistics,55*(4), 283-294. doi:10.1002/nav.20284

Farahani, R. Z., Hassani, A., Mousavi, S. M., & Baygi, M. B. (2014). A hybrid artificial bee colony for disruption in a hierarchical maximal covering location problem. *Computers & Industrial Engineering,75*, 129-141. doi:10.1016/j.cie.2014.06.012

Galvao R.D., & Revelle, C. (1996). A Lagrangean heuristic for the maximal covering location problem. *European Journal of Operational Research*, 88(1), 114-123.

Holland, H.J. (1975). Adaptation in Natural and Artificial Systems. *University of Michigan Press*

Jaramillo, J., Bhadury, J., & Batta, R. (2002). On The Use of Genetic Algorithms to Solve Location Problems. *Computers* &*Operations Research*, 29(6),761-779. doi:10.1016/S0305-0548(01)00021-1

Lee, J., & Lee, Y. (2010). Tabu based heuristics for the generalized hierarchical covering location problem. Computers &Industrial Engineering,

Lorena, L.A., & Pereira, M.A.(2002). A Lagrangean/surrogate heuristic for the maximal covering location problem using Hillman's edition. *International Journal of Industrial Engineering*, 9, 57-67.

ReVelle, C.S, & Eiselt, H.A. (2005). Location analysis: a synthesis and survey. European Journal of Operational Research, 165(1):1-19. https://doi.org/10.1016/j.ejor.2003.11.032

Senne, E.L.F., & Pereira, M.A., & Lorena, L.A.N. (2010). A decomposition heuristic for the maximal covering location problem. *Advances in Operations Research*, 2010. doi:10.1155/2010/120756

Shahanaghi K., & Ghezavati V.T. (2008). Efficient solution procedure to develop maximal covering location problem under uncertainty (using GA and simulation). *International Journal of Industrial Engineering & Production Research*, 19(4) 21-29.

Weaver J., & Church R. (1984) A comparison of direct and indirect primal heuristic/dual bounding solution procedures for the maximal covering location problem. Unpublished paper

Yoon, Y., & Kim, Y. (2013). An Efficient Genetic Algorithm for Maximum Coverage Deployment in Wireless Sensor Networks. *IEEE Transactions on Cybernetics,43*(5), 1473-1483. doi:10.1109/tcyb.2013.2250955

Zarandi, M. F., Davari, S., & Sisakht, S. H. (2011). The large scale maximal covering location problem. *Scientia Iranica, 18*(6), 1564-1570. doi:10.1016/j.scient.2011.11.008

APPENDIX A

Experimental results for different parameter settings and different repair operators

Rand_Rep																																	
Parameter Combination				Problem Category (p=5. S=800)																													
N. Ni	p_{rep}	Crossover operator		SJC324										SJC500										SJC818									
			Run #	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
			Solution	12072	12111	12104	11919	11930	12115	12104	11898	12146	12150	18071	17153	17357	17616	17773	18003	17739	17912	17676	18488	22425	21932	23159	21903	22473	22250	23131	21631	21172	22747
		1-point-2parents	CPU	8.054	7.881	8.054	7.956	7.973	7.972	7.928	7.938	7.967	8.057	19.617	19.103	19.479	19.677	19.445	19.338	19.441	19.321	19.49	19.493	52.639	52.471	52.471	52.595	52.45	52.411	52.486	52.635	52.387	52.524
			Solution	12,123	12086	12126	12083	11936	12085	11693	12135	12083	12060	17217	17261	17096	17678	17549	17331	17733	17530	15942	17251	22389	22937	22342	22933	22611	22102	22983	22430	23350	20992
	0.4	2 point- 2 parents	CPU	8.259	8.123	8.088	8.124	8.292	8.602	8.549	8.269	8.653	8.271	21.349	20.916	20.795	20.873	20.799	20.999	20.934	20.897	20.784	20.683	67.613	67.573	67.494	67.976	67.153	71.212	72.466	68.782	71.124	70.771
			Solution	12142	11913	12115	12024	12043	12143	12122	12143	12143	12121	17716	17739	17834	18029	17940	18198	18169	17761	17789	17944	22490	23448	22431	22271	21885	22439	21588	22029	22665	22244
		1-point-2parents	CPU	8.626	8.415	8.359	8.216	8.271	8.417	8.383	8.394	8.292	8.324	20.092	19.808	19.755	20.104	20.008	19.878	19.919	20.115	20.036	20.005	85.719	85.208	85.515	84.99	85.439	88.375	90.264	89.219	88.839	86.086
			Solution	12011	11956	12075	11875	12090	12115	11949	12044	12064	11955	17345	17562	17940	17498	17853	18102	17737	18082	17514	16980	22635	23505	22551	23103	22641	23182	23043	22303	23145	23160
50 (10)	0.7	2 point- 2 parents	CPU	8.409	8.251	8.126	8.076	8.096	8.248	8.245	8.247	8.249	8.248	20.838	20.933	20.819	20.89	20.895	20.757	20.482	21.096	20.876	20.851	73.159	68.702	70.319	73.84	69.112	72.473	70.316	73.371	70.19	73.465
			Solution	12123	12151	12142	12123	12151	12128	12123	12128	12123	12151	17890	18319	18296	17872	18644	18385	18051	18402	18543	18124	23410	22322	22856	23440	22494	21994	23165	22931	22337	22079
		1-point-2parents	CPU	24.561	24.552	24.65	24.646	24.672	24.86	24.474	24.594	24.522	24.611	57.088	56.672	57.182	58.289	56.986	57.284	57.011	57.144	57.254	57.455	112.296	112.567	112.505	112.094	112.126	112.059	112.395	112.122	112.336	112.658
			Solution	12128	12144	12142	12145	12086	12108	12133	12096	12031	12075	18001	17945	18132	18116	18512	18197	18116	18127	18152	18223	23470	23076	23778	23067	23644	23500	23785	23151	23871	22796
	0.4	2 point- 2 parents	CPU	23.916	23.844	24.114	24.009	23.892	24.162	24.096	23.904	23.916	23.911	61.502	62	61.072	61.08	61.096	61.479	60.652	61.101	61.2	61.212	120.248	125.748	125.798	124.326	124.789	145.721	148.159	135.412	147.895	133.654
			Solution	12143	12145	12123	12128	12144	12145	12151	12149	12143	12128	17917	17979	18457	18269	17909	17831	18234	18691	18313	18222	22613	22940	22635	22785	22414	22794	23046	22619	22722	22707
		1-point-2parents	CPU	24.472	24.27	24.243	24.325	23.95	23.984	24.341	24.506	24.077	24.403	57.345	57.07	56.952	56.141	56.637	57.497	56.491	56.282	56.004	56.745	112.04	112.213	112.799	112.049	112.679	109.86	108.366	109.669	109.887	109.277
			Solution	12145	12145	12123	12123	12116	12123	12117	12116	12106	12121	18614	18399	18607	18439	18552	18310	18210	18204	18460	18712	24190	23536	23723	23462	23680	24130	23591	23559	23753	23505
150 (30)	0.7	2 point- 2 parents	CPU	23.477	24.086	24.089	24.226	24.087	23.756	23.704	23.805	23.904	24.034	61.106	61.266	61.206	61.124	61.124	61.38	61.62	60.959	61.175	61.082	124.758	125.897	124.568	125.214	126.241	124.785	124.789	125.378	124.789	127.543
			Solution	12102	12104	11885	11966	12084	12111	12126	12126	12046	12123	17534	17662	17429	17639	17406	17537	18085	17427	17500	17767	22585	22233	21158	21395	22463	21804	22376	20761	20762	21838
	0.4	2 point- 3 parents	CPU	8.351	8.019	8.062	8.123	8.119	8.075	8.085	8.093	8.112	8.123	20.991	20.809	20.606	20.675	20.707	20.575	20.525	20.502	20.704	20.684	55.219	55.094	54.82	54.748	54.948	54.603	54.241	54.766	54.815	55.028
			Solution	12106	12145	12123	11905	12032	12116	12123	12142	12112	11919	17817	17893	18105	17732	17997	18251	17788	17800	17522	18474	22669	22116	22679	21834	22182	23851	23064	22201	21783	21347
50 (10)	0.7	2 point- 3 parents	CPU	8.069	8.038	8.121	8.081	8.068	8.151	8.056	8.126	8.121	8.045	20.761	20.579	20.596	20.466	20.614	20.555	20.554	20.459	20.724	20.708	54.733	55.171	54.906	54.242	54.798	54.99	54.732	54.723	55.199	54.79
			Solution	12116	12145	12118	12123	12123	12145	12145	12083	12084	12123	18310	18032	18004	17906	17862	18087	17702	17986	17798	17964	23028	23251	23213	23487	23262	23047	23157	22559	23311	23042
	0.4	2 point- 3 parents	CPU	24.161	24.074	24.101	24.087	23.823	24.Nis	23.994	24.188	24.126	24.047	61.833	60.85	61.305	61.332	60.742	61.194	61.378	61.271	60.743	60.974	118.452	118.451	117.915	117.665	117.696	117.796	117.878	117.591	117.788	118.355
			Solution	12145	12151	12123	12151	12145	12152	12123	12152	12145	11123	18533	18420	18536	18383	18244	18220	18194	18391	18320	18353	23249	24082	23614	23221	23697	23508	24049	23151	23723	22892
150(30)	0.7	2 point- 3 parents	CPU	24.177	23.774	24.794	24.757	24.032	24.046	26.153	25.904	25.916	25.664	61.348	61.446	60.875	60.681	60.8	60.941	60.871	61.897	61.082	61.081	118.138	117.245	116.457	117.548	117.589	118.225	116.679	117.245	118.345	117.639

Table [3]. Experimental results for Rand_Rep with respect to different parameter settings

Gre_Rep																																	
Parameter Combination				Problem Category (p=5. S=800)																													
N. Ni	p_{rep}	Crossover operator		SJC324										SJC500										SJC818									
			Run #	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
			Solution	12124	12046	12091	12089	12089	12114	12143	12140	12135	12134	18611	18211	18352	17982	18230	17916	17556	18818	18636	18205	20489	22084	21459	21258	21289	21690	20850	21829	21765	21699
		1-point-2parents	CPU	14820	14200	14824	14881	14587	14827	14765	14646	15130	14322	40982	40493	40833	40549	40858	40859	41263	40714	40710	40223	103584	105547	108447	106268	106758	106897	106624	106924	107182	108417
			Solution	12080	12115	12152	12143	12077	12100	12109	12147	12122	12019	18300	18452	18321	18231	17331	18200	17940	17456	17884	18320	20954	20843	20932	21842	23729	22032	20495	20563	20832	21895
	0.4	2 point- 2 parents	CPU	14900	14167	14534	14861	14201	14972	14732	15030	15130	14010	41672	41437	42010	41030	42860	42579	42854	42684	41094	43894	102932	103853	103485	104573	103978	104632	103786	102573	102148	102562
			Solution	12142	12116	12152	12104	12104	12142	12142	12143	12110	12152	18542	18546	18280	18582	18509	18579	18043	18423	18476	18025	21398	22302	22059	21946	22160	21330	21920	22148	20742	21469
		1-point-2parents	CPU	14826	15015	14528	14719	14545	14982	14427	14798	14478	14579	40248	39798	40205	40266	40692	40266	40281	40457	40304	39569	103623	106000	106054	106362	106610	103762	106008	106187	106239	106546
			Solution	12048	12100	12150	12118	12145	12138	12147	12136	12133	12145	18433	18429	18546	18594	18871	18554	18432	18370	18829	18530	24002	23020	21094	23921	22898	24201	24352	23972	23527	22829
50 (10)	0.7	2 point- 2 parents	CPU	15200	15132	14867	14358	14256	15101	14222	14567	14327	14223	40302	41900	42009	42032	42985	41875	43058	43218	43820	42691	118291	117321	124652	119880	117463	127932	130928	138452	13826	137261
			Solution	12143	12152	12152	12143	12144	12152	12143	12135	12152	12152	18232	18120	18765	18358	18343	18348	18342	18643	18549	18800	21076	22862	23136	23475	21936	21418	23492	24253	22612	22391
		1-point-2parents	CPU	41579	41396	41926	41773	41853	41472	41628	40928	41529	41235	103806	103131	104793	103611	104128	106864	105340	106546	10020	105713	188000	183252	185905	186431	184712	184336	180151	183805	186058	181453
			Solution	12145	12148	12152	12122	12137	12087	12001	12030	12126	12122	17289	18800	17638	18464	18548	17903	18048	18540	18503	18745	22785	22934	22484	22923	22545	22559	22479	22608	22405	22458
	0.4	2 point- 2 parents	CPU	40322	40212	40214	40334	41005	40542	41235	41256	40885	40665	104690	104526	105139	104876	104769	104717	104565	104707	105038	104646	189000	183720	178940	183529	179089	189435	184527	189034	189945	189034
			Solution	12152	12152	12152	12142	12152	12152	12152	12152	12152	12152	18232	18120	18765	18358	18343	18348	18342	18643	18549	18800	22711	22695	23291	22287	22731	21516	22474	22330	22732	22321
		1-point-2parents	CPU	40078	40533	40992	40511	40212	40487	40439	40848	40995	40585	104596	105492	103563	112001	109832	106589	103245	102345	110932	110302	186585	184297	179286	188144	189319	184379	182285	191668	186158	184528
			Solution	12030	12150	12114	12151	12127	12124	12136	12101	12122	12110	18706	18636	18518	18877	18400	18797	18670	18254	18404	18577	23532	24327	23897	24512	23970	24201	24523	24090	24310	23238
150 (30)	0.7	2 point- 2 parents	CPU	40.010	41.200	41.232	41.100	42.980	40.012	40.131	40.154	40.211	40.212	103.922	104.855	103.253	102.343	105.302	110324	102899	109.684	102.942	101.322	183.729	176.421	183.921	182.549	183.942	181.435	182.957	182.472	183.942	182.932
			Solution	12100	12124	12118	12144	12132	12128	12040	12143	12142	12134	17032	17345	18291	18245	17892	17543	17378	18802	17214	17291	21982	21937	21748	22722	22758	21932	22446	22468	21839	22032
	0.4	2 point- 3 parents	CPU	13.020	12.119	12.234	12.654	13.001	12.432	12.654	12.889	12.991	12.805	42428	42756	40201	41579	42648	41682	41974	40389	42680	42748	103232	104.572	102325	105.489	102748	103.434	103.272	106.492	112.345	106624
			Solution	12090	11988	12088	12056	12142	12132	12117	12122	12143	12150	17929	17892	18472	17352	16996	17893	18357	17932	17662	17820	20910	22932	20938	21847	21948	20473	22475	21792	21783	21843
50 (10)	0.7	2 point- 3 parents	CPU	12.900	12.410	12.114	12.342	12.543	13.435	12.889	13.010	12.534	12.987	40302	42571	41582	42674	42719	41720	42782	49230	42719	42810	108.372	104.232	102321	105.642	103.452	103.467	103.578	106.492	102.342	102.546
			Solution	12001	12101	12132	12102	11890	12124	12125	12134	12132	12131	18082	18346	17932	17892	18034	18302	18353	18394	18625	17283	21714	23346	22372	22950	23293	22324	22350	22461	23340	23668
	0.4	2 point- 3 parents	CPU	41.032	41.011	42.321	42.123	43.532	41.234	41.432	42.356	42.563	43.242	123826	123912	135271	120910	123572	125431	125411	102831	114512	113273	170.000	168.211	173812	184.903	189.323	174.932	173.892	173.913	189.302	172.811
			Solution	11998	11899	12030	12121	12134	12122	12132	12120	12120	12126	17438	17030	17890	17385	18390	17299	18493	18400	18204	17293	23282	23605	22820	23465	22665	22639	22796	23191	23049	23908
150(30)	0.7	2 point- 3 parents	CPU	42.124	42.567	43.565	42.657	42.889	42.987	43.976	42.655	42.587	43.179	12.801	102.012	101.282	103.922	101392	101.396	102.302	103.245	102.896	102.495	167.000	174.933	173.842	163.821	181.392	184.922	178.302	163.921	183.911	181.021

Table [4]. Experimental results for Gre_Rep with respect to different parameter settings

Max_Rep

Parameter Combination				Problem Category (p=5. S=800)																											
N. Ni	Prep	Crossover operator		SJC324										SJC500										SJC818							
			Run #	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
			Solution	12137	12123	12151	12138	12134	12137	12152	12151	12138	12142	18453	18532	18454	18365	18539	18457	18564	18345	18434	18589	23421	23213	24021	23531	23124	23465	23857	24014	24032	23862
		1-point-2parents	CPU	29.354	31.025	30.574	30.567	30.021	30.450	30.954	30.140	30.213	29.254	95.437	95.874	95.862	95.610	95.212	95.247	95.784	95.858	95.852	95.632	312.467	324.658	317.201	319.654	332.789	331.025	317.895	317.854	320.658	332.014
			Solution	12147	12152	12151	12145	12151	12147	12152	12151	12152	12147	18693	18717	18697	18643	18532	18642	18569	18563	18586	18743	23758	24023	23543	23241	23246	23545	23453	24035	23849	23853
	0.4	2 point- 2 parents	CPU	30.213	31.023	30.231	29.436	28.457	29.354	29.301	29.201	30.010	28.965	95.648	93.285	94.343	92.548	92.547	93.548	95.474	94.451	94.852	94.030	327.895	312.478	314.745	316.587	325.402	321.024	320.127	315.476	318.954	320.012
			Solution	12148	12151	12137	12148	12151	12152	12151	12148	12152	12151	18543	18467	18478	18589	18543	18553	18532	18678	18754	18643	23953	23859	23684	23954	24034	24129	23859	24103	23853	23953
		1-point-2parents	CPU	27.541	28.354	28.645	29.542	28.031	29.321	29.356	28.874	28.942	28.341	92.321	93.212	93.932	92.520	92.654	92.874	92.147	92.016	93.456	93.021	321.215	328.456	325.620	322.121	324.795	325.647	320.214	321.879	322.098	320.978
			Solution	12150	12152	12152	12152	12151	12152	12151	12152	12152	12152	18727	18834	18828	18839	18738	18838	18748	18685	18748	18832	24531	24331	24398	23210	24531	23970	24531	24341	24345	24331
50 (10)	0.7	2 point- 2 parents	CPU	28.654	27.569	28.342	27.786	28.867	27.541	28.303	28.959	29.001	27.958	93.631	91.230	93.098	91.202	91.245	92.174	92.410	93.287	91.205	93.546	325.315	325.478	322.124	320.154	326.547	325.645	324.213	320.124	320.546	322.036
			Solution	12148	12148	12137	12152	12148	12152	12151	12151	12151	12152	18649	18539	18623	18543	18345	18534	18431	18564	18642	18489	23214	24129	23859	23845	23768	23958	24193	23682	23768	24057
		1-point-2parents	CPU	80.475	80.654	81.247	82.012	81.247	81.414	81.361	82.014	80.141	80.954	177.897	177.845	179.532	179.523	175.421	175.698	175.203	174.520	175.698	177.547	612.423	601.652	620.134	621.464	598.755	598.523	602.134	612.047	612.410	611.470
			Solution	12147	12151	12152	12151	12151	12152	12152	12151	12147	12152	18487	18532	18478	18832	18742	18325	18646	18589	18632	18785	24247	24214	24045	24103	24213	23945	24054	23783	24034	24213
	0.4	2 point- 2 parents	CPU	82.032	82.466	82.120	82.020	82.741	82.957	82.254	82.654	82.301	81.925	175.489	179.855	178.952	178.500	178.451	179.542	177.410	174.178	176.548	175.603	620.147	620.147	622.147	612.585	613.025	620.397	603.214	600.132	602.134	614.565
			Solution	12151	12150	12148	12152	12152	12151	12152	12152	12151	12152	18674	18643	18584	18738	18753	18743	18675	18721	18810	18759	23950	24103	23869	24204	24201	24012	24103	23956	24394	24331
		1-point-2parents	CPU	81.242	81.201	82.045	81.474	81.242	80.325	80.965	80.147	80.545	81.046	177.845	178.542	174.560	174.474	175.562	175.420	174.302	175.289	174.982	174.564	598.741	589.524	598.562	599.621	603.245	600.124	598.785	596.214	589.531	603.254
			Solution	12152	12152	12152	12152	12152	12152	12152	12152	12152	12152	18839	18834	18821	18758	18768	18831	18824	18758	18832	18839	24451	24531	24485	24521	24518	24531	24521	24484	24531	24531
150 (30)	0.7	2 point- 2 parents	CPU	80.645	80.321	81.204	80.176	80.327	80.897	81.027	81.475	82.019	81.323	174.879	175.487	174.589	175.698	174.589	174.001	174.568	175.639	176.597	174.898	599.654	587.854	595.632	602.147	589.623	596.565	589.720	569.210	569.720	598.584
			Solution	12151	12138	12151	12152	12145	12136	12138	12151	12148	12147	18532	18453	18643	18452	18235	18533	18489	18476	18443	18576	23224	23145	23543	23642	23345	23426	23956	23657	23152	23758
	0.4	2 point- 3 parents	CPU	48.033	47.478	7.214	48.012	48.254	47.203	47.952	47.658	47.851	47.900	48.752	82.148	80.232	82.179	82.784	82.170	82.179	82.365	82.001	83.984	356.487	365.452	357.985	341.023	365.879	364.210	323.549	374.651	347.956	356.201
			Solution	12148	12152	12151	12152	12148	12147	12151	12152	12152	12151	18505	18705	18674	18574	18548	18754	18673	18694	18732	18432	24010	23989	23755	23898	24021	24101	23767	23546	23215	23421
50 (10)	0.7	2 point- 3 parents	CPU	47.846	47.952	48.214	48.230	47.560	48.521	47.787	47.521	47.698	47.897	82.014	81.023	80.220	80.978	80.714	81.024	80.457	81.212	81.023	81.974	325.487	325.648	325.687	326.587	321.021	312.025	328.972	329.654	336.542	332.101
			Solution	12151	12152	12138	12137	12148	12151	12137	12148	12151	12152	18348	18423	18431	18359	18594	18459	18293	18432	18563	18392	23948	23758	23565	23673	23907	23758	23869	24014	24019	24102
	0.4	2 point- 3 parents	CPU	88.054	87.985	88.952	89.547	8.925	90.201	90.234	89.524	89.754	89.659	192.014	192.354	194.517	195.421	193.247	197.564	192.012	190.236	1.920.124	193.201	656.479	654.751	657.854	635.478	658.978	645.214	648.954	647.412	650.321	632.014
			Solution	12151	12152	12151	12148	12151	12148	12137	12151	12152	12148	18751	18674	18458	18568	18674	18738	18594	18694	18732	18745	24203	24104	24235	24375	24334	24237	24217	24079	24365	24382
150(30)	0.7	2 point- 3 parents	CPU	90.215	92.021	91.325	91.478	91.515	90.219	90.901	91.213	92.001	91.278	195.246	192.545	192.031	194.579	195.475	190.354	189.204	189.745	195.422	194.232	652.458	624.796	624.795	652.489	623.146	632.580	630.021	642.130	625.462	635.746

Table [5]. Experimental results for Max_Rep with respect to different parameter settings

APPENDIX B

Summary of the experimental results for each problem set

SJC324	Worst Sol.	Best Sol.	Avg Sol.	Gap (%)
Rand_Rep	12123,00	12151,00	12139,90	0,10
Gre_Rep	12104,00	12152,00	12130,70	0,18
Max_Rep	12148,00	12152,00	12151,10	0,01

Table [6]. Summary of the results for SJC324

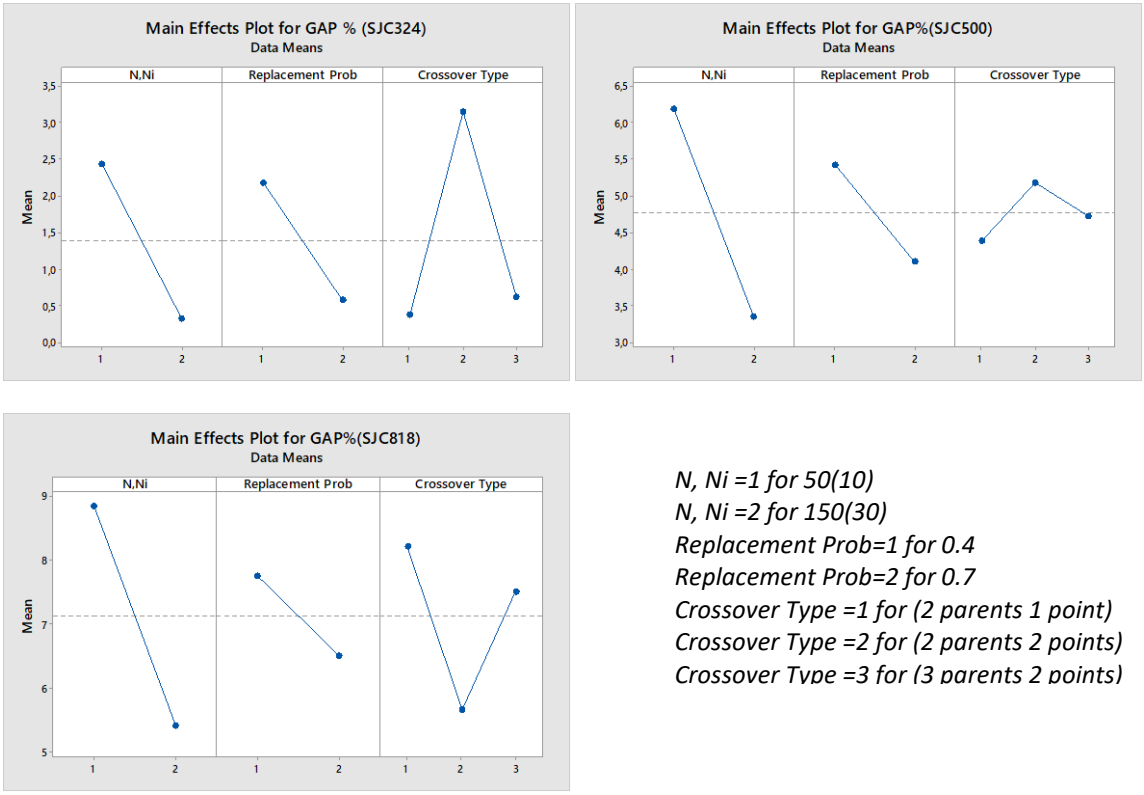
SJC500	Worst Sol.	Best Sol.	Avg Sol.	Gap (%)
Rand_Rep	18204,00	18802,00	18450,70	2,17
Gre_Rep	18254,00	18877,00	18583,90	1,46
Max_Rep	18758,00	18839,00	18810,40	0,26

Table [7]. Summary of the results for SJC500

SJC828	Worst Sol.	Best Sol.	Avg Sol.	Gap (%)
Rand_Rep	23462,00	24190,00	23712,90	3,33
Gre_Rep	23238,00	24523,00	24060,00	1,92
Max_Rep	24451,00	24531,00	24510,40	0,08

Table [8]. Summary of the results for SJC818

APPENDIX C



N, Ni =1 for 50(10)
N, Ni =2 for 150(30)
Replacement Prob=1 for 0.4
Replacement Prob=2 for 0.7
Crossover Type =1 for (2 parents 1 point)
Crossover Type =2 for (2 parents 2 points)
Crossover Type =3 for (3 parents 2 points)

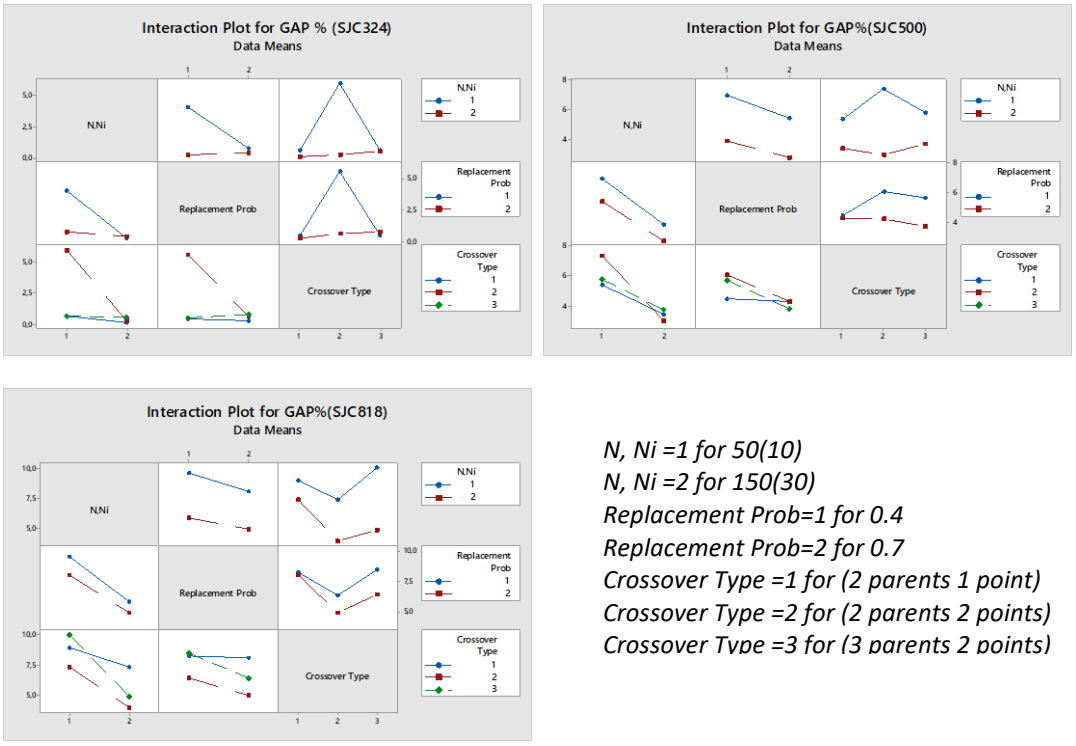
Figure [C1]. Main effects plots for SJC24, SJC500 and SJC800 data sets for Ran_Rep algorithm setting

Our findings indicate that when the repairing operator is Rand_Rep, for all the problems, setting N=150 N_i=30 and prep=0.7 provides better performances. For SJC324 and SJC500, selecting 2 parents 1 point crossover yields better performance compared to the other crossover operators. For SJC818, 2 parents 2 points crossover yields the best performance. However, we also need to consider the interactions among different parameter settings. Below Figure [C2] shows the interaction effects for the 3 problem sets for Rand_Rep. We summarize the findings as follows:

(i) *For SJC324, for N(N_i)=50(10), increasing prep from 0.4 to 0.7 improves the performance of the algorithm, but for N(N_i)=150(10), there is no significant difference between setting p_{rep} either to 0.4 or 0.7. When N(N_i)=50(10), choosing different crossover operators significantly affects the performance of the algorithm, while choosing either 2parents-1 point or 3 parents-2 points crossover improves the performance, choosing 2 parents 2 points crossover operator, results in worse performance. There is no significant difference between 2 parents-1 point and 3 parents-2 points crossover. When N(N_i)=150(10), crossover operators do not significantly affect the performance of the algorithm, but 2 parents-1 point crossover yields the best performance. We also observe that crossover operator performance and p_{rep} have interactions within each other. If 2 parents-2 points operator is selected, setting p_{rep}=0.7 improves the performance, for other crossover operators the probability has no significant effect.*

(ii) *For SJC500, for both N(N_i) settings, setting prep=0.7 improves the performance. If we set N(N_i)=50(10) choosing 2parents-2 points crossover operator yields the worst performance, while the other two operators yield better and similar results. If we set N(N_i)=150(30), 2parents-2 points crossover operator improves the performance of the algorithm. If we choose 2-parents-1 point crossover operator, then prep loses its importance, but for the other operators setting p_{rep}=0.7 improves the performance. For all the crossover operators setting N(N_i)=150(30) is better.*

(iii) *For SJC818, setting N(N_i)=150(30) improves the performance for all the other settings. For all N(N_i) settings 2 parents-2 points yields the best performance. For all the settings, setting p_{rep}=0.7 is the best choice.*



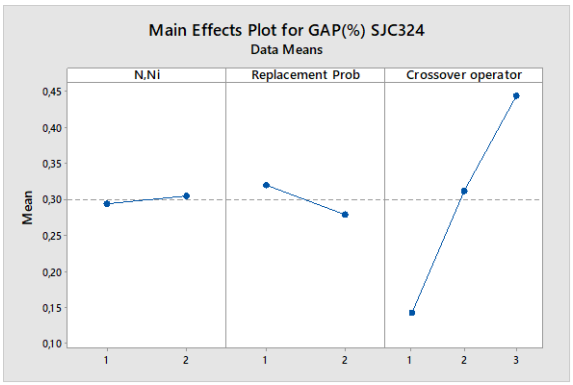
N, Ni =1 for 50(10)
N, Ni =2 for 150(30)
Replacement Prob=1 for 0.4
Replacement Prob=2 for 0.7
Crossover Type =1 for (2 parents 1 point)
Crossover Type =2 for (2 parents 2 points)
Crossover Type =3 for (3 parents 2 points)

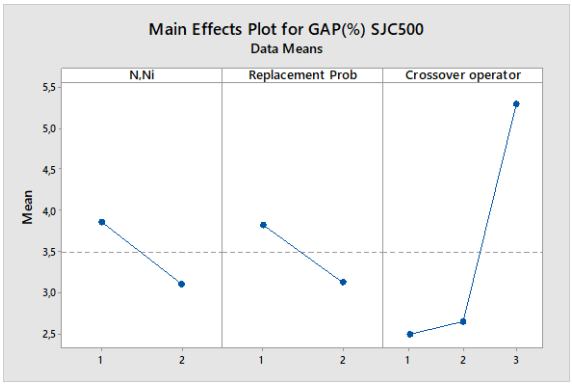
Figure [C2]. Interaction effects plots for SJC24, SJC500 and SJC800 data sets for Ran_Rep algorithm setting

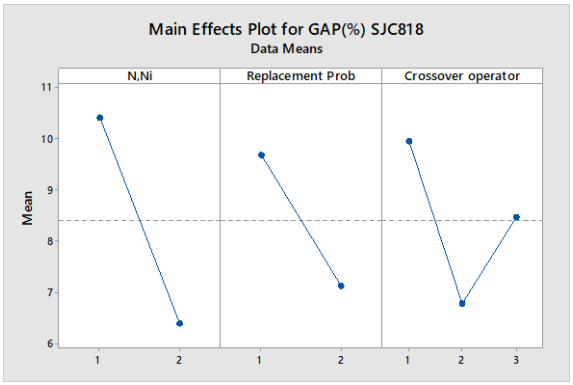
Findings indicate that for different problem sets, different crossover operators perform the best. For instance, while for SJC324 choosing 2 parents-1 point is the best choice, for the others choosing 2 parents-2 points is better. Therefore, for SJC324 the crossover operator as 2 parents-1 point, for the others we set 2 parents-2points. The other settings are set as N(N_i)=150(30), and p_{rep}=0.7.

We make the same analysis for Gre_Rep and Max_Rep operators, to choose the settings. When we observe the main effects plots for Gre_Rep setting, findings indicate that while for SJC324, setting N(N_i) to 50(10) or to 150(30) does not makes so much difference for the other problems, setting N(N_i)=150(30) yields better performances. For all the problems setting prep yields lower deviations from the optimal solution. For SJC324 and SJC500 choosing 2 parents-1 point crossover yields the best performance,

however, for SJC500 2 parents-2 point crossover may also be considered, since there is no much difference with 2 parents-1 points crossover operator. For SJC818, 2 parents-2 points is the best choice.

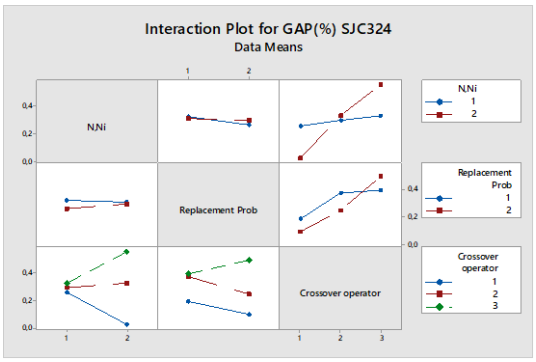


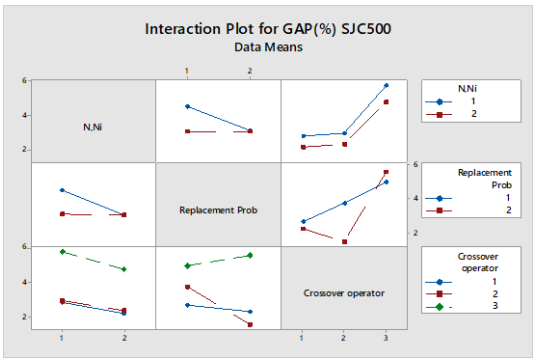


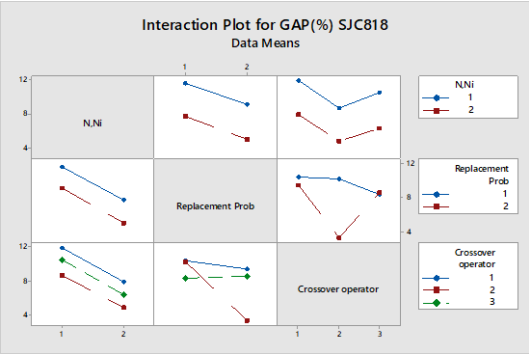


N, Ni =1 for 50(10)
N, Ni =2 for 150(30)
Replacement Prob=1 for 0.4
Replacement Prob=2 for 0.7
Crossover Type =1 for (2 parents 1 point)
Crossover Type =2 for (2 parents 2 points)
Crossover Type =3 for (3 parents 2 points)

Figure [C3]. Main effects plots for SJC24, SJC500 and SJC800 data sets for Gre_Rep algorithm setting







N, Ni =1 for 50(10)
N, Ni =2 for 150(30)
Replacement Prob=1 for 0.4
Replacement Prob=2 for 0.7
Crossover Type =1 for (2 parents 1 point)
Crossover Type =2 for (2 parents 2 points)
Crossover Type =3 for (3 parents 2 points)

Figure [C4]. Interaction effects plots for SJC24, SJC500 and SJC800 data sets for Gre_Rep algorithm setting

From the interactions plots we summarize the findings as follows:

(i) *For SJC324 for both N(N_i) settings, p_{rep} does not affect the performance of the algorithm significantly, but if set p_{rep}=0.7 and crossover operator as 3parents-2 points the performance of the algorithm worsens. Also setting N(N_i)=150(30) always worsens the performance of all the crossover operators. For all crossover operators and N(N_i) settings, p_{rep}=0.7 is better. For all N(N_i) and p_{rep} values 2 parents-1 point is the best choice.*

(ii) *For SJC500, if set N(N_i)=50(10) then setting p_{rep}=0.7 and crossover operator as 2 parents 1 point is the best choice. If we set N(N_i)=150(30), p_{rep} does not make so much difference, both values may be set. If we set $_{prep}$=0.4, then 2 parents-1 point and if we set p_{rep}=0.7 then 2 parents-2 points are the best choices for the crossover operator. Setting N(N_i)=150(30) improves the performance of all crossover operators.*

(iii) *For SJC818, for both N(N_i) settings, setting p_{rep}=0.7 improves the performance. For both N(N_i) settings, 2 parents-2 points crossover performs better compared to other crossover operators. For all p_{rep} values and crossover operators setting N(Ni)=150(30) is better.*

Again, our findings indicate that for different problem datasets different settings yields the best performance. Therefore, if we choose Gre_Rep for SJC324 we set N(N_i)=50(10), p_{rep}=0.7 and crossover operator as 2 parents-1 point. For the other two sets, we set N(N_i)=150(30), p_{rep}=0.7 and crossover operator as 2 parents-2 points.

The main plots for Max_Rep setting show that for all problem sets setting N(N_i)=150(30), p_{rep}=0.7 and choosing 2 parents-2points crossover yield the best performances for the algorithm. However, we need to also observe the interactions with each other. From Figure [C6], we summarize the findings as follows:

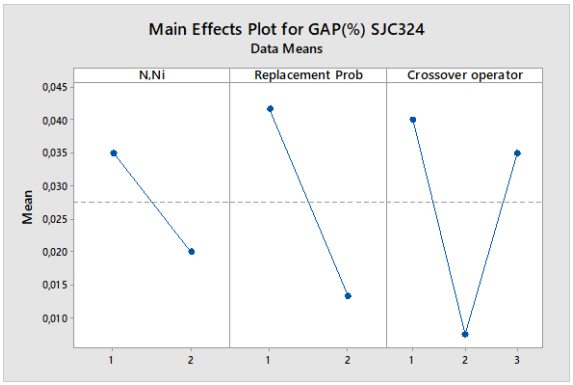
(i) *For SJC324, for both N(N_i) settings, setting p_{rep}=0.7 improves the performance. For both values of N(N_i) 2 parents-2points is the best choice, independently from p_{rep}. If we choose p_{rep}=0.4, then choosing N($_{Ni}$)=150(30) is better, and for p_{rep} the setting is not significantly affected. For the 2 parents crossovers again, choosing N(N_i)=150(30) is better, but for the 3 parents crossover, setting N(N_i)=50(10) is better.*

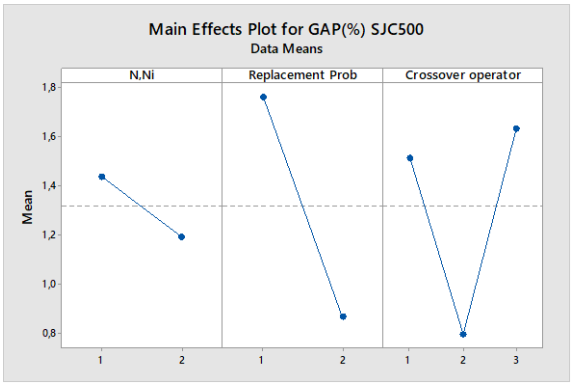
(ii) *For SJC500, for both N(N_i) settings, setting prep=0.7 improves the performance. For all the settings of N(N_i) and p_{rep} 2 parents-2 points is the best choice for the crossover operator. For*

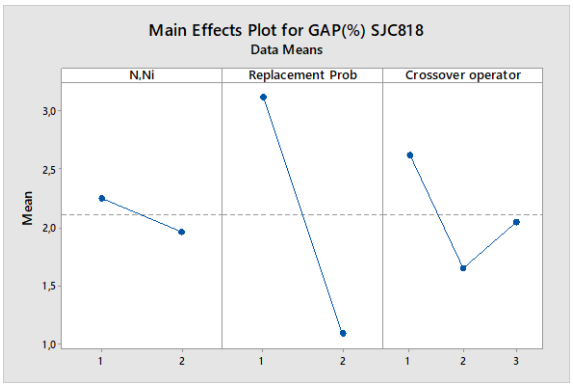
all prep and crossover operators, setting N(N_i)=150(30) improves the performance, although for 2 parents 2 points crossover operator is not affected significantly.

(iii) *For SJC818, for both N(N_i) settings, setting p_{rep}=0.7 improves the performance. If N(N_i)=50(10) then 3 parents-2 points is the best choice, and if N(N_i)=150(30) 2 parents-2 points is the best choice. For all p_{rep} values, again the 2 parents-2 points is the best choice. For all p_{rep} and crossover operators except 3 parents-2 points, setting N(N_i)=150(30) improves the performance of the algorithm.*

Our findings indicate that if we choose (N_i)=150(30), p_{rep}=0.7 and parents-2points crossover operator, our algorithm performs well for all the problem sets.

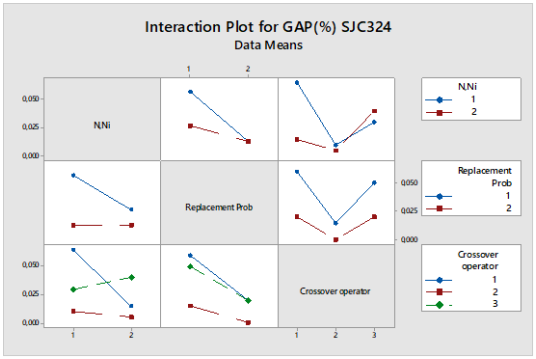


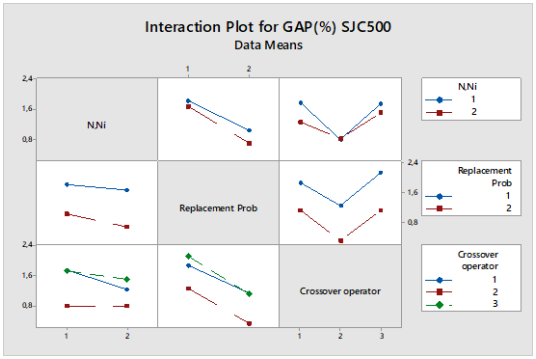


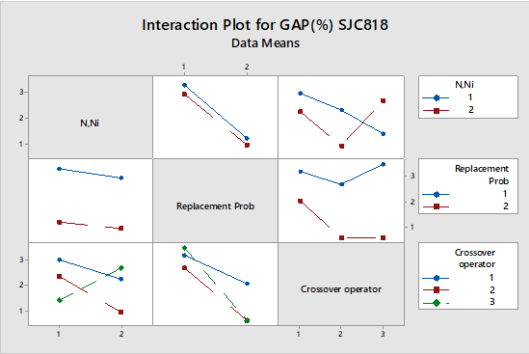


N, Ni =1 for 50(10)
N, Ni =2 for 150(30)
Replacement Prob=1 for 0.4
Replacement Prob=2 for 0.7
Crossover Type =1 for (2 parents 1 point)
Crossover Type =2 for (2 parents 2 points)
Crossover Type =3 for (3 parents 2 points)

Figure [C5]. Main effects plots for SJC24, SJC500 and SJC800 data sets for Max_Rep algorithm setting







N, Ni =1 for 50(10)
N, Ni =2 for 150(30)
Replacement Prob=1 for 0.4
Replacement Prob=2 for 0.7
Crossover Type =1 for (2 parents 1 point)
Crossover Type =2 for (2 parents 2 points)
Crossover Type =3 for (3 parents 2 points)

Figure [C6]. Interaction effects plots for SJC24, SJC500 and SJC800 data sets for Max_Rep algorithm setting

APPENDIX D

ANOVA Results for SJC324 for different algorithm settings

```
Null hypothesis         All means are equal
Alternative hypothesis  At least one mean is different
Significance level      α = 0,05

Equal variances were assumed for the analysis.


Factor Information

Factor        Levels  Values
Alg. Setting       3  1; 2; 3


Analysis of Variance

Source        DF    Adj SS  Adj MS  F-Value  P-Value
Alg. Setting   2    124691   62346     0,16    0,850
Error         27  10282864  380847
Total         29  10407555


Model Summary

      S   R-sq  R-sq(adj)  R-sq(pred)
617,128  1,20%      0,00%       0,00%


Means

Alg.
Setting   N   Mean  StDev      95% CI
1        10  11527    655  (11127; 11927)
2        10  11556    710  (11155; 11956)
3        10  11676    459  (11275; 12076)

Pooled StDev = 617,128
```

Tukey Pairwise Comparisons

```
Grouping Information Using the Tukey Method and 95% Confidence

Alg.
Setting   N   Mean  Grouping
3        10  11676  A
2        10  11556  A
1        10  11527  A

Means that do not share a letter are significantly different.
```

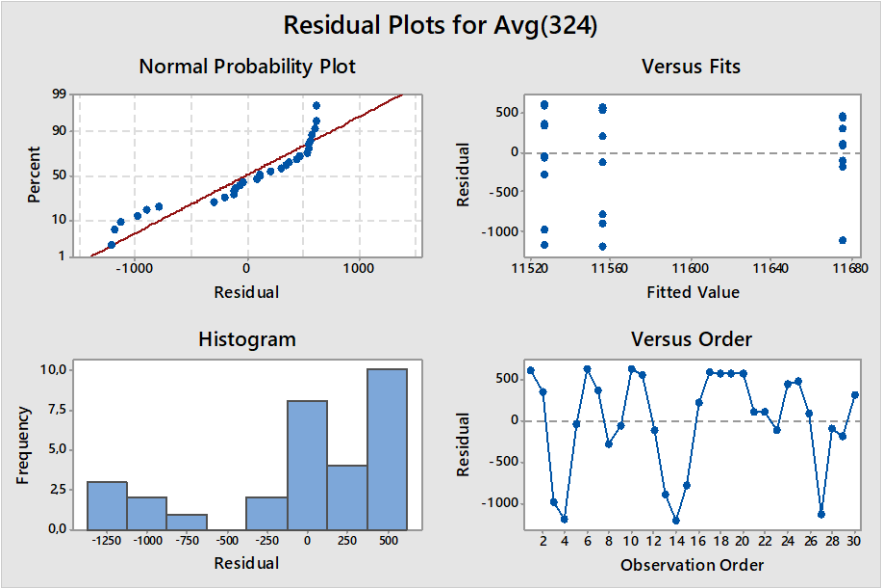


Figure [D1]. Residual Plots for SPJ324

There is no reason to suspect any violation of normality, independence and constant variance assumption.

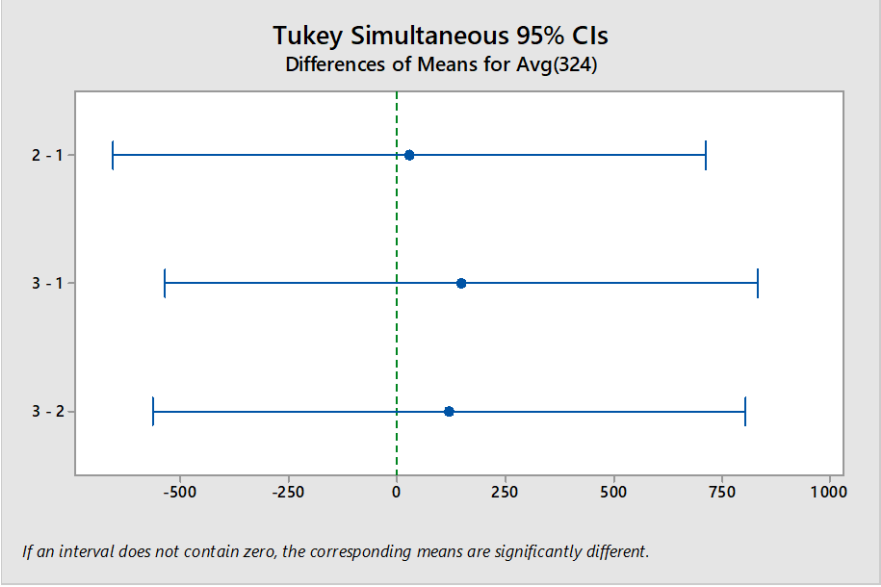


Figure [D2]. Differences of means for SJC324.

As from the figure all the intervals contain 0 which indicates that the performance of algorithm for different settings is not significantly different.

ANOVA Results for SJC500 for different algorithm settings

```
Method

Null hypothesis         All means are equal
Alternative hypothesis  At least one mean is different
Significance level      α = 0,05

Equal variances were assumed for the analysis.


Factor Information

Factor        Levels  Values
Alg. Setting       3  1; 2; 3


Analysis of Variance

Source        DF    Adj SS   Adj MS  F-Value  P-Value
Alg. Setting   2   5494669  2747334     3,76    0,036
Error         27  19703718   729767
Total         29  25198387


Model Summary

      S    R-sq  R-sq(adj)  R-sq(pred)
854,264  21,81%     16,01%       3,46%


Means

Alg.
Setting   N   Mean  StDev      95% CI
1        10  16957    831  (16403; 17512)
2        10  16820    969  (16266; 17374)
3        10  17789    749  (17234; 18343)

Pooled StDev = 854,264
```

Tukey Pairwise Comparisons

```
Grouping Information Using the Tukey Method and 95% Confidence

Alg.
Setting   N   Mean  Grouping
3        10  17789  A
1        10  16957  A B
2        10  16820    B

Means that do not share a letter are significantly different.
```

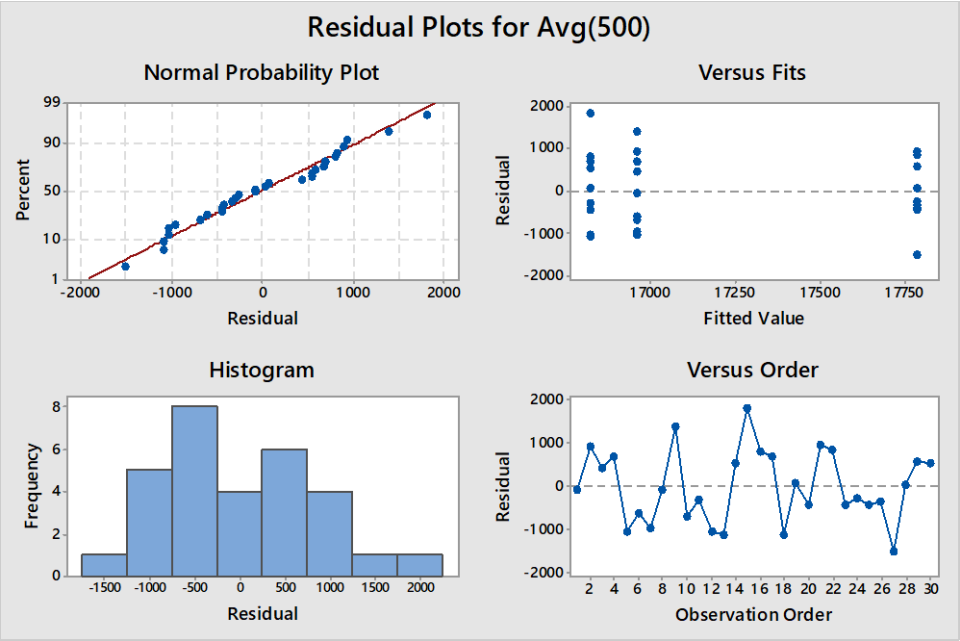


Figure [D3]. Residual plots for SJC500

There is no reason to suspect any violation of normality, independence and constant variance assumption.

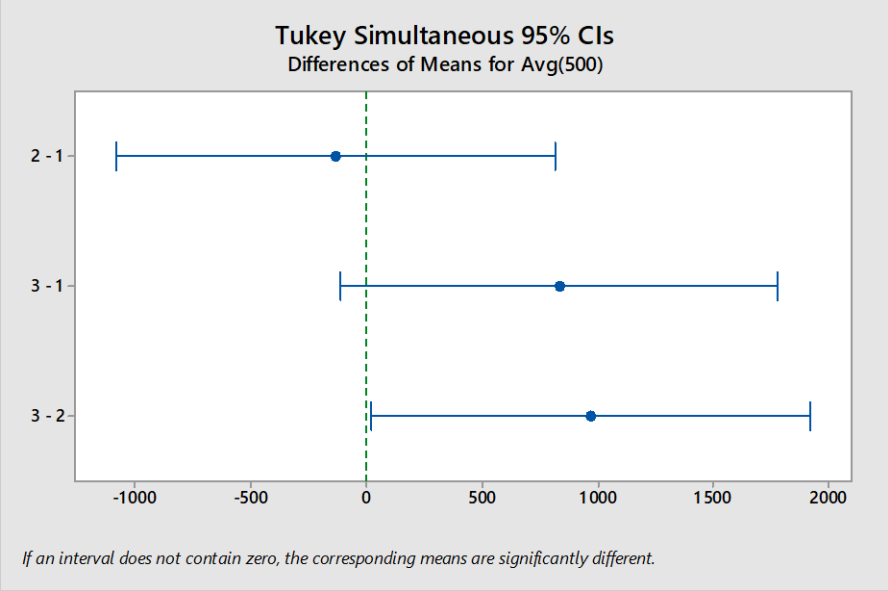


Figure [D4]. Differences of means for SJC500

Figure [D4] shows that Gre_Rep (denoted by 2 in the figure)and and Max_Rep (denoted by 3 in the figure) are significantly different, since the differences of the mean does not include 0.

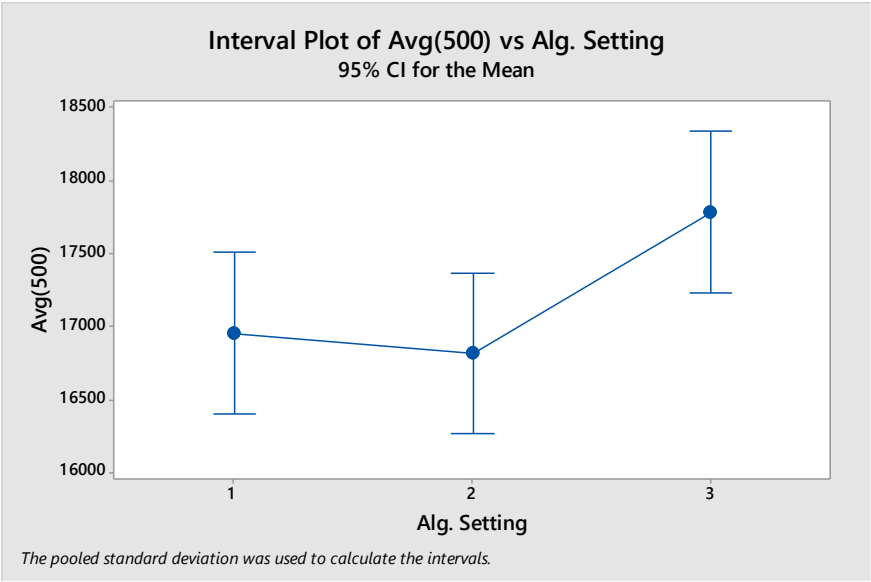


Figure [D5]. Interval plots for Rand_Rep, Gre_Rep and Max_Rep for SJC500

From Figure [D5] we observe that the Max_Rep (denoted by 2 in the figure) yields the highest objective function, and so give the best performance. Tukey's test claims that Max_Rep is significantly different from Gre_Rep. Hence, we may claim that the algorithm with Max_Rep is the best setting for SJC500.

ANOVA Results for SJC818 for different algorithm settings

Method

Null hypothesis All means are equal
Alternative hypothesis At least one mean is different
Significance level α = 0,05

Equal variances were assumed for the analysis.

Factor Information

Factor	Levels	Values
Alg. Setting	3	1; 2; 3

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Alg. Setting	2	17266340	8633170	4,32	0,024
Error	27	53966519	1998760		
Total	29	71232859			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
1413,78	24,24%	18,63%	6,47%

25

Means

Alg. Setting	N	Mean	StDev	95% CI
1	10	21242	888	(20325; 22160)
2	10	20948	1460	(20031; 21866)
3	10	22684	1754	(21767; 23602)

Pooled StDev = 1413,78

Tukey Pairwise Comparisons

Grouping Information Using the Tukey Method and 95% Confidence

Alg. Setting	N	Mean	Grouping
3	10	22684	A
1	10	21242	A B
2	10	20948	B

Means that do not share a letter are significantly different.

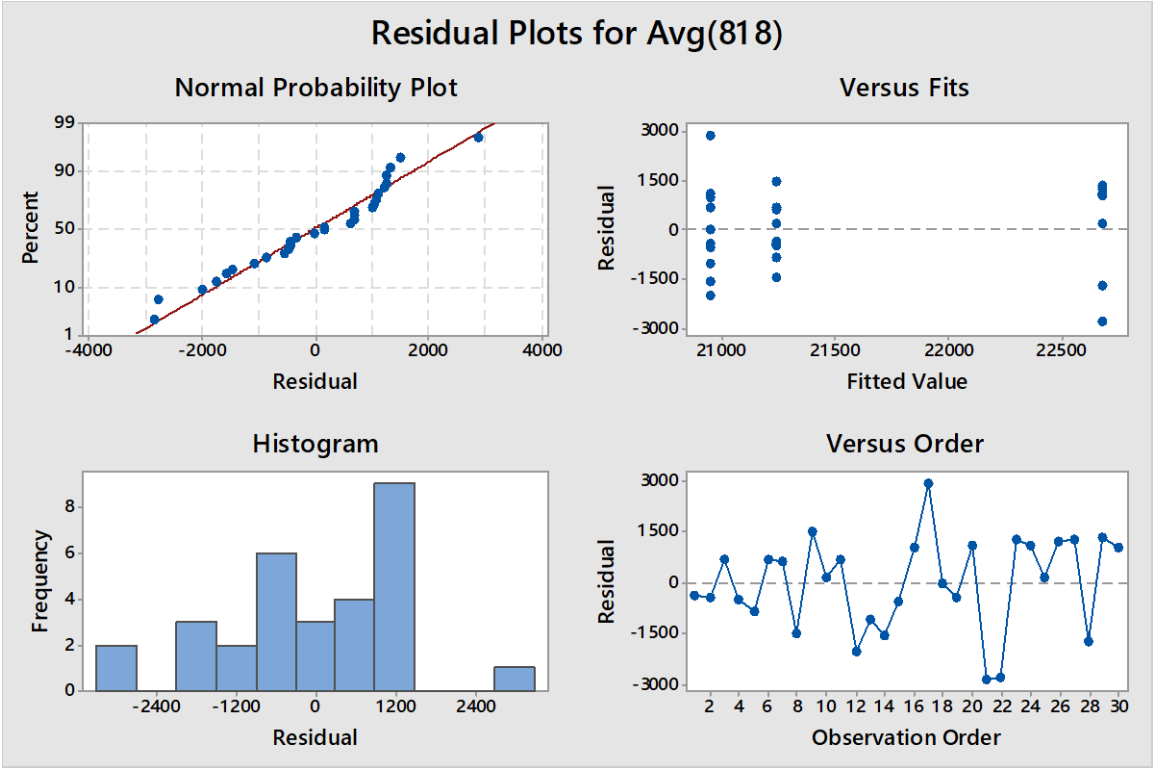


Figure [D6]. Residual plots for SJC818

There is no reason to suspect any violation of 25 normality, independence and constant variance assumption.

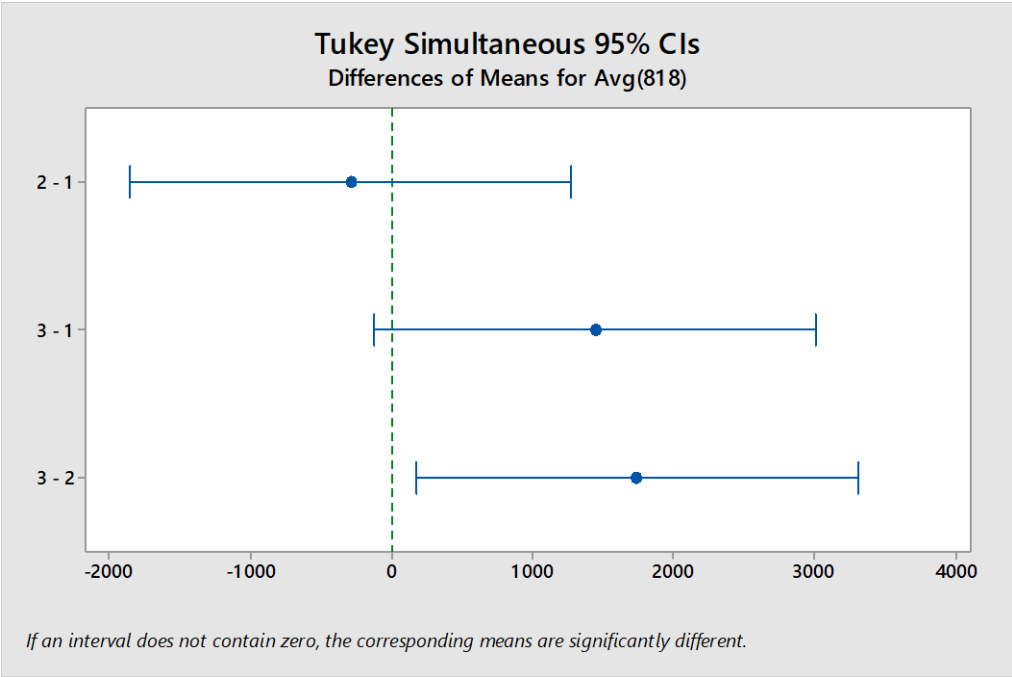


Figure [D7]. Difference of means for SJC818

We observe that MAX_rep and Gre_Rep are statistically different from each other since the interval does not include 0.

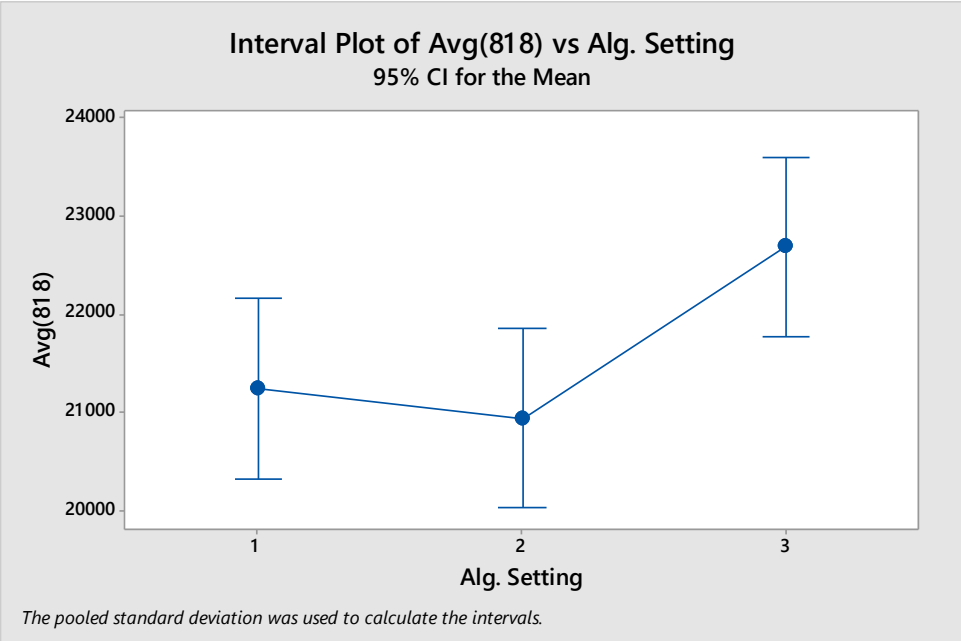


Figure [D8]. Interval plots for Rand_Rep, Gre_Rep and Max_Rep for SJC818

Figure [D8] shows that Max_Rep yields the highest average which indicates it performs the best compared to the other settings.