Ahmed, *Cogent Engineering* (2018), 5: 1423743
**https://doi.org/10.1080/23311916.2018.1423743**

cogent engineering

COMPUTER SCIENCE | RESEARCH ARTICLE

# A hybrid algorithm combining lexisearch and genetic algorithms for the quadratic assignment problem

Zakir Hussain Ahmed[1]*

Received: 29 September 2017
Accepted: 31 December 2017
First Published: 05 January 2018

*Corresponding author: Zakir Hussain Ahmed, Department of Computer Science, Al Imam Mohammad Ibn Saud Islamic University (IMSIU), P.O. Box No. 5701, Riyadh 11432, Kingdom of Saudi Arabia
E-mails: zaahmed@imamu.edu.sa, zhahmed@gmail.com

Reviewing editor:
Alex Alexandridis, Technological Educational Institute of Athens, Greece

Additional information is available at the end of the article

**Abstract:** Lexisearch and genetic algorithms are two different types of methods for solving combinatorial optimization problems. Lexisearch algorithm gives us exact optimal solution, whereas, genetic algorithms give heuristic solution to a problem. In this paper, a hybrid algorithm (LSGA) that combines lexisearch and genetic algorithms is developed to obtain heuristic solution to the quadratic assignment problem. The proposed algorithm uses lexisearch algorithm to generate initial population, self-adaptively three crossover operators, and randomly one of four mutation operators, restricted combined mutation operator as local search, and multi-parent sequential constructive crossover as immigration method. The self-adaptive crossover operator that consists of one-point crossover, swap path crossover and sequential constructive crossover can produce better solutions. Also, the random selection of a mutation operator effectively prevents LSGA from being stuck in local optimal zone. Further, the immigration method with combined mutation effectively generated very good chromosomes, which promotes the convergence rate and accuracy of the solution. Experimental results on four categories of benchmark QAPLIB instances show the effectiveness of the LSGA. Out of 35 instances 18 instances have been solved optimally, and for the remaining instances, solutions are very close to the optima. Finally, a comparative study has been carried out between LSGA and

Zakir Hussain Ahmed

## ABOUT THE AUTHOR

Zakir Hussain Ahmed is an Associate Professor in the Department of Computer Science et al. Imam Mohammad Ibn Saud Islamic University, Saudi Arabia. He obtained MSc in Mathematics (Gold Medalist), MTech in Information Technology and PhD in Mathematical Sciences from Tezpur University (Central), Assam, India. He served in various institutions in India. His research interests include artificial intelligence, discrete optimization, digital image processing and pattern recognition. He has publications in the fields of artificial intelligence, discrete optimization and image processing.

## PUBLIC INTEREST STATEMENT

Given a set of facilities and locations along with the flows between facilities and the distances between locations, the objective of the "Quadratic Assignment Problem" is to assign each facility to a location in such a way as to minimize the total assignment cost. The problem has a several applications such as the location of interdependent plants or facilities, the layout of interacting departments in an office building, the location of medical facilities in a hospital, the backboard wiring problem in the design of computer and other electronic equipment, and so on. As the problem is very difficult to solve, several methods have been proposed to find optimal solution. However, finding optimal solution within less computational efforts is still challenging one. This paper proposes a hybrid algorithm by combining two types of methods that encourages to make use of advantages of both methods. The experimental results also acknowledge its effectiveness.

unified particle swarm optimization (UPSO) for the same instances. In terms of solution quality, LSGA outperformed UPSO for all category of instances. Also, in terms of computational time, except for seven instances, LSGA outperformed UPSO.

**Subjects: Evolutionary Computing; Computer Science (General); Operations Research**

**Keywords: quadratic assignment problem; hybrid algorithm; lexisearch algorithm; genetic algorithm; multi-parent crossover; sequential constructive crossover; adaptive mutation**

## 1. Introduction

The quadratic assignment problem (QAP) is one of the most difficult combinatorial optimization problems. It was first introduced by Koopmans and Beckmann (1957) that can be defined as follow: There are a set of n facilities and a set of n locations. Let $f_{ij}$ be a flow of information between facilities $i$ and $j$, and $d_{kl}$ be the distance between locations k and l. Also, let $a = \{a(1), a(2), ..., a(n)\}$ be an assignment, where $a(i)$ represents the location of the facility $i$. The problem is to assign to each location exactly one facility to minimize the total cost (objective function)

$$Z_a = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{a(i)a(j)} \tag{1}$$

The QAP is proved to be NP—hard problem (Sahni & Gonzalez, 1976). It has a wide variety of applications such as the location of interdependent plants or facilities, the layout of interacting departments in an office building, the location of medical facilities in a hospital, the backboard wiring problem in the design of computer and other electronic equipment, some production scheduling problems with interactive cost and so on.

Based on different viewpoints or purposes, the methods for solving the QAP as well as any other combinatorial optimization problem can be classified mainly in two classes– exact and heuristic. In solving the QAP, many both exact and heuristic procedures have been reported in the literature over the past fifty years. The methods that provide the exact optimal solution to the problem are called exact methods. Though quite a few efficient exact algorithms have been developed, still only few instances of size $n \geq 30$ from QAPLIB have been solved optimally. Thus, for larger problem instances, heuristics have been developed. Heuristics are the techniques, which seek good solutions (i.e. near optimal solutions) at a reasonable computational cost without being able to guarantee either optimality or even in many cases to state how close to the optimal solution.

Branch and bound, dynamic programming, Lagrangian relaxation based methods, linear and integer programming based methods, lexisearch algorithms, etc., are well-known exact methods for solving the QAP. The most recent heuristic methods that can be adapted to a wide range of combinatorial optimization problems are called metaheuristics. Examples of such methods are Neural Networks (Uwate, Nishio, Ueta, Kawabe, & Ikeguchi, 2004), tabu search (TS) (Czapiński, 2013), genetic algorithm (Ahmed, 2010a), simulated annealing (Misevicius, 2003), Colony Optimization (ACO) (Hong, 2013), particle swarm optimization (PSO) (Hafiz & Abdennour, 2016), etc.

Looking at the advantages and disadvantages of exact and metaheuristic algorithms, it appears to be natural to combine ideas and methods from both classes. However, such combined approaches became more popular only over the last years, and there have been many literatures that developed combined algorithms. In some literatures, exact and heuristic algorithms are executed sequentially or in parallel. In other literatures, one technique is a subordinate component of another (Puchinger & Raidl, 2005).

Nagar, Heragu, and Haddock (1995) described a combination of branch and bound and a GA for a two-machine flow-shop scheduling problem in which solutions are represented as permutations of jobs. Before running the GA, branch and bound is implemented down to a predetermined depth $k$.

Suitable bounds are calculated and stored at each node of the branch and bound tree. Throughout the implementation of GA, the partial solutions up to position *k* are mapped onto the node of the tree.

An approach for finding near-optimal solutions to the traveling salesman problem (TSP) was proposed by Applegate, Bixby, Chvátal, and Cook (1998), who derived a set of diverse solutions by multiple runs of an iterated local search algorithm. The edge-sets of these solutions are merged, and the TSP is finally solved to optimality on this strongly restricted graph. As reported, their solutions are found to be superior to the best solution of the iterated local search.

An effective local and variable neighborhood search heuristic for the asymmetric TSP was presented by Burke, Cowling, and Keuthen (2001) that embedded an exact algorithm in the local search part, called HyperOpt, to exhaustively search relatively large promising regions of the solution space. As reported, this method overcome local optima and create high quality tours.

Interior point methods and metaheuristics were combined by Plateau, Tachat, and Tolla (2002) for solving the multiconstrained knapsack problem, where an interior point method is the first part of their algorithm. Computational results on some benchmark instances shows that the presented combination is a promising research direction.

Raidl and Feltl (2004) solved the generalized assignment problem using a hybrid GA. First, the LP-relaxation of the problem is solved using CPLEX2 and its solution is used by a randomized rounding procedure to create meaningful initial population for the GA. As reported, this type of LP-based initialization is very effective.

Nwana, Darby-Dowman, and Mitra (2005) developed a parallel high-level teamwork hybrid that consists in combining a branch and bound algorithm with simulated annealing. The simulated annealing algorithm sends improved upper bounds to the exact algorithm.

Franceschi, Fischetti, and Toth (2006) developed a refinement heuristic based on integer linear programming for the distance-constrained capacitated vehicle routing problem. As reported, computational results on a large set of instances show effectiveness of the method.

Mezmaz, Melab, and Talbi (2007) presented a parallel hybrid exact multi-objective algorithm that combines genetic algorithm and memetic algorithm with branch and bound algorithm. The algorithm is applied and validated on a bi-objective flow-shop scheduling problem.

Archetti, Speranza, and Savelsbergh (2008) presented a solution approach for the split delivery vehicle routing problem that integrates heuristic search with optimization. The first part of the algorithm is to use the information provided by a tabu search heuristic to identify parts of the solution space that contain good solutions. Then explore this part of the solution space by suitable integer programming model. The reported computational results were found to be very encouraging.

Hewitt, Nemhauser, and Savelsbergh (2009) developed an approach that combines mathematical programming algorithms with heuristic search technique for the fixed charge network flow problem. The algorithm first solves integer programs resulting from the arc-based formulation of the problem to obtain its lower bounds. The algorithm also incorporates randomization to diversify the search. Computational experiments show the effectiveness of the proposed algorithm.

A hybrid approach to solve the capacitated vehicle routing problem is developed by Guimarans, Herrero, Riera, Juan, and Ramos (2011) that combines a probabilistic algorithm with constraint programming and Lagrangian relaxation. The algorithm first generates a starting solution, which is then improved using a local search method that combines Lagrangian relaxation and constraint

programming to verify the feasibility of new proposed solutions quickly. The efficiency of algorithm is measured by testing some benchmark instances.

Holborn, Thompson, and Lewis (2012) developed a combined method that combines tabu search and branch-and-bound algorithm for the vehicle routing problem with pickups, deliveries and time windows. As reported, the approach is fast method to construct individuals and achieves promising results.

Long and Wu (2014) developed a hybrid method that combines genetic algorithm and the Hooke-Jeeves method to solve a class of constrained global optimization problems. More precisely, the Hooke-Jeeves method was embedded into genetic algorithm as an acceleration operator during the iterations. As reported, the numerical experiments show that proposed method achieves better performances than genetic algorithm, Hooke-Jeeves method and some available global optimization solvers.

Recently, lexisearch algorithm (Ahmed, 2013a) is found to be one of the best exact algorithms, and genetic algorithm (Ahmed, 2014) is found to be one of the best heuristic algorithm for the QAP. Hence, in this paper, a hybrid algorithm that combines lexisearch algorithm with genetic algorithm is developed to find heuristic solution to the QAP. More specifically, a simple lexisearch algorithm is used to generate initial population for the genetic algorithm. Then a genetic algorithm using self-adaptively three crossover operators, randomly selected one of four mutation operators, a restricted combined mutation operator as local search, and multi-parent sequential constructive crossover as an immigration method, is developed. Experimental results on QAPLIB instances show the effectiveness of the proposed hybrid algorithm. Finally, a comparative study has been carried out between the proposed algorithm and unified particle swarm optimization (UPSO) of Hafiz and Abdennour (2016) for some QAPLIB instances, and found that the proposed algorithm is better.

The remainder of this paper is organized as follows: Section 2 reports the proposed hybrid algorithm, abbreviated as LSGA, to find heuristic solution to the problem. Section 3 describes computational results for the proposed hybrid algorithm. Finally, section 4 presents comments and concluding remarks.

## 2. Proposed hybrid algorithm (LSGA) for the QAP

LSGA is a hybrid algorithm that combines lexisearch and genetic algorithms for the QAP. First an initial population is generated using lexisearch algorithm (Ahmed, 2013a) by considering only five iterations for a chromosome. A total of "*n*" chromosomes are generated using lexisearch algorithm by starting each location as first gene. Then the stochastic remainder selection method (Deb, 1995) is applied to the current population to create a mating pool. Next, three crossover operators—One-point crossover (OPX) (Lim, Yuan, & Omatu, 2000), swap path crossover (SPX) (Ahuja, Orlin, & Tiwari, 2000) and sequential constructive crossover (SCX) (Ahmed, 2014), have been applied self-adaptively to a pair of chromosomes, where, if one of them produces better offspring than both parents, then skip the remaining crossover operator(s), and go for the mutation. In mutation, one of the four mutation operators—Adaptive, Exchange, 3-Exchange and Gene-Exchange (Ahmed, 2016), is chosen randomly for a selected chromosome. Further, if the present best solution is better than the previous best solution, then the combined mutation operator (Ahmed, 2010b) is applied as a local search method for further improvement. Also, to diversify the population, some better chromosomes are injected using an immigration method based on multi-parent crossover method (Ahmed, 2015a), which are further improved by a local search algorithm. During the implementation of genetic algorithm, the immigration method generates some very good chromosomes, which considerably promote the convergence rate and accuracy of the solution.

### *2.1. Generating initial population*

As starting with a good initial population leads faster convergence of GA, several researchers use optimal solution for the relaxation of the problem, which is then repaired using problem specific

procedure. Plateau et al. (2002) combined interior point methods and metaheuristics for solving the multidimensional knapsack problem. They first performed an interior point method with early termination, and then by rounding and applying several different ascent heuristics, a population of different feasible solutions is generated, which is used as initial population. Raidl and Feltl (2004) described a hybrid GA for the generalized assignment problem, in which the LP relaxation of the problem is solved, and its solution is exploited by a randomized rounding procedure, which is then repaired, if infeasible, and improved to create an initial population of promising integral solutions.

In the proposed LSGA, a simple lexisearch algorithm without lower bound calculation is used to generate initial population. In lexisearch algorithm, the set of all possible solutions to a problem is arranged in a hierarchy, such that each incomplete word represents the block of words with this incomplete word as the leader. For this problem, first an "alphabet table" based on the distance matrix, *D*, is constructed. Each location is considered as a letter in an alphabet and each assignment as a word with this alphabet. The entire set of words in this dictionary (namely, the set of solutions) is partitioned into blocks. Value of a word is calculated based on the objective function and compared with the "best solution value" found so far. If the word is not better than the "best solution value" found so far, then jump out to the next super-block, otherwise, enter into the sub block by concatenating the present leader with appropriate letter and set a bound for the new sub-block (Ahmed, 2013a). The simple lexisearch algorithm for generating initial population can be stated as follows.

Step 0: Let $F$ and $D$ be the flow and distance matrices respectively. Form the "alphabet table" based on $D$. Repeat steps 1 to 4 for $n$ number of times ($n$ is the population size also).

Step 1: Set $Z_a = M$ (as large a possible); first "location $\delta$" varies sequentially from 1 to $n$; $k = 2$; $Z_0 = 0$.

Step 2: Consider the first "unassigned and unchecked" location (say, $\delta$) in the $k$th row of the alphabet table. If there is no such location in the row, go to step 4.
Suppose $(\alpha_1, \alpha_2, \alpha_3, ..., \alpha_{k-1})$ be an incomplete assignment, then the cost (value) of assigning "location $\delta$" for the "facility $k$" is calculated as follows:

$$c_\delta = \sum_{i=1}^{k-1} (f_{ik} d_{\alpha_i \delta} + f_{ki} d_{\delta \alpha_i}) \tag{2}$$

Value of present incomplete assignment, $Z_k = c_\delta + Z_{k-1}$. If $(Z_k \geq Z_a)$ then drop the "location $\delta$", and go to step 4; otherwise, go to step 3.

Step 3: If $(k = n)$ then replace $Z_a = Z_k$; otherwise, set $k = k + 1$ and go to step 1. If five complete assignments are generated, then stop, otherwise, go to step 4.

Step 4: Set $k = k - 1$ and reject all the subsequent assignments. If $(k < 2)$ then stop; otherwise, go to step 2.

### *2.2. Three crossover operators*

Cossover is one of the most important operators in GA search that selects a pair of chromosomes (called parents) and exchanges information between them. Following three crossover operators have been considered in the proposed LSGA.

#### *2.2.1. Sequential constructive crossover operator*

The sequential constructive crossover (SCX) was initially developed for the TSP (Ahmed, 2010a), which was then applied successfully to the QAP (Ahmed, 2014), can be stated as follows.

Step 0: Start from the location (suppose, $p$) of the first facility of any randomly chosen parent.

Step 1: Sequentially search both parent chromosomes and consider the first "legitimate location" (the location that is not yet assigned) appeared after "location $p$" in each parent. If any parent has no any "legitimate location", after "location p", search sequentially from the beginning of the parent and consider the first "legitimate location", and go to Step 2.

Step 2: Suppose "location $\alpha$" and "location $\beta$" are found in 1st and 2nd parent respectively, then for selecting the next location go to Step 3.

Step 3: Compute the cost of one incomplete offspring chromosome by incorporating "location $\alpha$" as the next location (suppose, $c_\alpha$). Similarly, compute the cost of other incomplete offspring by incorporating "location $\beta$" as the next location (suppose, $c_\beta$). Then go to Step 4. Suppose ($\alpha_1$, $\alpha_2$, $\alpha_3$, ..., $\alpha_{k-1}$) be a partially constructed offspring and "location $\delta$" is selected for concatenation, then the cost (value) of assigning this location for the facility $k$ is calculated using Equation (2).

Step 4: If $c_\alpha \leq c_\beta$, then select "location $\alpha$", otherwise, "location $\beta$" as the next location to be assigned for the next facility and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, go to Step 5; otherwise, rename the present location as "location $p$" and go to Step 1.

Step 5: Evaluate the first parent and the offspring chromosomes. If value of the offspring is less than the value of the parent, replace the first parent by the offspring, otherwise skip it.

*2.2.2. One-point crossover operator*

Lim et al. (2000) developed the one-point crossover (OPX), which can be stated ads follows.

Step 0: Let $P_1$ and $P_2$ be two parent chromosomes. Choose a crossing point (site) randomly between 1 and $n - 1$, suppose $x$.

Step 1: Set offspring, $O$, as the first $x$ locations of the first parent.

Step 2: For the remaining locations, sequentially search the second parent from the beginning and consider first "legitimate" locations.

*2.2.3. Swap path crossover operator*

The swap path crossover (SPX) was proposed by Ahuja et al. (2000), which is described as follows.

Step 0: Let $P_1$and $P_2$ be two parent chromosomes. Start at the first gene.

Step 1: Parents are examined from left to right until all the genes have been considered.

Step 2: If the alleles at the position being looked at are the same, then move to the next position; otherwise, swap two alleles in $P_1$ or in $P_2$ so that the alleles at the current position become alike.

Step 3: If the alleles at the position being looked at are the same, then move to the next position; otherwise, swap two alleles in $P_1$ or in $P_2$ so that the alleles at the current position become alike.

Step 4: Evaluate the chromosomes after swapping and select the best one for the further offspring construction

Step 5: Start at the next position, and repeat steps 1 through step 5 until a complete offspring is generated.

***2.3. Four mutation operators***

Mutation is a process to diversify population by modifying information in the genes. It randomly changes the information of some selected gene(s). Ahmed (2016) reported an extensive study on eight different mutation operators of which four mutations—adaptive, exchange, three-exchange and gene-exchange- are found to be competing, which will be used for the proposed LSGA.

*2.3.1. Adaptive mutation*

The adaptive mutation, an intelligent exchange mutation, was proposed by Ahmed (2015b), which is described as follows.

Step 0: Consider all chromosomes in the current population.

Step 1: Create a one-dimensional array of size *n* (size of the problem), suppose, A, by storing a location (gene) that appears minimum number of times in the current position of all chromosomes.

Step 2: If mutation is allowed, select randomly two genes such that they are not same in the corresponding positions of the array, A, and swap them.

#### *2.3.2. Exchange mutation*

Exchange mutation selects two positions randomly and swaps the genes on these positions.

#### *2.3.3. Three-exchange mutation*

Three-exchange mutation selects three different positions at random, say, $r_1$, $r_2$ and $r_3$, and swaps the genes on these positions as follows: $P(r_1) \leftrightarrow P(r_2)$ and then $P(r_2) \leftrightarrow P(r_3)$.

#### *2.3.4. Gene-exchange mutation*

Gene-exchange mutation selects two genes randomly and swaps them.

### *2.4. Combined mutation operator*

The combined mutation operator was initially developed for the bottleneck TSP (Ahmed, 2010b), which was then applied successfully to other problems also (Ahmed, 2013b). Suppose ($\beta_1$, $\beta_2$, $\beta_3$, ... , $\beta_n$) is an assignment, then the combined mutation operator for the QAP can be stated as follows:

Step 0: For *i*: = 1 to *n* − 1 do the following steps.

Step 1: For *j*: = *i* + 1 to n do the following steps.

Step 2: If inserting location $\beta_i$ after location $\beta_j$ reduces the cost of the assignment, then insert the location $\beta_i$ after the location $\beta_j$. In any case go to step 3.

Step 3: If inverting substring between the locations $\beta_i$ and $\beta_j$ reduces the present assignment cost, then invert the substring. In any case go to step 4.

Step 4: If swapping the locations $\beta_i$ and $\beta_j$ reduces the present assignment cost, then swap them.

### *2.5. Immigration*

To improve GAs, the population must be diversified, and hence, immigration method is used. In this work, multi-parent sequential constructive crossover (Ahmed, 2015a) is used to create a chromosome, which is further improved using combined mutation and then injected to the population after some generations. The multi-parent sequential constructive crossover algorithm is as follows.

Step 0: First, fix the number of parents, suppose *m*. Start from the location (suppose, *p*) of the first facility of any randomly chosen parent, and go to Step 1.

Step 1: Sequentially search all the m parent chromosomes and consider the first "legitimate location" (the location that is not yet assigned) appeared after "location *p*" in each parent. If any parent has no "legitimate location", after "location *p*", search sequentially from the beginning of the parent and consider the first "legitimate location", and go to Step 2.

Step 2: Suppose locations {$\beta_1$, $\beta_2$, ..., $\beta_m$} are found in 1st, 2nd, ..., *m*th parent respectively, then for selecting the next location go to Step 3.

Step 3: Compute the cost of one incomplete offspring chromosome by incorporating "location $\beta_1$" as the next location (suppose, $c_{\beta 1}$). Similarly, compute the cost of other incomplete offsprings by incorporating remaining locations and suppose cost of these incomplete offsprings are $c_{\beta 2}$, $c_{\beta 3}$, ..., $c_{\beta m}$ with respect to the locations in order. Then go to Step 4. Suppose ($\alpha_1$, $\alpha_2$, $\alpha_3$, ..., $\alpha_{k-1}$) be a partially constructed offspring and "location $\delta$" is selected for concatenation, then the cost (value) of assigning this location for the "facility *k*" is calculated using Equation (2).

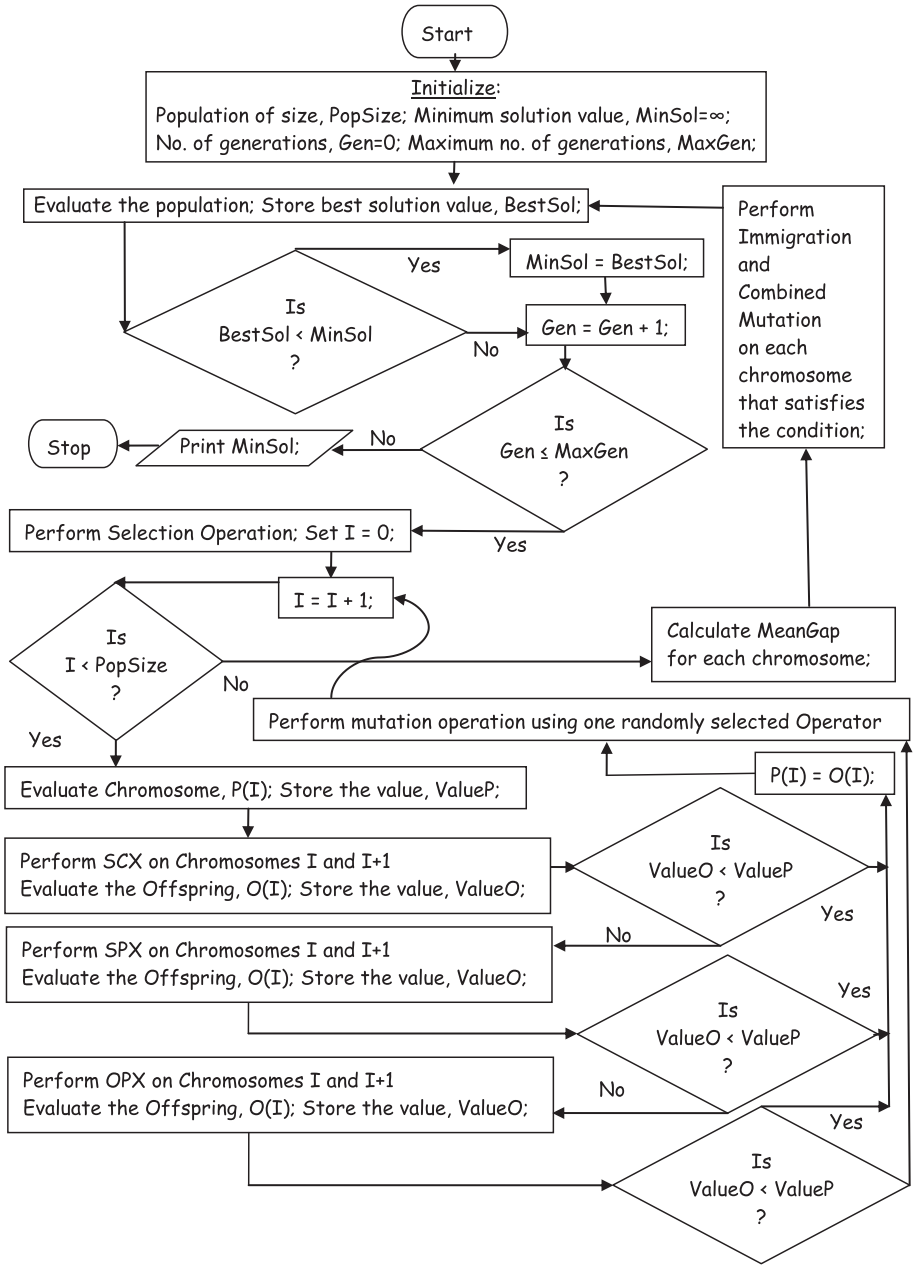
Step 4: Select "location $\beta$", if $c_{\beta} = \min\{c_{\beta 1}, c_{\beta 2}, \ldots, c_{\beta m}\}$, as the next location to be assigned for next facility and concatenate it to the partially constructed offspring chromosome. If the offspring is a complete chromosome, go to Step 5; otherwise, rename the present "location $\beta$" as "location $p$" and go to Step 1.

Step 5: Evaluate the first parent and the offspring. If value of the offspring is less than the value of the parent, replace the parent the offspring, otherwise skip the offspring.

Following condition is applied for applying immigration. Let $z_i$ be the objective function value of the $i$th chromosome in the population of size $P_s$, and $\hat{z}$ is the average objective function value of the current population. Then calculate percentage of MeanGap as follows.

$$\text{MeanGap} = 100(z_i - \hat{z})/\hat{z}, \quad \text{for } i = 1, 2, \ldots., P_s \tag{3}$$

**Figure 1. Flow-chart of LSGA.**



For any chromosome *i*, if (MeanGap > 1.00), then apply the immigration and combined mutation methods to the chromosome. The proposed LSGA may be summarized using flow-chart as shown in Figure 1.

## 3. Experimental results

The proposed hybrid algorithm (LSGA) has been encoded in Visual C++ on a PC with Intel(R) Core(TM) i7-3770 CPU @ 3.40 GHz and 8.00 GB RAM under MS Windows 7, and tested with 35 QAPLIB instances (Burkard, Karisch, & Rendl, 1997). The instances are selected from different subcategories, i.e., Category I: randomly generated (1–10), Category II: based on the grid distances (11–19), Category III: real life like (20–27) and Category IV: real life problems (29–35). Also, for each subcategory, both low and high sized instances are selected, to see the effect of the size on the algorithm performance.

After trial and error method, parameters are set as follows: a maximum of 50 generations as termination condition, 100 as population size, crossover probability 1.00 (100%) and mutation probability 1.00 (100%). To see the consistency of the results, LSGA is run ten times; and percentage of excess of average solution value (Excess (%)) is reported. The percentage of excess of average solution value (ASV) from the best-known solution value (BKV) reported in QAPLIB, is given by the formula

$$\text{Excess}\,(\%) = 100(\text{ASV} - \text{BKV})/\text{BKV} \tag{4}$$

Table 1 shows the selected benchmark instances, their categories, BKVs so far and the results. Table reports best percentage excess (BPE), average percentage excess (APE), worst percentage excess (WPE) and standard deviation of the percentage excess (StdDev); and average complete computational time (CT) (in minutes) and the average time when the final best solution is seen for the first time (FT) on 10 runs. The table also reports the time ratio of FT to CT to see how much time is spent for confirming a solution. Further, if the BKV is achieved more than once during the 10 runs, it is indicated by the value in parentheses along with the BPE. Table 1 also shows the average value category-wise (partial average) and for all problem instances (grand average) of BPE, WPE, APE, StdDev, FT, CT and Ratio.

In terms of computational time, it is seen that, on average (partial average of ratio), LSGA sees final best solution for the first time within 59, 74, 78 and 50% of complete computational times for the respective categories. This shows that LSGA finds optimal solution, on average, in the middle of the generations for the randomly generated instances (Category I) and real-life instances (Category IV). It seems that the first and fourth category problems can be solved quickly, whereas, second and third category problems take long time to solve. On the grand average, it is observed that LSGA finds optimal solution in the third quarters of generations.

In terms of solution quality, for the problem instances in the respective categories LSGA performed with 0.80, 0.16, 0.09 and 0.00% average BPE respectively. It seems that the fourth category problems are the easiest to solve by the algorithm, whereas, first the category problems are difficult to solve. The last row of Table 1 shows the grand average of BPE for all the 35 problem instances. For the all instances, LSGA performed with 0.29, 0.94 and 0.55% grand average of BPE, WPE and APE respectively. These percentage of excess are very close to each other, which can be confirmed by looking at grand average of StdDev. Out of 35 instances 18 instances have been solved optimally, at least once in ten runs. For the remaining instances, best solutions are very close to the optimal solutions, with maximum percentage of excess is 1.62%.

To have some depth search behaviour of LSGA on combinatoriaoptimization problems, a fitness landscape analysis (FLA) is performed (Jones & Forrest, 1995; Merz & Freisleben, 2000; Vanneschi, Tomassini, Clergue, & Collard, 2003). Usually, FLA has been used to guess the problem hardness,

**Table 1. Results obtained by LSGA for 35 QAPLIB problem instances over 10 runs**

| | Instance | Size | BKV | Excess (%) | | | | Computational time | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | BPE | WPE | APE | StdDev | FT | CT | Ratio |
| Category I | had20 | 20 | 6922 | 0 (10) | 0.00 | 0.00 | 0.00 | 0.03 | 0.13 | 0.23 |
| | lipa40b | 40 | 476,581 | 0 (10) | 0.00 | 0.00 | 0.00 | 0.25 | 1.50 | 0.17 |
| | rou20 | 20 | 725,522 | 0 (6) | 0.49 | 0.09 | 0.13 | 0.06 | 0.15 | 0.40 |
| | tai20a | 20 | 703,482 | 0.30 | 1.18 | 0.67 | 0.35 | 0.04 | 0.11 | 0.36 |
| | tai30a | 30 | 1,818,146 | 0.48 | 1.76 | 1.23 | 0.50 | 0.33 | 0.56 | 0.59 |
| | tai40a | 40 | 3,139,370 | 1.06 | 2.13 | 1.52 | 0.41 | 0.98 | 1.30 | 0.75 |
| | tai50a | 50 | 4,938,796 | 1.62 | 2.53 | 2.07 | 0.36 | 2.41 | 3.09 | 0.78 |
| | tai60a | 60 | 7,205,962 | 1.49 | 2.44 | 2.00 | 0.39 | 5.15 | 6.34 | 0.81 |
| | tai80a | 80 | 13,499,184 | 1.53 | 2.32 | 2.01 | 0.30 | 18.13 | 20.50 | 0.88 |
| | tai100a | 100 | 21,052,466 | 1.53 | 2.14 | 1.85 | 0.23 | 43.36 | 47.51 | 0.91 |
| | **Partial average** | | | **0.80** | **1.50** | **1.14** | **0.27** | **7.07** | **8.12** | **0.59** |
| Category II | Nug30 | 30 | 6124 | 0 (5) | 0.20 | 0.06 | 0.07 | 0.36 | 0.54 | 0.67 |
| | sko42 | 42 | 15,812 | 0 (4) | 0.42 | 0.19 | 0.17 | 1.30 | 1.83 | 0.71 |
| | sko49 | 49 | 23,386 | 0.14 | 0.54 | 0.25 | 0.15 | 2.62 | 3.33 | 0.79 |
| | sko81 | 81 | 90,998 | 0.10 | 0.53 | 0.30 | 0.17 | 4.50 | 7.94 | 0.57 |
| | sko90 | 90 | 115,534 | 0.33 | 0.54 | 0.42 | 0.08 | 32.87 | 36.56 | 0.90 |
| | sko100a | 100 | 152,002 | 0.26 | 0.54 | 0.39 | 0.11 | 52.67 | 56.95 | 0.92 |
| | sko100d | 100 | 149,576 | 0.32 | 0.63 | 0.44 | 0.11 | 50.03 | 56.18 | 0.89 |
| | tho150 | 150 | 8,133,398 | 0.23 | 0.72 | 0.49 | 0.21 | 133.44 | 284.74 | 0.47 |
| | wil50 | 50 | 48,816 | 0.02 | 0.17 | 0.07 | 0.05 | 2.52 | 3.42 | 0.74 |
| | **Partial average** | | | **0.16** | **0.48** | **0.29** | **0.12** | **31.15** | **50.17** | **0.74** |
| Category III | tai20b | 20 | 122,455,319 | 0 (10) | 0.00 | 0.00 | 0.00 | 0.03 | 0.10 | 0.30 |
| | tai30b | 30 | 637,117,113 | 0 (10) | 0.00 | 0.00 | 0.00 | 0.45 | 0.55 | 0.82 |
| | tai40b | 40 | 637,250,948 | 0 (9) | 0.01 | 0.00 | 0.00 | 1.05 | 1.49 | 0.70 |
| | tai50b | 50 | 458,821,517 | 0 (3) | 0.47 | 0.14 | 0.18 | 2.89 | 3.45 | 0.84 |
| | tai60b | 60 | 608,215,054 | 0 (3) | 0.12 | 0.04 | 0.04 | 5.95 | 6.85 | 0.87 |
| | tai80b | 80 | 818,415,043 | 0.01 | 1.14 | 0.61 | 0.46 | 19.16 | 21.26 | 0.90 |
| | tai100b | 100 | 1,185,996,137 | 0.01 | 0.55 | 0.28 | 0.21 | 43.92 | 48.48 | 0.91 |
| | tai150b | 150 | 498,896,643 | 0.72 | 1.04 | 0.85 | 0.12 | 244.18 | 270.53 | 0.90 |
| | **Partial average** | | | **0.09** | **0.42** | **0.24** | **0.13** | **39.70** | **44.09** | **0.78** |
| Category IV | bur26a | 26 | 5,426,670 | 0 (3) | 1.49 | 0.67 | 0.74 | 0.22 | 0.51 | 0.43 |
| | chr15a | 15 | 9896 | 0 (6) | 0.83 | 0.23 | 0.31 | 0.05 | 0.10 | 0.50 |
| | chr25a | 25 | 3796 | 0 (8) | 4.79 | 0.95 | 1.80 | 0.27 | 0.37 | 0.73 |
| | els19 | 19 | 17,212,548 | 0 (10) | 0.00 | 0.00 | 0.00 | 0.27 | 0.37 | 0.73 |
| | esc64a | 64 | 116 | 0 (10) | 0.00 | 0.00 | 0.00 | 0.72 | 8.66 | 0.08 |
| | kra30a | 30 | 88,900 | 0 (5) | 1.57 | 0.79 | 0.74 | 0.26 | 0.58 | 0.45 |
| | kra30b | 30 | 91,420 | 0 (5) | 0.25 | 0.07 | 0.08 | 0.20 | 0.57 | 0.35 |
| | ste36a | 36 | 9526 | 0 (3) | 1.45 | 0.48 | 0.50 | 0.92 | 1.24 | 0.74 |
| | **Partial average** | | | **0.00** | **1.30** | **0.40** | **0.52** | **0.36** | **1.55** | **0.50** |
| | **Grand average** | | | **0.29** | **0.94** | **0.55** | **0.26** | **19.19** | **25.65** | **0.65** |

specified by the fitness distance correlation (FDC) coefficient, denoted by r, is defined for a set of distances from known optimum $D = \{d_1, d_2, ..., d_n\}$ with corresponding finesses $F = \{f_1, f_2, ..., f_n\}$ as follows (Jones & Forrest, 1995):

$$\mathrm{Cov}(F,D)=\frac{1}{n}\sum_{i=1}^{n}(f_i-\bar{f})(d_i-\bar{d}) \tag{5}$$

$$r=\frac{\mathrm{Cov}(F,D)}{\sigma_F\sigma_D} \tag{6}$$

where Cov(*F, D*) is the covariance of *F* and *D*; $\bar{f},\bar{d},\sigma_F$ and $\sigma_D$ are mean and standard deviation of *F* and *D* respectively. For minimization problem, as fitness distance correlation coefficient increases the problem becomes easier. Value $r=1$ indicates perfect correlation between fitness and distance to the optimum, and $r=-1$ indicates that the fitness function is completely misleading. Further as

**Table 2. Average distances and fitness distance correlation coefficient based on landscape sampling by LSGA**

| | Instance | *d* | *q* | *r* |
|---|---|---|---|---|
| Category I | had20 | 15.32 | 0.01 | 0.36 |
| | lipa40b | 32.56 | 0.02 | 0.42 |
| | rou20 | 17.35 | 0.10 | −0.51 |
| | tai20a | 18.20 | 0.54 | 0.01 |
| | tai30a | 27.50 | 1.22 | −0.29 |
| | tai40a | 38.25 | 1.72 | −0.12 |
| | tai50a | 48.35 | 2.05 | 0.01 |
| | tai60a | 59.05 | 2.13 | −0.10 |
| | tai80a | 77.75 | 2.17 | 0.13 |
| | tai100a | 98.75 | 1.95 | −0.06 |
| Category II | nug30 | 27.65 | 0.06 | 0.15 |
| | sko42 | 38.60 | 0.19 | −0.17 |
| | sko49 | 45.40 | 0.25 | 0.26 |
| | sko81 | 78.75 | 0.30 | 0.24 |
| | sko90 | 88.65 | 0.42 | 0.01 |
| | sko100a | 96.25 | 0.39 | 0.03 |
| | sko100d | 97.25 | 0.44 | 0.83 |
| | tho150 | 149.3 | 0.49 | −0.45 |
| | wil50 | 46.40 | 0.07 | −0.02 |
| Category III | tai20b | 18.80 | 0.01 | 0.17 |
| | tai30b | 27.40 | 0.01 | −0.08 |
| | tai40b | 37.35 | 0.03 | −0.07 |
| | tai50b | 48.25 | 0.15 | 0.35 |
| | tai60b | 57.15 | 0.06 | −0.03 |
| | tai80b | 75.50 | 0.52 | 0.11 |
| | tai100b | 91.80 | 0.23 | 0.12 |
| | tai150b | 149.25 | 0.69 | 0.03 |
| Category IV | bur26a | 25.90 | 0.48 | 0.76 |
| | chr15a | 13.15 | 0.23 | 0.88 |
| | chr25a | 24.90 | 0.95 | 0.89 |
| | els19 | 18.10 | 0.00 | 0.36 |
| | esc64a | 63.30 | 0.00 | 0.21 |
| | kra30a | 28.65 | 0.79 | 0.71 |
| | kra30b | 26.70 | 0.07 | 0.68 |
| | ste36a | 34.40 | 0.48 | 0.29 |

**Table 3. Comparative study of LSGA and UPSO for 35 QAPLIB problem instances**

| | Instance | LSGA | | | | UPSO | | |
|---|---|---|---|---|---|---|---|---|
| | | BPE | StdDev | FT | CT | BPE | StdDev | Time |
| Category I | had20 | **0.00** | 0.00 | 0.03 | 0.13 | 0.06 | 0.24 | 10.27 |
| | lipa40b | **0.00** | 0.00 | 0.25 | 1.50 | **0.00** | 8.07 | 12.98 |
| | rou20 | **0.00** | 0.13 | 0.06 | 0.15 | 0.56 | 1.42 | 9.31 |
| | tai20a | **0.30** | 0.35 | 0.04 | 0.11 | 2.80 | 0.88 | 10.93 |
| | tai30a | **0.48** | 0.50 | 0.33 | 0.56 | 2.50 | 0.61 | 13.26 |
| | tai40a | **1.06** | 0.41 | 0.98 | 1.30 | 3.40 | 0.21 | 14.48 |
| | tai50a | **1.62** | 0.36 | 2.41 | 3.09 | 4.01 | 0.28 | 16.57 |
| | tai60a | **1.49** | 0.39 | 5.15 | 6.34 | 3.79 | 0.30 | 18.26 |
| | tai80a | **1.53** | 0.30 | 18.13 | 20.50 | 3.43 | 0.29 | 22.98 |
| | tai100a | **1.53** | 0.23 | 43.36 | 47.51 | 3.51 | 0.21 | 29.54 |
| Partial average | | **0.80** | **0.27** | **7.07** | **8.12** | 2.41 | 1.25 | 15.86 |
| Category II | nug30 | **0.00** | 0.07 | 0.36 | 0.54 | 1.34 | 0.72 | 13.27 |
| | sko42 | **0.00** | 0.17 | 1.30 | 1.83 | 0.77 | 0.75 | 13.45 |
| | sko49 | **0.14** | 0.15 | 2.62 | 3.33 | 0.67 | 0.55 | 14.75 |
| | sko81 | **0.10** | 0.17 | 4.50 | 7.94 | 1.72 | 0.30 | 21.16 |
| | sko90 | **0.33** | 0.08 | 32.87 | 36.56 | 1.00 | 0.43 | 24.51 |
| | sko100a | **0.26** | 0.11 | 52.67 | 56.95 | 1.66 | 0.28 | 29.25 |
| | sko100d | **0.32** | 0.11 | 50.03 | 56.18 | 1.68 | 0.22 | 27.43 |
| | tho150 | **0.23** | 0.21 | 133.44 | 284.74 | 2.74 | 0.39 | 49.35 |
| | wil50 | **0.02** | 0.05 | 2.52 | 3.42 | 0.73 | 0.32 | 16.15 |
| Partial average | | **0.16** | **0.12** | 31.15 | 50.17 | 1.37 | 0.44 | **23.26** |
| Category III | tai20b | **0.00** | 0.00 | 0.03 | 0.10 | **0.00** | 1.17 | 10.43 |
| | tai30b | **0.00** | 0.00 | 0.45 | 0.55 | 0.01 | 2.13 | 12.55 |
| | tai40b | **0.00** | 0.00 | 1.05 | 1.49 | 0.90 | 2.55 | 14.66 |
| | tai50b | **0.00** | 0.18 | 2.89 | 3.45 | 1.03 | 1.34 | 16.94 |
| | tai60b | **0.00** | 0.04 | 5.95 | 6.85 | 1.03 | 3.25 | 18.67 |
| | tai80b | **0.01** | 0.46 | 19.16 | 21.26 | 2.51 | 1.06 | 22.93 |
| | tai100b | **0.01** | 0.21 | 43.92 | 48.48 | 1.53 | 1.42 | 28.27 |
| | tai150b | **0.72** | 0.12 | 244.18 | 270.53 | 3.36 | 0.54 | 49.05 |
| Partial average | | **0.09** | **0.13** | 39.70 | 44.09 | 1.30 | 1.68 | **21.69** |
| Category IV | bur26a | **0.00** | 0.74 | 0.22 | 0.51 | 0.02 | 0.11 | 11.15 |
| | chr15a | **0.00** | 0.31 | 0.05 | 0.10 | 0.40 | 4.85 | 10.37 |
| | chr25a | **0.00** | 1.80 | 0.27 | 0.37 | 10.85 | 12.05 | 11.19 |
| | els19 | **0.00** | 0.00 | 0.27 | 0.37 | **0.00** | 7.60 | 10.16 |
| | esc64a | **0.00** | 0.00 | 0.72 | 8.66 | **0.00** | 0.00 | 19.87 |
| | kra30a | **0.00** | 0.74 | 0.26 | 0.58 | 2.25 | 1.06 | 12.76 |
| | kra30b | **0.00** | 0.08 | 0.20 | 0.57 | 0.42 | 1.53 | 13.61 |
| | ste36a | **0.00** | 0.50 | 0.92 | 1.24 | 3.21 | 2.81 | 12.84 |
| Partial average | | **0.00** | **0.52** | **0.36** | **1.55** | 2.14 | 3.75 | 12.74 |
| Grand average | | **0.29** | **0.26** | 19.19 | 25.65 | 1.83 | 1.71 | **18.38** |

Note: Bold values show the better solution quality as well as less computational times.

suggested by Vanneschi et al. (2003), if the value of r lies in the range (−0.15, 0.15), then the behaviour of the instance is unpredictable. As the aim is to understand the behaviour of LSGA, so, FLA is suitable for the purpose. FLA is the indicator for both algorithm's behaviour and the problem landscape.

For the present experiment, hamming distance from the best-known solution is used as a distance measure for each instance. As a sample for the fitness landscape for each instance, LSGA is run ten thousand times. Table 2 shows average distance of the samples from the best-known solution (*d*), average quality of the samples, *q* (same as APE) and FDC coefficient (*r*).

It is seen from the FDC coefficient (*r*) in Table 2, there is a difference in correlation among instances of different categories. For randomly generated instances (Category I), except for the four instances—had20, lipa40b, rou20 and tai30a; FDC coefficient of other instances fall in (−0.15, 0.15), that is, there is no correlation between fitness and distance. For grid-distance-based instances (Category II), significant FDC coefficient exists except for three instances—sko90, sko100a and wil50. For real-life-like instances (Category III), except for the two instances—tai20b and tai50b—FDC coefficients suggest that there is no correlation between fitness and distance. For all real-life instances (Category IV), FDC coefficients are found to be significant. That is, there is strong correlation between fitness and distance for those real-life instances.

The LSGA is now compared with unified particle swarm optimization (UPSO) (Hafiz & Abdennour, 2016) for the same 35 QAPLIB instances. It is to be noted that UPSO has been run on a similar PC with Intel i7 processor and 8 GB RAM. The results are reported in Table 3. It is very interesting to observe that LSGA (0.29% Grand Average BPE) outperformed UPSO (1.83% Grand Average BPE) on all category instances. However, for four instances—lipa40b, tai20b, els19 and esc64a—performance of both algorithms is same. In terms of computational time, except for seven instances LSGA takes lesser time than UPSO. Hence, in terms of the solution quality as well as computational time, LSGA is found to be better. This can be seen very clearly in the Figure 2. Also, another comparative study is carried out between LSGA and another Discrete Particle Swarm Optimization (DPSO) by Pradeepmon, Sridharan, and Panicker (2018) for only 6 instances. This is reported in Table 4. By looking at the table, one can conclude that LSGA is better.

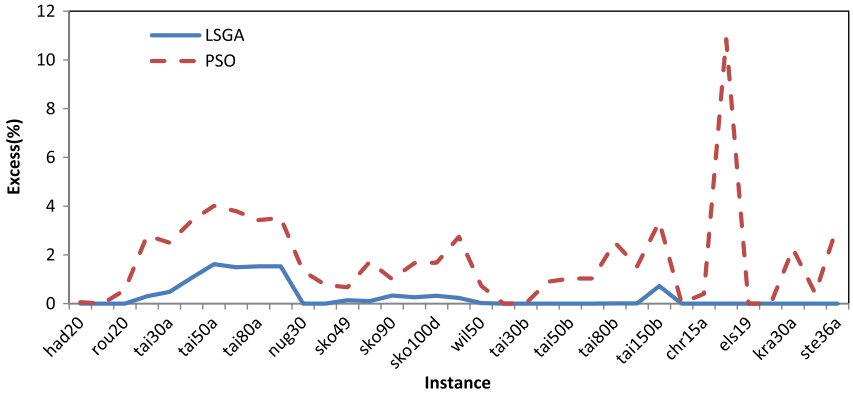



**Figure 2. Comparative study between LSGA and UPSO.**

**Table 4. Comparative study of LSGA and DPSO for 6 instances**

| | Instance | Size | BKV | LSGA | | | | DPSO | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | BPE | WPE | APE | StdDev | BPE | WPE | APE |
| 1 | bur26a | 26 | 5,426,670 | 0 (3) | 1.49 | 0.67 | 0.74 | 0.150 | 0.447 | 0.272 |
| 2 | had20 | 20 | 6922 | 0 (10) | 0.00 | 0.00 | 0.00 | 0.000 | 2.080 | 0.364 |
| 3 | kra30a | 30 | 88,900 | 0 (5) | 1.57 | 0.79 | 0.74 | 2.542 | 9.224 | 6.864 |
| 4 | kra30b | 30 | 91,420 | 0 (5) | 0.25 | 0.07 | 0.08 | 1.903 | 6.607 | 3.975 |
| 5 | Nug30 | 30 | 6124 | 0 (5) | 0.20 | 0.06 | 0.07 | 1.339 | 5.062 | 3.011 |
| 6 | rou20 | 20 | 725,522 | 0 (6) | 0.49 | 0.09 | 0.13 | 1.837 | 5.988 | 4.268 |

## 4. Conclusions and discussion

A hybrid algorithm (LSGA) that combines lexisearch and genetic algorithms has been proposed for finding effective solution to the quadratic assignment problem. As starting with a good initial population leads faster convergence of GA, a simple lexisearch algorithm with limited iterations is used for generating initial population. About the other operators, self-adaptively three crossover operators—sequential constructive crossover, one-point crossover and swap path crossover; a randomly chosen mutation operator out of four mutation operators—adaptive mutation, exchange mutation, three-exchange mutation and gene-exchange mutation. Further, multi-parent sequential constructive crossover along with combined mutation is used as immigration method to diversify the search space.

Experimental results on four categories of benchmark QAPLIB instances show the effectiveness of the proposed LSGA. Out of 35 instances 18 instances have been solved optimally, at least once in ten runs and for the remaining instances, solutions are very close to the optimal solutions. It is seen that the fourth category problems are the easiest to solve by LSGA, whereas, first the category problems are difficult to solve.

To have depth search behaviour of LSGA on the problem, a fitness landscape analysis is performed. This study shows that there is strong correlation between fitness and distance for real-life instances; however, there is a poor correlation between fitness and distance for random instances.

Then a comparative study between LSGA and unified particle swarm optimization (UPSO) is presented for the same instances. It is seen that LSGA outperformed UPSO on all category instances. In terms of computational time, except for seven instances LSGA takes lesser time than UPSO. Hence, in terms of the solution quality as well as computational time LSGA is found to be better. Though LSGA is found to be better than UPSO, however, for some instances, it does not find best known solution within ten runs. Hence, a better local search and immigration methods may further improve the solution quality and hence, may obtain best known solutions for the remaining instances also, which is under the investigation.

**Acknowledgment**
The author is very much thankful to the NSTIP for its financial and technical supports. The author is also very much thankful to the honourable reviewers for their valuable comments and suggestions which helped to improve the quality of the manuscript.

**Funding**
This research was supported by the NSTIP strategic technologies program No. 10 in the Kingdom of Saudi Arabia vide [grant number 11-INF1788-08].

**Author details**
Zakir Hussain Ahmed[1]
E-mail: zaahmed@imamu.edu.sa
ORCID ID: http://orcid.org/0000-0003-1938-6137
[1] Department of Computer Science, Al Imam Mohammad Ibn Saud Islamic University (IMSIU), P.O. Box No. 5701, Riyadh 11432, Kingdom of Saudi Arabia.

**Citation information**
Cite this article as: A hybrid algorithm combining lexisearch and genetic algorithms for the quadratic assignment problem, Zakir Hussain Ahmed, *Cogent Engineering* (2018), 5: 1423743.

**References**

Ahmed, Z. H. (2010a). Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. *International Journal of Biometrics & Bioinformatics, 3*(6), 96–105.

Ahmed, Z. H. (2010b). A hybrid sequential constructive sampling algorithm for the bottleneck traveling salesman problem. *International Journal of Computational Intelligence Research, 6*(3), 475–484.

Ahmed, Z. H. (2013a). A new reformulation and an exact algorithm for the quadratic assignment problem'. *Indian Journal of Science and Technology, 6*(4), 4368–4377.

Ahmed, Z. H. (2013b). A hybrid genetic algorithm for the bottleneck traveling salesman problem. *ACM Transactions on Embedded Computing Systems, 12*, 9.

Ahmed, Z. H. (2014). A simple genetic algorithm using sequential constructive crossover for the quadratic assignment problem. *Journal of Scientific & Industrial Research, 73*, 763–766.

Ahmed, Z. H. (2015a). A multi-parent genetic algorithm for the quadratic assignment problem. *OPSEARCH, 52*(4), 714–732. https://doi.org/10.1007/s12597-015-0208-7

Ahmed, Z. H. (2015b). An improved genetic algorithm using adaptive mutation operator for the quadratic assignment problem. In *38th international conference on telecommunications and signal processing (TSP)*, 1–5. IEEE.

Ahmed, Z. H. (2016). Experimental analysis of crossover and mutation operators on the quadratic assignment problem. *Annals of Operations Research, 247*(2), 833–851. https://doi.org/10.1007/s10479-015-1848-y

Ahuja, R., Orlin, J. B., & Tiwari, A. (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research, 27*, 917–934. https://doi.org/10.1016/S0305-0548(99)00067-2

Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (1998). On the solution of the traveling salesman problem. *Documenta Mathematica, Extra Volume ICM,* III, 645–656.

Archetti, C., Speranza, M. G., & Savelsbergh, M. W. P. (2008). An optimization-based heuristic for the split delivery vehicle routing problem. *Transportation Science, 42*, 22–31. https://doi.org/10.1287/trsc.1070.0204

Burkard, R., Karisch, S., & Rendl, F. (1997). QAPLIB—A quadratic assignment problem library. *Journal of Global Optimization, 10*, 391–403. https://doi.org/10.1023/A:1008293323270

Burke, E. K., Cowling, P. I., & Keuthen, R. (2001). Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, & H. Tijink (Eds.), *Applications of evolutionary computing: Evo workshops 2001* (Vol. 2037 of LNCS, pp. 203–212). Springer. doi:10.1007/3-540-45365-2

Czapiński, M. (2013). An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform. *Journal of Parallel and Distributed Computing, 73*(11), 1461–1468. https://doi.org/10.1016/j.jpdc.2012.07.014

Deb, K. (1995). *Optimization for engineering design: Algorithms and examples*. New Delhi: Prentice Hall of India Pvt. Ltd..

Franceschi, R., Fischetti, M., & Toth, P. (2006). A new ILP-based refinement heuristic for vehicle routing problems. *Mathematical Programming, 105*, 471–499. https://doi.org/10.1007/s10107-005-0662-8

Guimarans, D., Herrero, R., Riera, D., Juan, A. A., & Ramos, J. J. (2011). Combining probabilistic algorithms, constraint programming and lagrangian relaxation to solve the vehicle routing problem. *Annals of Mathematics and Artificial Intelligence, 62*, 299–315. https://doi.org/10.1007/s10472-011-9261-y

Hafiz, F., & Abdennour, A. (2016). Particle Swarm Algorithm variants for the quadratic assignment problems—A probabilistic learning approach. *Expert Systems with Applications, 44*, 413–431. https://doi.org/10.1016/j.eswa.2015.09.032

Hewitt, M., Nemhauser, G. L., & Savelsbergh, M. W. P. (2009). Combining exact and heuristic approaches for the capacitated fixed charge network flow problem. *INFORMS Journal on Computing, 22*(2), 314–325.

Holborn, P. L., Thompson, J. M., & Lewis, R.. (2012). Combining heuristic and exact methods to solve the vehicle routing problem with pickups, deliveries and time windows. In J.-K. Hao & M. Middendorf (Eds.), *EvoCOP 2012* (pp. 63–74). LNCS.

Hong, G. (2013). A hybrid ant colony algorithm for quadratic assignment problem. *The Open Electrical and Electronic Engineering Journal, 7*, 51–54. https://doi.org/10.2174/1874129001307010051

Jones, T., & Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. *In ICGA, 95*, 184–192.

Koopmans, T. C., & Beckmann, M. J. (1957). Assignment problems and the location of economic activities. *Econometrica, 25*, 53–76. https://doi.org/10.2307/1907742

Lim, M. H., Yuan, Y., & Omatu, S. (2000). Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem. *Computational Optimization and Applications, 15*, 249–268. https://doi.org/10.1023/A:1008743718053

Long, Q., & Wu, C. (2014). A hybrid method combining genetic algorithm and Hooke-Jeeves method for constrained global optimization. *Journal of Industrial and Management Optimization, 10*(4), 1279–1296. https://doi.org/10.3934/jimo

Merz, P., & Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation, 4*, 337–352.

Mezmaz, M., Melab, N., & Talbi, E. G. (2007). Combining metaheuristics and exact methods for solving exactly multi-objective problems on the grid. *Journal of Mathematical Modelling and Algorithms, 6*(3), 393–409. https://doi.org/10.1007/s10852-007-9063-8

Misevicius, A. (2003). A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica, 14*(4), 497–514.

Nagar, A., Heragu, S. S., & Haddock, J. (1995). A meta-heuristic algorithm for a bi-criteria scheduling problem. *Annals of Operations Research, 63*, 397–414.

Nwana, V., Darby-Dowman, K., & Mitra, G. (2005). A cooperative parallel heuristic for mixed zero-one linear programming. *European Journal of Operations Research, 164*, 12–23.

Plateau, A., Tachat, D., & Tolla, P. (2002). A hybrid search combining interior point methods and metaheuristics for 0–1 programming. *International Transactions in Operational Research, 9*, 731–746. https://doi.org/10.1111/itor.2002.9.issue-6

Pradeepmon, T. G., Sridharan, R., & Panicker, V. V. (2018). Development of modified discrete particle swarm optimization algorithm for quadratic assignment problems. *International Journal of Industrial Engineering Computations, 9*. doi:10.5267/j.ijiec.2017.11.003

Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In J. Mira & J.R. ′Alvarez (Eds.), *IWINAC 2005* (pp. 41–53). LNCS 3562.

Raidl, G. R., & Feltl, H. (2004). An improved hybrid genetic algorithm for the generalized assignment problem. In G. B. Lamont, H. Haddad, G. A. Papadopoulos, & B. Panda (Eds.), *Proceedings of the 2003ACM Symposium on Applied Computing* (pp. 990–995). ACM Press.

Sahni, S., & Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM, 23*, 555–565. https://doi.org/10.1145/321958.321975

Uwate, Y., Nishio, Y., Ueta, T., Kawabe, T., & Ikeguchi, T. (2004). Performance of chaos and burst noises injected to the hopfield NN for quadratic assignment problems. In *IEICE transactions on fundamentals of electronics, communications and computer sciences, E87-A (4)* (pp. 937–943).

Vanneschi, L., Tomassini, M., Clergue, M., & Collard, P. (2003). Difficulty of unimodal and multimodal landscapes in genetic programming. In *Genetic and Evolutionary Computation—GECCOO* (pp. 1788–1799). Springer.