Performance Analysis of Hybrid Genetic Algorithms for the Generalized Assignment Problem

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

Abstract
This paper presents three hybrid algorithms, based on genetic algorithm (GA), to solve the generalized assignment problem (GAP) with the objective to minimize the assignment cost under the limitation of the agent capacity. First sequential constructive crossover (SCX) is modified for the problem and then showed competence over one-point crossover (OPX). Our hybrid algorithms use SCX, exchange mutation and three local search algorithms. Experimental results on four sets of benchmark instances from OR-library show the effectiveness of the proposed hybrid algorithms. The proposed algorithms are then compared with bee (BEE) and differential evolution (DE-SK) algorithms. In terms of solution quality as well as computational times, one of our hybrid genetic algorithm (HGA3) outperformed both BEE as well as DE-SK.

Key words:
Generalized assignment problem; hybrid genetic algorithm; sequential constructive crossover; constructive mutation; local search.

1. Introduction

The generalized assignment problem (GAP) belongs to the class of NP-hard problems and is considered as one of the most difficult problems. The problem seeks a minimum cost (or maximum profit) of assignment of n jobs to m agents such that each job is assigned to precisely one agent subject to resource restrictions on the agents. Let us formally define the problem as follows. Let I = {1, 2, ..., m} be a set of agents, and let J = {1, 2, ..., n} be a set of jobs. For any iϵI, jϵJ, given cost matrix, C=[cij], where cij is the cost of assigning job j to agent i (or assigning agent i to job j), resource matrix, R = [rij], where rij is the resource required by agent i to perform job j, and capacity vector, B = [bi], where bi is the resource availability (capacity) of agent i. Also, xij is a 0-1 variable, that is, 1 if agent i performs job j and 0 otherwise. The mathematical formulation of the problem is:

$$Mininmize \ \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} \quad (1)$$

$$Subject\ to \ \sum_{i\in I} x_{ij} = 1, \forall j \in J \quad (2)$$

$$\sum_{j\in J} r_{ij}x_{ij} \le b_i, \ \forall i \in I \quad (3)$$

$$x_{ij} \in \{0,1\}, \qquad \forall i \in I; \ \forall j \in J \quad (4)$$

Equation (2) ensures that each job is assigned to exactly one agent and equation (3) ensures that the total resource requirement of the jobs assigned to an agent does not exceed the capacity of the agent. The GAP was first introduced by Ross and Soland (1975), which is shown to be NP-hard. Many real-life applications can be modeled as a GAP (Narciso and Lorena, 1999). Various approaches can be found to solve this problem, some of which were summarized (Catrysse and Van Wassenhove, 1992). Since, the problem is NP-hard, it is very difficult to solve, and in fact, to date no optimal algorithm has been found, which is able to solve the problem in polynomial time. So, heuristic algorithms have been used to solve the instances of the problem. Out of heuristics, genetic algorithms (GAs) are successfully implemented to this kind of combinatorial optimization problems (Goldberg, 1989).

Experiencing the novel merits of GA, many researchers started GA application initially to GAP. Application of a GA to improve the existing GAP solution was proposed by Juell et al. (2003). An improved hybrid GA for the GAP to find minimum cost assignment for a set of jobs to a set of agents was proposed by Feltl et al. (2004). A path relinking approach with ejection chains was proposed for the problem, which is one of the best performing algorithms (Yagiura et al., 2006). Bees algorithm (BEE) with an ejection chain neighborhood mechanism was developed for solving the GAP by Özbakir et al. (2010). A computational study has been reported and the results are compared with algorithms from the literature. Eight different improved differential evolution (DE) algorithms were developed for solving the problem by Sethanan and Pitakaso (2016). Each algorithm uses DE with a different combination of local search techniques.

We are going to use GA for obtaining heuristically optimal solution to the problem. Selection, crossover and mutation are three basic operators in GA, of which crossover is the most important operator. So, numerous crossover operators have been developed for solving various optimization problems. However, most of the researchers use one-point crossover (OPX) operator for the GAP (Chu and Beasley, 1997). In this paper, we modify the sequential constructive crossover (SCX) (Ahmed, 2010a) operator for the problem. Then a GA based on SCX is developed for solving the GAP, which is then compared with GA using OPX for some

Manuscript received September 5, 2019
Manuscript revised September 20, 2019

benchmark instances reported in the literature (Beasley, 1990). Experimental results show that SCX operator is better than OPX. Then our GA is incorporated with three local search algorithms to enhance the solution. Finally, a comparative study is carried out of our algorithms against two state-of-art heuristic algorithms, namely, BEE (Özbakir et al., 2010) and DE (Sethanan and Pitakaso, 2016).

2. Proposed Genetic Algorithm

In GAs, first, solutions are encoded as feasible chromosomes (or individuals) such that the genetic operators result in feasible chromosomes. A simple GA begins by generating set (initial population) of chromosomes randomly, and then goes through (possibly) three genetic operators to create new, and optimistically, better populations as following generations. For the GAP, to meet this requirement, we use an efficient representation in which the solution structure is an ordered structure (n-dimensional vector) of integer numbers. These integer numbers identify the agents, as assigned to vector elements denoted by the jobs, as follows.

Job	1	2	3	.....	n-1	n
Agent	3	1	m	.....	2	1

This representation ensures that all the equality constraints in (2) are automatically satisfied since exactly one agent is assigned to each job. However, this representation does not guarantee that the capacity constraints in (3) will be satisfied. The steps involved in our GA heuristic for the GAP are as follows:

2.1 Initial Population and Objective Function

We use a simple version of sequential constructive sampling algorithm for generating a chromosome of size n. For that, we first construct an alphabet table (Ahmed, 2010b) based on the resource matrix as follows. Alphabet matrix, A=[a(i,j)], is a matrix of order mXn formed by positions of elements of the resource matrix, R=[rij]. The jth column of matrix A consists of the positions of the elements in the jth column of the matrix R when they are arranged in the non-decreasing order of their values. If a(p, j) stands for the pth element in the jth column of A, then a(1, j) corresponds to the position of smallest element in jth column of the matrix R. Alphabet table "$[a(i,j) - r_{a(i,j),j}]$" is the combination of elements of matrix A and their values. Our algorithm for generating a chromosome may be summarized as follows:

Step 0: Construct the 'alphabet table' based on the given resource matrix.

Step 1: Consider a 'legitimate' random job by generating random number, p, between 1 and n. Go to step 2.

Step 2: Go to the pth column of the 'alphabet table', and assign first agent (say, 'agent q') of the column such that so far total resource required by the 'agent q' is less than the capacity of the agent. If assigning that agent exceeds the capacity of the agent, then choose the next possible agent sequentially in that column. If no such agent can be assigned for that job, delete the present incomplete chromosome and try to generate another chromosome from the beginning, else, go to step 3.

Step 3: If agents are assigned for all jobs, then stop, else go to step 1.

Consider a problem to assign n=15 jobs to m=5 agents with cost, resource and capacity matrices are given in Table 1, Table 2 and Table 3 respectively. The alphabet table, based on the resource matrix, is shown in Table 4. For example, 1-8 in first column stands for 8 unit of resource required by 1st agent to perform 1st job. Suppose, randomly generated job sequence is {3, 14, 5, 1, 7, 12, 4, 15, 10, 8, 11, 2, 13, 9, 6}, then a chromosome may be constructed as given in Table 5. Note that the notation R1 means resource used by agent 1. So, we generate a chromosome as (1, 2, 3, 5, 1, 2, 4, 3, 1, 4, 4, 4, 5, 1, 3), and the complete assignment of the jobs to the agents is (job↔ agent) {1↔1, 2↔2, 3↔3, 4↔5, 5↔1, 6↔2, 7↔4, 8↔3, 9↔1, 10↔4, 11↔4, 12↔4, 13↔5, 14↔1, 15↔3}. This assignment yields the total assignment cost of 295 (= 17+ 16+ 16+ 15+ 24+ 16+ 19+ 19+ 19+ 19+ 25+ 23+ 19+ 24+ 24) units.

Table 1: Cost matrix C

Job→		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Agent	**1**	17	21	22	18	24	15	20	18	19	18	16	22	24	24	16
	2	23	16	21	16	17	16	19	25	18	21	17	15	25	17	24
	3	16	20	16	25	24	16	17	19	19	18	20	16	17	21	24
	4	19	19	22	22	20	16	19	17	21	19	25	23	25	25	25
	5	18	19	15	15	21	25	16	16	23	15	22	17	19	22	24

Table 2: Resource matrix R

Job→		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Agent	**1**	8	15	14	23	8	16	8	25	9	17	25	15	10	8	24
	2	15	7	23	22	11	11	12	10	17	16	7	16	10	18	22
	3	21	20	6	22	24	10	24	9	21	14	11	14	11	19	16
	4	20	11	8	14	9	5	6	19	19	7	6	6	13	9	18
	5	8	13	13	13	10	20	25	16	16	17	10	10	5	12	23

Table 3: Capacity vector B

Agent	1	2	3	4	5
Capacity	36	34	38	27	33

Table 4: Alphabet table based on resource matrix (R is the resource required)

1-R	**2-R**	**3-R**	**4-R**	**5-R**	**6-R**	**7-R**	**8-R**	**9-R**	**10-R**	**11-R**	**12-R**	**13-R**	**14-R**	**15-R**
1- 8	2- 7	3- 6	5-13	1- 8	4- 5	4- 6	3- 9	1- 9	4- 7	4- 6	4- 6	5- 5	1- 8	3-16
5- 8	4-11	4- 8	4-14	4- 9	3-10	1- 8	2-10	5-16	3-14	2- 7	5-10	1-10	4- 9	4-18
2-15	5-13	5-13	2-22	5-10	2-11	2-12	5-16	2-17	2-16	5-10	3-14	2-10	5-12	2-22
4-20	1-15	1-14	3-22	2-11	1-16	3-24	4-19	4-19	1-17	3-11	1-15	3-11	2-18	5-23
3-21	3-20	2-23	1-23	3-24	5-20	5-25	1-25	3-21	5-17	1-25	2-16	4-13	3-19	1-24

Table 5: Construction of a chromosome

Job	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Chromosome	1-8 R_1= 16+8 =24	2-7 R_2=7	3-6 R_3=6	5-13 R_5=13	1-8 R_1= 8+8 =16	4-5 R_4= 25+5 =29>27 Next 3-10 R_3= 31+10 =41>38 Next 2-11 R_2=7+11 =18	4-6 R_4=6	3-9 R_3= 22+9 =31	1-9 R_1= 24+9 =33	4-7 R_4= 12+7 =19	4-6 R_4= 19+6 =25	4-6 R_4= 6+6 =12	5-5 R_5= 13+5 =18	1-8 R_1=8	3-16 R_3= 6+16 =22

2.2 Fitness Function and Selection Method

The fitness function is the objective function for our problem. However, during the GA search there is a possibility of obtaining infeasible solution that violates the capacity constraint. So, it must consider not just the assignment cost, but also the degree of infeasibility of a solution. Rather than penalize the fitness (or the objective function) when a solution is infeasible, which is a common approach in GAs, and associate two values with each solution. One of these values is called fitness and the other is called unfitness. Let skj represent the agent assigned to job j (j= 1, 2, ..., n) in solution k (k= 1, 2, ..., N). The fitness fk, of solution k is equal to its objective function value as calculated by $f_k = \sum_{j=1}^{n} c_{s_{kj},j}$. The unfitness uk of solution k is a measure of infeasibility (in relative terms) as calculated by $u_k = \sum_{i=1}^{n} max\left[0, \left(\sum_{j \in J, s_{kj}=i} r_{ij}\right) - b_i\right]$. Note here that uk =0 if and only if solution k is feasible. So, fitness function is $f_k = f_k + u_k$. In our GA, the stochastic remainder selection method (Deb, 1995) is used for selection.

2.3 Sequential Constructive Crossover Operator

Crossover is the most important operator in GAs where one (or two) offspring chromosome(s) is (are) generated first by applying to pair of selected parent chromosomes. One can use one-point or multi-point crossover operator. Among various crossover operators, the sequential constructive crossover (SCX) is successfully applied to many combinatorial optimization problems by Ahmed (2013a, 2013b, 2014a, 2014b, 2018, 2019), Bennaceur and Ahmed (2017), Al-Omeer and Ahmed (2019). We are going to modify the SCX, apply on our problem and compare with one-point crossover (OPX) proposed by Chu and Beasley (1997). Our modified SCX operator for the GAP is as follows:

Step 0: Start from 'job 1' (i.e., current job i =1).

Step 1: Suppose 'agent α' and 'agent β' are assigned in 1st and 2nd parent respectively, then for selecting the agent for 'job i' go to Step 2.

Step 2: If ciα < ciβ, then select 'agent α', otherwise, 'agent β' as the agent for 'job i', provided that the selected agent does not violate the capacity constraint. If the agent violates the constraint, then search sequentially all agents according to their cost in ascending order and consider the first one that does not violate the constraint. Then concatenate the selected agent to the partially constructed offspring chromosome. If the offspring is a complete chromosome, then stop, otherwise, set i = i + 1, and go to Step 1.

Let us consider a pair of chromosomes P1: (1, 2, 3, 5, 1, 2, 4, 3, 1, 2, 5, 4, 5, 4, 3) and P2: (1, 2, 3, 5, 1, 4, 4, 3, 1, 4, 4, 5, 5, 1, 3) with costs 295 and 289 respectively with respect to the cost matrix given in Table 1. By applying above SCX as in Table 6, we obtain the offspring (1, 2, 3, 5, 1, 2, 4, 3, 1, 4, 5, 5, 5, 1, 3) with value 284, which is less than both parents.

The crossover procedure is followed by a mutation procedure. We consider exchange mutation with mutation

probability (Pm) that selects two genes randomly and then exchanges them. The probability of applying mutation is set very low, whereas the probability of crossover is set very high. A simple GA repeatedly applies these (three) operators until either the population converges or until reaches the maximum number of generations (iterations).

Table 6: Working of the sequential constructive crossover operator

Job	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
P_1	1	2	3	5	1	2	4	3	1	2	5	4	5	4	3
P_2	1	2	3	5	1	4	4	3	1	4	4	5	5	1	3
O	1(17) R_1=8	2(16) R_2=7	3(16) R_3=6	5(15) R_5=13	1(24) R_1= 8+8 =16	2(16) R_2= 7+11 =18	4(19) R_4=6	3(19) R_3= 6+9 =15	1(19) R_1= 16+9 =25	4(19) R_4= 6+7 =13	5(22) R_5= 13+10 =23	5(17) R_5= 23+10 =33	5(19) R_5= 33+5 =38 >33 Next 3(17) R_3= 15+11 =26	1(24) R_1= 25+8 =33	3(24) R_3= 15+16 =31

Our GAs may be summarized as follows (Ahmed, 2010a).

```
Genetic Algorithm ()
{   Initialize population;
    Evaluate the population;
    Generation = 0;
    While termination condition is not satisfied
    {       Generation = Generation + 1;
            Select good chromosomes by selection procedure;
            Perform crossover with crossover probability (Pc);
            Perform mutation with mutation probability (Pm);
            Evaluate the population;
    }
}
```

A GA that incorporates local search is called hybrid GA. We are considering following two mutation operators with 100% probability as local search methods.

2.4 Local Search

Two local search methods, 3-exchange mutation and constructive mutation are proposed here. The first one is old (Ahmed, 2016) and the second one is new. Both are described below.

2.4.1 3-Exchange Mutation

3-exchange mutation selects three different positions of a chromosome at random, say, r1, r2 and r3, and exchange the genes on these positions as follows: P(r1) ↔ P(r2) and then P(r2) ↔ P(r3), if they are different. If the cost of the mutated chromosome is bigger than the cost of the parent, we repeat another four times. If in any repetition mutated chromosome cost is less than the parent chromosome, we replace the parent chromosome, otherwise, skip.

2.4.2 Constructive Mutation

We first construct 'alphabet table' based on cost matrix C provided that the costs in a column are arranged in non-ascending order. Suppose p=a(1, j) and q= αj, then find the index t∈J={1,2,…,n} for which D=max{cpt - cqt} is the maximum. Then replace αt by a(1, t) provided that the capacity constraints are not violated. However, if the mutated chromosome cost is less than the parent chromosome, we replace the parent chromosome, otherwise, skip. For example, applying on the chromosome P: (1, 2, 4, 5, 1, 2, 4, 3, 1, 3, 5, 5, 5, 1, 3) with cost 291, one may obtain the offspring (1, 2, 3, 5, 1, 2, 4, 3, 1, 4, 2, 4, 5, 1, 3) with value 287, which is less than the parent chromosome.

3. Computational Results

For comparing the efficiency of the crossover operators, simple GAs using OPX (GA-OPX) and SCX (GA-SCX) have been encoded in Visual C++ and run on a Laptop with Intel(R) Core(TM) i3-3217 with CPU @ 1.80GHz and 4.00GB RAM under MS Windows 7. In the experiments, only four sets of benchmark instances (Gapa - Gapd) of size from 100 to 200, obtained from the OR library (Beasley, 1990) were used. Each set is composed of 6 instances, so, 24 instances in total were used. The experiments were performed twenty times for each instance. The solution quality is measured by the percentage of excess (%) of the obtained solution (SOL) over the best-known solution

(BKS) reported in the literature, as given by the formula: Excess (%) = 100*(SOL-BKS)/BKS. It is to be noted that GAs are controlled by some parameters, namely, population size, crossover probability, mutation probability, and termination criterion. However, to have a clear picture of crossover operators, we set 1.0 (i.e., 100%) as crossover probability, 0.10 (i.e., 10%) as mutation probability, 100 as population size, and a maximum of 100*n generations as a termination criterion.

Table 7: Summary of the results by the crossover operators for some benchmark instances

Instance	(m, n)	BKS	GA-OPX			GA-SCX		
			Best (%)	Avg (%)	Time	Best (%)	Avg (%)	Time
Gapa-1	(5, 100)	1698	9.95	13.88	2.75	0.00	0.09	1.32
Gapa-2	(5, 200)	3235	11.68	15.88	10.59	0.00	0.00	0.18
Gapa-3	(10, 100)	1360	18.31	24.24	2.87	0.07	0.18	2.82
Gapa-4	(10, 200)	2623	27.95	32.88	10.27	0.08	0.13	4.09
Gapa-5	(20, 100)	1158	27.55	32.38	3.03	0.00	0.27	3.80
Gapa-6	(20, 200)	2339	31.55	41.35	11.25	0.00	0.04	12.74
Gapb-1	(5, 100)	1843	5.37	10.46	2.25	0.54	2.16	3.31
Gapb-2	(5, 200)	3552	18.07	22.28	8.30	0.76	2.27	9.34
Gapb-3	(10, 100)	1407	2.49	4.21	2.33	0.14	1.68	3.84
Gapb-4	(10, 200)	2827	4.88	6.93	10.71	0.67	2.31	11.48
Gapb-5	(20, 100)	1166	8.83	12.14	3.16	0.86	2.24	7.33
Gapb-6	(20, 200)	2339	7.82	9.11	11.29	0.30	1.65	25.12
Gapc-1	(5, 100)	1931	1.35	3.67	2.60	0.83	1.63	3.00
Gapc-2	(5, 200)	3456	3.96	9.89	10.47	1.27	2.58	9.71
Gapc-3	(10, 100)	1402	5.06	10.23	2.84	0.50	1.68	3.30
Gapc-4	(10, 200)	2806	4.78	7.20	10.88	1.00	3.67	7.60
Gapc-5	(20, 100)	1243	11.02	16.15	3.37	0.80	3.10	5.02
Gapc-6	(20, 200)	2391	11.46	13.58	11.58	0.84	2.51	23.52
Gapd-1	(5, 100)	6353	8.03	8.97	1.82	2.09	3.32	4.08
Gapd-2	(5, 200)	12742	7.85	9.44	6.19	2.01	2.93	17.25
Gapd-3	(10, 100)	6348	8.76	9.49	2.34	0.74	2.30	6.99
Gapd-4	(10, 200)	12433	12.63	13.61	9.61	1.00	2.90	26.59
Gapd-5	(20, 100)	6192	10.52	11.40	3.14	1.40	3.27	9.97
Gapd-6	(20, 200)	12245	9.88	10.44	11.13	2.79	5.02	49.80
Average			**11.24**	**14.58**	**6.45**	**0.78**	**2.00**	**10.51**

Table 7 shows comparative study of the crossover operators for the instances. We report percentage of excess of best solution and average solution over BKS of 20 runs. We also report average computational time (in seconds) by the algorithms. In terms of solution quality, GA-SCX outperforms GA-OPX. However, GA-SCX takes little more time than GA-OPX. Overall, GA-SCX is found to the best for the tested instances.

Table 8: Comparative study among our HGAs, BEE and DE-SK

Instance	(m, n)	BKS	HGA1		HGA2		HGA3		BEE		DE-SK	
			Sol.	Time(a)	Sol.	Time(a)	Sol.	Time(a)	Sol.	Time(b)	Sol.	Time(c)
Gapa-1	(5, 100)	1698	1698	0.22	1698	0.26	1698	0.28	1698	0.18	1698	0.11
Gapa-2	(5, 200)	3235	3235	0.05	3235	0.05	3235	0.05	3235	0.95	3235	0.59
Gapa-3	(10, 100)	1360	1360	0.41	1360	0.41	1360	0.44	1360	0.27	1360	1.02
Gapa-4	(10, 200)	2623	2623	1.32	2623	1.38	2623	1.46	2623	1.31	2623	0.35
Gapa-5	(20, 100)	1158	1158	0.30	1158	0.31	1158	0.38	1158	0.41	1158	0.65
Gapa-6	(20, 200)	2339	2339	1.06	2339	1.12	2339	1.22	2339	1.81	2339	1.21
Gapb-1	(5, 100)	1843	1843	2.11	1843	2.20	1843	2.21	1843	5.97	1843	30.56
Gapb-2	(5, 200)	3552	3552	9.03	3552	9.89	3552	10.03	3552	45.99	3552	102.40
Gapb-3	(10, 100)	1407	1407	3.01	1407	3.55	1407	3.84	1407	0.36	1407	50.89
Gapb-4	(10, 200)	2827	2827	8.23	2827	9.86	2827	10.23	2827	315.04	2827	212.50
Gapb-5	(20, 100)	1166	1166	6.11	1166	6.23	1166	6.38	1166	1.12	1166	60.25
Gapb-6	(20, 200)	2339	2341	12.89	2339	15.12	2339	17.89	2339	28.65	2339	350.50
Gapc-1	(5, 100)	1931	1931	2.20	1931	3.06	1931	3.20	1931	3.61	1931	35.40
Gapc-2	(5, 200)	3456	3456	11.14	3456	14.32	3456	15.14	3456	18.09	3456	202.85
Gapc-3	(10, 100)	1402	1402	2.22	1402	2.45	1402	2.70	1402	5.81	1402	52.98
Gapc-4	(10, 200)	2806	2806	13.08	2806	13.11	2806	13.81	2806	488.89	2806	280.95

Gapc-5	(20, 100)	1243	1243	4.16	1243	5.06	1243	5.19	1243	23.16	1243	101.34
Gapc-6	(20, 200)	2391	2397*	19.24	2395*	22.36	2391	21.47	2392*	646.18	2391	420.46
Gapd-1	(5, 100)	6353	6353	8.10	6353	8.91	6353	9.10	6353	916.03	6353	60.89
Gapd-2	(5, 200)	12742	12744*	25.97	12744*	35.97	12744*	35.97	12744*	122.41	12744*	150.49
Gapd-3	(10, 100)	6348	6355*	12.03	6355*	13.03	6355*	13.03	6356*	538.29	6355*	104.50
Gapd-4	(10, 200)	12433	12459*	43.09	12442*	49.78	12442*	51.40	12442*	743.95	12442*	234.54
Gapd-5	(20, 100)	6192	6196*	12.89	6196*	13.47	6196*	14.70	6221*	1704.46	6196*	209.45
Gapd-6	(20, 200)	12245	12288*	53.16	12282*	66.33	12276*	61.21	12276*	922.94	12276*	398.67
Average Time				**10.50**		**12.43**		**12.56**		**272.33**		**127.65**
No. of Optimal				**18/24**		**18/24**		**19/24**		**18/24**		**19/24**
%age of Optimal				**75.00%**		**75.00%**		**79.00%**		**75.00%**		**79.00%**

(a) Run on Intel(R) Core(TM) i3-3217U with 1.8 GHz CPU and 4.00GB RAM
(b) Run on Intel Pentium CoreDuo PC with 1.6 GHZ CPU and 512 MB RAM.
(c) Computer specification is not reported.
The solution quality of our proposed GA (i.e., GA-SCX) is enhanced by hybridizing with different local searches, and then the best solutions are compared to the optimal solution. For these comparisons, GA with 3-exchange mutation (HGA1), constructive mutation (HGA2) and combined constructive 3-exchange mutation (HGA3) local search techniques are tested and reported in Table 8. The HGAs are compared using two performance measures: (1) solution quality, and (2) computational time. Among the HGAs, HGA2 is found to be better than HGA1, and HGA3 is the best. However, HGA3 takes little more computational times than other two HGAs. We also compare our HGAs with other heuristics found in the literature, namely, BEE (Özbakir et al., 2010), and DE-SK (Sethanan and Pitakaso, 2016). The computation results are shown in the same Table 8. From this table, HGA3 and DE-SK yield the optimal cost 19 out of 24 instances (79.00% optimal); HGA1, HGA2 and BEE yield the optimal cost 18 out of 24 instances (75.00% optimal). In terms of solution quality, HGA3 and DE-SK algorithm are found to be best. Now, in terms of computational time, though the algorithms were run on different machines, still by looking at the machine specification as well as computational time, one can decide that HGA1 is the best. Overall, in terms of solution quality as well as computational time, our algorithm HGA3 is found to the best for the small to medium instances.

4. Conclusion and Discussion

In this study, hybrid genetic algorithms were developed to optimize the sequence of jobs assigned to agents with the minimum assignment costs. For that we have modified the sequential constructive crossover (SCX) to propose a simple genetic algorithm (GA) for solving the generalized assignment problem (GAP). We presented a comparative study between SCX and one-point crossover (OPX) for some benchmark GAP instances. In terms of quality of the solution, SCX is found to be far better than the OPX. The GA was used in the combination of two local techniques - 3-exchange and constructive mutation algorithms. Three hybrid GAs are presented and each of them uses GA with a different local search technique. From the comparative study among the proposed algorithms, we found that GA with combined 3-exchange and constructive mutation (HGA3) is the best. Moreover, our proposed algorithms were used to compare their performance with those of BEE (Özbakir et al., 2010), and DE-SK (Sethanan and Pitakaso, 2016) algorithms as the existing state-of-art heuristics found in the literature. The computational results reveal that our HGA3 is the best for the small to medium instances. This is because the HGA3 is designed to enhance the search capability by improving the diversification using the GA's operators and combined constructive and 3-exchange mutations can improve the intensification. Though HGA3 is found to be the best algorithm, however, for some instances, it could not find best known solution within twenty runs. Hence, incorporating a better local search method may further improve the solution quality and hence, may obtain best known solutions for the remaining instances also, which is under our investigation.

Acknowledgements

This research was supported by the Deanery of Academic Research, Al Imam Mohammad Ibn Saud Islamic University, Saudi Arabia vide Grant No. 350902. The author thanks the Deanery for the financial support. The author also thanks the anonymous honourable reviewers for their constructive comments and suggestions.

References

[1] Ahmed ZH. (2010a). Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator, International Journal of Biometrics & Bioinformatics, 3, 96-105.
[2] Ahmed ZH. (2010b). A lexisearch algorithm for the bottleneck traveling salesman problem, Int Journal of Computer Science & Security, 3, 569-577.
[3] Ahmed ZH. (2013a). A hybrid genetic algorithm for the bottleneck traveling salesman problem, ACM Transactions on Embedded Computing Systems, 12, Art. No. 9.

[4] Ahmed Z H. (2013b). An experimental study of a hybrid genetic algorithm for the maximum travelling salesman problem, Mathematical Sciences, 7, 1-7.
[5] Ahmed ZH. (2014a). The ordered clustered travelling salesman problem: A hybrid genetic algorithm, The Scientific World Journal, Art ID 258207, 13 pages. doi:10.1155/2014/258207.
[6] Ahmed ZH. (2014b). A simple genetic algorithm using sequential constructive crossover for the quadratic assignment problem, Journal of Scientific & Industrial Research, 73, 763-766.
[7] Ahmed ZH. (2016). Experimental analysis of crossover and mutation operators for the quadratic assignment problem, Annals of Operations Research, 247, 833-851.
[8] Ahmed ZH. (2018). The Minimum Latency Problem: A Hybrid Genetic Algorithm, IJCSNS International Journal of Computer Science and Network Security, 18(11), 153-158.
[9] Ahmed ZH. (2019). Algorithms for the Quadratic Assignment Problem, LAP LAMBERT Academic Publishing, Mauritius.
[10] Al-Omeer MA & Ahmed ZH. (2019). Comparative study of crossover operators for the MTSP, 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 3-4 April 2019, 1-6.
[11] Beasley JE. (1990). OR-library: distributing test problems by electronic mail, Journal of the Operational Research Society, 41, 1069–1072.
[12] Bennaceur H & Ahmed ZH. (2017). Frequency Model Based Crossover Operators for Genetic Algorithms Applied to the Quadratic Assignment Problem, International Arab Journal of Information Technology, 14, 138-145.
[13] Deb K. (1995). Optimization for Engineering Design: Algorithms and Examples, Prentice Hall of India Pvt Ltd, New Delhi, India.
[14] Catrysse DG & Van Wassenhove LN. (1992). A survey of algorithms for the Generalized Assignment Problem, European Journal of Operational Research, 60, 260-272.
[15] Chu PCH & Beasley JE. (1997). A Genetic Algorithm for the Generalised Assignment Problem, Computers & Operations Research, 24, 17-23.
[16] Feltl H, Unther G & Raidl R. (2004). An Improved Hybrid Genetic Algorithm for the Generalised Assignment Problem, Symposium on Applied Computing, Session: ECO, 990-995.
[17] Goldberg DE. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, New York.
[18] Juell P, Perera A & Nygard K. (2003). Application of a Genetic Algorithm to Improve an Existing Solution for the General Assignment Problem, Proceedings of the 16th International Conference on Computer Applications in Industry and Engineering, Lasvegas.
[19] Narciso MG & Lorena LAN. (1999). Lagrangean/surrogate Relaxation for Generalized Assignment Problems, European Journal of Operational Research, 114, 165-177.
[20] Özbakir L, Baykasoğlu A & Tapkan P. (2010). Bees algorithm for generalized assignment problem, Applied Mathematics and Computation, 215, 3782–3795.
[21] Ross GT & Soland RM. (1975). A branch and bound algorithm for the generalized assignment problem, Mathematical Programming, 8, 91–103.
[22] Sethanan K & Pitakaso R. (2016). Improved differential evolution algorithms for solving generalized assignment problem, Expert Systems with Applications, 45, 450–459.
[23] Yagiura M, Ibaraki T & Glover F. (2006). A path relinking approach with ejection chains for the generalized assignment problem, European Journal of Operational Research, 169, 548–569.

Zakir Hussain Ahmed is a Full Professor in the Department of Computer Science at Al Imam Mohammad Ibn Saud Islamic University, Riyadh, Kingdom of Saudi Arabia. He obtained MSc in Mathematics (Gold Medalist), Diploma in Computer Application, MTech in Information Technology and PhD in Mathematical Sciences (Artificial Intelligence/Combinatorial Optimization) from Tezpur University (Central), Assam, India. Before joining the current position, he served in Tezpur University, Sikkim Manipal Institute of Technology, Asansol Engineering College and Jaypee Institute of Engineering and Technology, India. His research interests include artificial intelligence, combinatorial optimization, digital image processing and pattern recognition. He has several publications in the fields of artificial intelligence, combinatorial optimization and image processing.