Expert Systems With Applications 45 (2016) 450–459

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Improved differential evolution algorithms for solving generalized assignment problem

Kanchana Sethanan [a,*], Rapeepan Pitakaso [b,1]

[a] Research Unit on System Modeling for Industry, Department of Industrial Engineering, Faculty of Engineering, Khon Kaen University, Khon Kaen 40002, Thailand
[b] Metaheuristics for Logistic Optimization Laboratory, Department of Industrial Engineering, Faculty of Engineering, Ubon Ratchathani University, Thailand

ARTICLE INFO

Keywords:
Generalized assignment problem
Differential evolution algorithm
Shifting procedure
Exchange procedure

ABSTRACT

This paper presents algorithms based on differential evolution (DE) to solve the generalized assignment problem (GAP) with the objective to minimize the assignment cost under the limitation of the agent capacity. Three local search techniques: shifting, exchange, and *k*-variable move algorithms are added to the DE algorithm in order to improve the solutions. Eight DE-based algorithms are presented, each of which uses DE with a different combination of local search techniques. The experiments are carried out using published standard instances from the literature. The best proposed algorithm using shifting and *k*-variable move as the local search (DE-SK) techniques was used to compare its performance with those of Bee algorithm (BEE) and Tabu search algorithm (TABU). The computational results revealed that the BEE and DE-SK are not significantly different while the DE-SK outperforms the TABU algorithm. However, even though the statistical test shows that DE-SK is not significantly different compared with the BEE algorithm, the DE-SK is able to obtain more optimal solutions (87.5%) compared to the BEE algorithm that can obtain only 12.5% optimal solutions. This is because the DE-SK is designed to enhance the search capability by improving the diversification using the DE's operators and the *k*-variable moves added to the DE can improve the intensification. Hence, the proposed algorithms, especially the DE-SK, can be used to solve various practical cases of GAP and other combinatorial optimization problems by enhancing the solution quality, while still maintaining fast computational time.

1. Introduction

The generalized assignment problem (GAP) is the problem of assigning *n* different tasks to *m* agents with the objective of minimizing the total assignment cost. GAP was first proposed by Ross and Soland (1975). In the GAP, each task is assigned to only one agent representing a single resource with limited capacity and each task requires a certain amount of a resource used to execute that task. GAP is known to be a Non-Polynomial or NP-hard problem (Fisher & Jaikumar, 1981) and has a wide variety of applications in the real world. As an example, we can find the combination of generalized assignment concepts in problems arising in the vehicle routing problem, traveling salesman problem, location allocation problems, and scheduling problem. Motivated by real life applications, a variety of generalizations of GAP have been proposed, for example, the multi-objective GAP by Subtil, Carrano, Souza, and Takahashi (2010), the dynamic multi-resource generalized assignment problem by Shtub and Kogan (1998), the multi-resource generalized assignment problem with additional constraints by Privault and Herault (1998), the fuzzy assignment problems by Li, Li, Jin, and Wang (2012a), the fuzzy multiple objective generalized assignment problems by Tapkan, ÖZbakıR, and BaykasoğLu (2013), the equilibrium generalized assignment problem by Liu et al. (2012) and the generalized assignment problem with minimum quantities by Krumke and Thielen (2013). Other generalizations of GAP can be found in Cattrysse and Van Wassenhove (1992) and Wang (2002).

To solve the GAP, many different solution methods have been developed by researchers in the past decades. Several exact methods such as branch and price, cutting plane, branch and cut, and branch and bound algorithms have been proposed to solve the problem for its optimal solution (Avella, Boccia, & Vasilyev, 2010; Fisher, Jaikumar, & Van Wassenhove, 1986; Nauss, 2003; Ross & Soland, 1975; Savelsbergh, 1997). However, due to the complexity of the problem, exact methods developed for GAP can solve only small sized instances of up to 200 tasks and 20 agents (Lui et al., 2012). Therefore, efficient approximation methods, i.e., heuristics and metaheursitics, are needed to solve the problem in recent years, especially for large sized problems. Heuristics and metaheursitics have been intensively studied to discover the effective algorithm for solving a large scale GAP. Hence, in this paper, a well-known metaheuristic called the differential evolution algorithm (DE), which is one of the

* Corresponding author. Tel.: +66 43 362299; fax: +66 43 362299,+6643 472735.
E-mail addresses: ksethanan@gmail.com, skanch@kku.ac.th (K. Sethanan), enrapepi@ubu.ac.th (R. Pitakaso).
[1] Tel.: +66 4535 3352; fax: +66 45 353333.

http://dx.doi.org/10.1016/j.eswa.2015.10.009
0957-4174/

evolutionary algorithms, is used to solve the problem and enhance the solution obtained in tractable time. DE was first proposed by Storn and Price (1997) and it has continuously proven to be an effective algorithm to solve both continuous and discrete optimization (Brest, Greiner, Boskovic, Mernik, & Zumer, 2006; Gao, Zhou, Li, Pan, & Yi, 2015; Gong, Cai, & Jiang, 2008; Gong, Cai, & Ling, 2010; Nearchou, 2005, 2007, 2008; Qin, Huang, & Suganthan, 2009). Recently, there have been many researchers expanding the use of DE to solve various combinatorial optimization problems such as the vehicle routing problem, assembly line balancing, vehicle routing with time windows and production flow shop and job shop scheduling (see Cao and Lai, 2010; Dechampai, Tanwanichkul, Sethanan, & Pitakaso, 2015; Fan &Yan, 2015; Mokhtari & Salmasnia, 2015; Nearchou, 2005, 2007, 2008; Onwubolu & Davendra, 2006; Pan, Tasgetiren, & Liang, 2008; Wisittipanich & Kachitvichyanukul, 2011, 2012).

With the success of DE applications in many combinatorial optimization problems, DE was adopted to solve the GAP in this paper. Originally, DE was used to solve many kinds of problems without local search in order to maintain low computational time. To improve the solution for the GAP while still maintaining fast computational time to solve the problem, three local search techniques, shifting, SWAP, and *k*-variable move algorithms, were incorporated into the DE mechanism. The shifting and SWAP algorithms were found in the literature, while the *k*-variable move algorithm was first designed in this paper in an attempt to obtain lower computational time. The performance of the modified DE on this problem might be a good indicator for its ability to solve the GAP and also the complex constrained combinatorial optimization problems. Therefore, in order to show the competitiveness of the proposed approach, it is benchmarked with the well-known heuristics on the problem, BEE (Özbakir, Baykasoğlu, & Tapkan, 2010) and Tabu (Diaz & Fernandez, 2001), that are found in the literature. In view of these shortcomings, our main contributions in this study are not only developing the new local search technique (the *k*-variable move) but also hybridizing all three local search techniques (shifting algorithm, SWAP algorithm, and *k*-variable move algorithm) in the DE algorithms for the GAP with the possibility of keeping low computational time. The remainder of this paper is organized as follows: Section 2 reviews the literature; Section 3 presents the mathematical formulation of the GAP; Section 4 exhibits the proposed heuristic; the computational results are discussed in Section 5, and Section 6 is the conclusion and discussion part of the paper.

2. Related literature

The GAP is a well-known problem. It is one of the widely applied and extensively studied combinatorial optimization problems. Many real life applications can be modelled as a GAP. As a consequence, there has been an increase in GAP research motivated by practical applications. To solve the GAP, various approaches can be found. Exact algorithms have been developed to solve small sized problems. Since the GAP has been shown to be NP-hard, in an effort to find a near optimal solution for the larger, more practical problems, efficient heuristic algorithms including evolutionary algorithms are required for its solution.

The heuristics methods that have been proposed in the literature are generally composed of two steps: (1) construct a set of initial solutions, which can be feasible or infeasible solutions and (2) improve the initial solutions to find better solutions. To construct a set of initial solutions, Lagrangian relaxation is one of the popular techniques widely used (Barcia & Jörnsten, 1990; Jeet & Kutanoglu, 2007; Jörnsten & Näsberge, 1986; Litvinchev, Mata, Rangel, & Saucedo, 2010; Lorena & Narciso, 1996; Narciso & Lorena, 1999; Park, Lim, & Lee, 1998). The construction of the initial solutions can also be built randomly (Diaz & Fernandez, 2001; Liu et al., 2012; Munoz & Munoz, 2012) or by using greedy heuristic information (Munoz & Munoz, 2012; Özbakir et al., 2010). To improve the initial solutions, various heuristics and metaheuristics have been used such as the Genetic algorithm (GA), Tabu Seach (TS), the dynamic Tabu Search algorithm (DTS), Simulated Annealing (SA), Bees algorithm (BA), the Variable Depth Search algorithm, a set partitioning algorithm, the Greedy Randomized Adaptive Search, simple iterative search, and the Cross Entropy algorithm (Amini & Racer, 1994; Cattrysse, Salomon, & Van Wassenhove, 1994; Chu & Beasley, 1997; Diaz & Fernandez, 2001; Higgins, 2001; Liu et al., 2012; Laguna, Kelly, González-Velarde, & Glover, 1995; Shmoys & Tardos, 1993; Narciso & Lorena, 1999; Osman,1995; Özbakir et al., 2010; Romeijn & Romero-Morales, 2000; Trick, 1992; Yagiura, Yamaguchi, & Ibaraki, 1998; Yagiura, Ibaraki, & Glover, 2004 , 2006).

In the past decade, metaheuristic approaches have been applied extensively in various different combinatorial optimization problems, especially for complex and large-scale problems. Recently, there has been significant effort to apply evolutionary computation techniques for combinatorial optimization problems. Among recent evolutionary algorithms (EA) for GAP are those such as TS and SA (see Osman, 1995), GA (see Wilson, 1997; Chu & Beasley, 1997), and the BEE algorithm (see Özbakir et al., 2010).

The DE is one of the most powerful and interesting evolutionary algorithms and is effectively applied to both continuous and discrete optimization problems. DE like the method of GA permits each successive generation of solutions to develop from the previous generations. However, the main difference between the DE and GA is the selection process and the mutation scheme that makes DE self-adaptive. In DE, all solutions have the same opportunity of being selected as parents with independence of their fitness value. DE employs a selection process that the better one of new solution and its parent wins the competition that provides great advantage of converging performance over GA (Hegerty, Hung, & Kasprak, 2009). The application of DE can be found in many research studies. Generally, the classical DE consists of 4 common steps: (1) generate initial solution, (2) perform mutation process, (3) perform recombination process, and (4) perform selection process. However, the convergence of the traditional DE is strongly dependent on the choice of three main control parameters: the mutation scale factor *F*, the crossover constant *Cr*, and the population size *NP*, resulting in easy falling into a local search and slow convergence speed. Choosing proper mutation strategies and control parameters is one of crucial approaches to enhance the performance of DE algorithm. Hence, in 2015, Yan and Fan proposed a self-adaptive DE algorithm with discrete mutation control parameters (DMPSADE) to solve the optimization problem. To assess the performance of the DMPSADE, it was compared with 3 non-DE algorithms and 8 state-of-the-art DE variants by using 25 benchmark functions. The statistical results show that the average performance of DMPSADE is better than that of all other previous studies. For the application of DE in solving the GAP, there have been very few studies, for example, for DE solving the GAP, see Tasgetiren, Suganthan, Chua, and Al-Hajri (2009) and a novel discrete DE (DDE) algorithm for the multi-objective GAP, see Jiang, Xia, Chen, Meng, and He (2013).

Recently, a hybrid approach such as a local search technique is another approach that has become one of the most powerful techniques developed to obtain a better solution than the work applying the classical metaheuristics (Sangsawang, Sethanan, Fujimoto, & Gen, 2015). It can be incorporated into a metaheuristic mechanism for obtaining a better solution to the problem. Neighbourhood search algorithm is one of the efficient local search techniques that can be hybridized with heuristic and metaheuristic algorithms. Generally, the neighbourhood search algorithm can be distinguished into 3 types which are: (1) SWAP algorithm (Diaz & Fernandez, 2001), (2) shifting algorithm (Diaz & Fernandez, 2001; Munoz & Munoz, 2012), and (3) ejection chain neighbourhood (Yagiura et al., 2004; Yagiura et al., 2006). For the SWAP algorithm, if task i is assigned to agent j and task $i+1$ is assigned to agent $j+1$, the exchange procedure is just the

algorithm that re-assigns task i to agent $j+1$ and task $i+1$ to agent j. For the shifting algorithm, task $i+1$ will be assigned to agent j when task i is still with agent j. The shifting and SWAP algorithm can be executed only when the following two conditions are met: (1) the capacity of agent j and $j+1$ is not violated and (2) the cost of moving a task to an agent reduces the assignment cost. The ejection chain algorithm is the SWAP algorithm which attempts to move tasks i from agent j and then move task $i+1$ from agent $j+1$ to agent j, then task i must move to another agent so that task i can move into it. The moving continuously executes until no improvement can be found. The new assigning can be done only when the capacity constraint is not violated.

The hybrid approaches to solve complex and large-scale problemscan be efficiently used such as vehicle routing, traveling salesman, location allocation, and scheduling. For example, to solve the scheduling problem, a hybrid discrete differential evolution (HDDE) by combining the presented discrete version of the DE and inserting a neighborhood based local search procedure was developed to solve blocking flow shop scheduling problems. It was shown that the HDDE algorithm outperformed the TS approach (see Wang, Pan, Suganthan, Wang, & Wang, 2010). Recently, the hybrid genetic algorithm (HGA) and hybrid particle swarm optimization (HPSO) were developed to solve two-stage reentrant flexible flow shop with blocking constraint with the objective to minimize makespan. The results reveal that the HGA and the HPSO algorithms can improve the makespan solutions by an average of 7.92% and 7.27% compared to those of the traditional GA and particle swarm optimization (PSO), respectively (see Sangsawang et al., 2015). In the same year, the GA hybridized ant colony optimization (GACO) was developed to solve the reentrant hybrid flow shop with time window constraints. The results show that the GACO is very effective and can solve problems optimally with reasonable computational effort (see Chamnanlor, Sethanan, Gen, & Chien, 2015). In the field of transportation, a hybrid optimization algorithm (HOA), which is based on a combination of DE and GA, was investigated by Erbao and Lai (2009). It was found that both DE and GA could effectively use in optimizing large scale problems.

Although the hybrid approaches have been effectively used in a variety of fields, their uses to solve the GAP are quite limited. For example, the fuzzy logic or Markov chain theory in connection with the bee algorithm and GA are proposed to solve real world GAP with uncertain systems (see Fu, Diabat, & Tsai, 2014; Li et al., 2012a; Li, Li, Jin, & Wang, 2012b; Tapkan et al., 2013). It can be seen that most previous research has focused on developing the hybrid approach to solve complex and large-scale problems. Therefore, in this paper, efficient local search techniques (i.e., shifting algorithm, SWAP algorithm, and the k-variable move algorithm) are hybridized with DE in order to enhance the solution quality for the problem. The k-variable move algorithm which employs the idea of the SWAP algorithm is developed, while the shifting and SWAP algorithms were found in the literature.

3. Mathematical model for the generalized assignment problem

This section presents a 0–1 mixed integer programming model with the objective to minimize the assignment cost. The model is presented below with a brief explanation of each constraint. Parameters and decision variables used in the model are defined as follows:

Indices

i index of task i ; $i = 1 \ldots n$

j index of agent j ; $j = 1 \ldots m$

Parameters

C_{ij} cost of assigning task i to agent j

R_{ij} resource usage when assigning task i to agent j

b_j capacity of agent j

Decision variables

$Y_{ij} = 1$ if task i is assigned to agent j, Otherwise 0

Objective function

$$\text{minimize} \quad \sum_{j=1}^{m}\sum_{i=1}^{n} Y_{ij}C_{ij} \tag{1}$$

Subject to

$$\sum_{j=1}^{m} Y_{ij} = 1 \quad \forall i = 1 \ldots n \tag{2}$$

$$\sum_{i=1}^{n} Y_{ij}R_{ij} \le b_j \quad \forall j = 1 \ldots m \tag{3}$$

$$Y_{ij} \in \{0, 1\} \quad \forall i = 1 \ldots n \text{ and } j = 1 \ldots m \tag{4}$$

Eq. (1) is an objective function which aims to minimize the total cost of the assignment of task i to agent j. Constraint (2) ensures that task i will be assigned to exactly one agent and Constraint (3) controls that the total resource used in assigning all tasks to agent j will not exceed its capacity. Constraint (4) ensures that Y_{ij} is a binary variable.

4. DE for the generalized assignment problem

DE was first used in the area of continuous optimization and there are many successful applications that use DE to solve a combinatorial optimization. In these applications, a key task is to design procedures to transform real numbers which are the results from DE operators (mutation, recombination selection process) into the solution to the problem. These procedures normally are called encoding and decoding procedures. The following section explains the encoding and decoding procedure of the proposed algorithm.

4.1. Encoding and decoding method of DE

Consider a problem to assign 15 tasks to 5 agents with the cost of assigning task i to agent j as shown in Table 1. The numbers shown in this table are the costs of assigning task i to agent j (C_{ij}). For example, the cost of assigning task 1 to agent 3 and 4 are 16 and 19 cost units, respectively. In assigning task i to agent j, it consumes a specified amount of resource as shown in Table 2. For example, resource consumptions for assigning task 9 to agents 4 and 5 are 19 and 16 resource units, respectively.

Normally, each agent has a limited capacity. In this example, the maximum capacities for agents 1, 2, 3, 4 and 5 are 36, 34, 38, 27 and 33 units, respectively. To optimize the sequence of the tasks assigned to the agents, a vector encoding of DE to represent the problem is designed with the dimension $1 \times N$, where N is the number of tasks. An example of a vector used in the proposed algorithm is shown in Fig. 1. From Fig. 1, the numbers in the first row represent task labels. The numbers in the second row are randomly generated in the first iteration of the proposed algorithm. In later iterations, these numbers will be obtained from the DE operators (mutation and recombination process). To decode the vector shown in Fig. 1, the solutions of the GAP can be obtained by using the rank position value (ROV) which is the procedure used to sort the position values of a vector in ascending order. From the ranking of the values in the position value, the sequence of the tasks is obtained. The result of ROV is shown in Fig. 2. From Fig. 2, after applying ROV, the order Ω of this vector is {3, 11, 10, 8, 4, 6, 1, 7, 9, 12, 14, 15, 13, 2, 5}. The order Ω will be used to assign the tasks to the agents according to the procedure which is described next.

The initial solution phase can be generated by using the following procedure.

(1) Randomly select the order of the items that are being assigned to a set of agents and name it as order Ω.

Table 1
Cost of assigning task *i* to agent *j*.

j	*i*														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	17	21	22	18	24	15	20	18	19	18	16	22	23	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
The resource usage of assigning task *i* to agent *j*.

j	*i*														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
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

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.60	0.85	0.01	0.25	0.96	0.26	0.60	0.10	0.62	0.07	0.02	0.67	0.76	0.73	0.74

Fig. 1. Example of a vector used to represent the sequence of the items.

3	11	10	8	4	6	1	7	9	12	14	15	13	2	5
0.01	0.02	0.07	0.10	0.25	0.26	0.60	0.60	0.62	0.67	0.73	0.74	0.76	0.85	0.96

Fig. 2. sequence of the items after using ROV.

(2) Assign all items to exactly one agent. The item (i.e., item *i*) will be assigned to the agent with the least resource usage. If the agent does not have enough capacity to process that item, item *i* must be assigned to the agent with the second least (or more) resource usage.
(3) Continue assigning items until all items are assigned to exactly one agent.

For example, if order Ω is {5, 4, 1, 7, 8, 9, 12, 15, 14, 13, 3, 10, 2, 6, 11}, the first assignment is to assign task 5 to agent 1 since this agent takes the lowest time (i.e., 8 units) to execute task 5 among all 5 agents (agents 8, 11, 24, 9, and 10). After executing task 5 to agent 1, the currently remaining capacity of this agent is 24 units ($36 - 8 = 24$ units). The next assignment is to assign task 4 to one of the agents 1, 2, 3, 4 and 5. From the procedure, this task is assigned to agent 5 since its executing time on this agent is the least (i.e., 13 units). The assignment procedure is repeated until all the tasks have been assigned. In this example the tasks in order Ω are assigned without any capacity violation until task 14 is assigned. Since task 14 requires 8 units of resource, it is assigned to agent 1. Unfortunately, the remaining capacity of agent 1 is only 7 units. Thus task 14 will be assigned to the agent with the second lowest resource usage. From this example, task 14 is assigned to agent 4 (with the remaining available resource of15 units) and requires 9 units of this resource for executing. Hence, the remaining available resource of agent 4 is 6 units (15−9).

Finally, the complete assignment of the tasks to the agents is (task↔agent) {5↔1, 4↔5, 1↔1, 7↔4,8↔3,9↔1,12↔4,15↔3, 14↔4,13↔5,3↔3,10↔2, 2↔2,6↔2,11↔5}. This assignment yields the total assignment cost of 295 units. For another example, if we have Ω as {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15} to assign to the same agents 1, 2, 3, 4, and 5 by using the same procedure discussed above, the result of the complete assignment is {1↔1, 2↔2,3↔3, 4↔5, 5↔1, 6↔4, 7↔4, 8 ↔3, 9↔1, 10↔4, 11↔4, 12↔5, 13↔5, 14↔1, 15↔3} and this assignment yields the total assignment cost of 289 units. It is clear that order Ω plays an important role in determining assignment cost. In this paper, the DE will be applied to find the best order Ω with the lowest cost.

The vector shown in vector one is an example of the vector used in the proposed heuristics. Let *NP* denote the number of vectors generated in each iteration of the DE mechanism. *NP* similar vectors will be randomly generated and the DE mechanism applied to obtain the different vectors leading to a different GAP result. The next section explains the general procedures of DE used in the proposed heuristics.

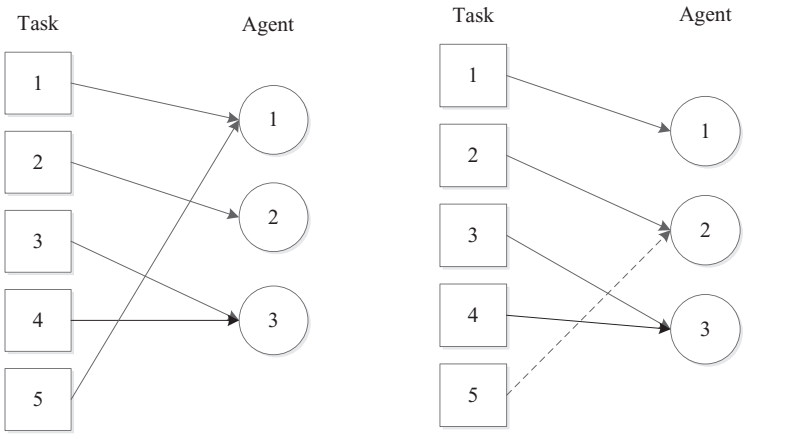
4.2. Generate a set of initial vectors (target vectors)

If *NP* is the population size or the number of vectors that will be used in each iteration of DE, *NP* vectors will be randomly generated in the first iteration and each vector is similar to the vector shown in Fig. 1. Each vector will be transformed into mutant vector and trial vector using mutation and recombination process.

4.3. Mutation process

NP Mutant vectors ($V_{i,j,G}$) will be generated using Eq. (5). Three *NP* Mutant vectors ($V_{i,j,G}$) will be generated using Eq. (5). Three target vectors are randomly selected (r_1, r_2 and r_3) and Eq. (5) applied into these selected vectors. *F* is a predefined scaling factor. In this paper, it is set to 2.0 (Qin et al., 2009).

$$V_{i,j,G} = X_{r1,j,G} + F(X_{r2,j,G} - X_{r3,j,G}) \quad (5)$$



a) Result before applying shifting algorithm b) Result after applying shifting algorithm

Fig. 3. Example of shifting algorithm.

4.4. Recombination process

NP Trial vectors ($V_{i,j,G}$) are the product of the recombination process. A trial vector is obtained from Eq. (6). A position's value of a vector can be a position's value of a target or trial vector depending on the random number which is generated for that position and the value of *CR* (predefined parameter, in this article, it is set to be 0.8 (Qin et al., 2009)). The position's value in a specified position will be copied from the position's value of that position in the target vector if the random number for that position is higher than the CR value. On the other hand, if the random number generated for that position is lower than that of *CR*, the position's value in that position will be equal to the value of the mutant vector. The formula that represents this mechanism is Eq. (6).

$$U_{i,j,G} = \begin{cases} V_{i,j,G} \text{ if } rand_{i,j} \le CR \text{ or } j = Irand \\ X_{i,j,G} \text{ if } rand_{i,j} > CR \text{ or } j \ne Irand \end{cases} \quad (6)$$

where $rand_{i,j}$ is a random number generated for vector i at position j and *Irand* is the random position randomly picked which has the value between 1 and D, where D is the dimension of the vector (number of tasks). The position designated by *Irand* will use the value of the mutant vector as the value in that position of the trial vector.

4.5. Selection process

Eq. (7) is used to select the next generation of the target vector ($X_{i,j,G+1}$). The target vector ($X_{i,j,G}$) or trial vector ($U_{i,j,G}$) that generates a better solution will be selected as the target vector of the next generation ($X_{i,j,G+1}$). The selection formula is shown in Eq. (7).

$$X_{i,j,G+1} = \begin{cases} U_{i,j,G} & \text{if } f(U_{i,j,G}) \le f(X_{i,j,G}) \\ X_{i,j,G} & \text{otherwise} \end{cases} \quad (7)$$

4.6. The local search

The construction of the solution of GAP already explained in Section 3 can be significantly improved by the use of the local search. Three types of local search were proposed which are: (1) shifting algorithm, (2) SWAP algorithm and (3) *k*-variable move. The first two algorithms are taken from Diaz and Fernandez (2001) and the last one (i.e., *k*-variable move) is developed in order to enhance the search capability of the proposed algorithm.

4.6.1. Shifting algorithm

The shifting algorithm is the improvement heuristic attempting to insert a task to an agent in order to generate better solution. Suppose task i is currently assigned to agent j. In the shifting algorithm, task i will be shifted to agent j' that has lower cost than that of agent j. For example, currently task 5 is assigned to agent 1 (cost is 24 units). The algorithm will shift task 5 to agent 2 since agent 2 yields lower assignment cost (i.e., 17 units). However, the capacity of agent 2 must be enough to add task 5 as its client. An example of the shifting algorithm is shown in Fig. 3. Fig. 3(a) shows the result of the assigning procedure when there are 5 tasks and three agents. Tasks 1, 2, 3, 4 and 5 are assigned to agent 1, 2, 3, 3, and 1, respectively. After applying the shifting procedure as explained above, task 5 will be shifted to agent 2 instead of agent 1 as shown in Fig. 3(b). It is noted that, in the shifting algorithm, the re-assignment will be done according to order Ω.

4.6.2. SWAP algorithm

The SWAP algorithm is the improvement heuristic attempting to exchange the agents of two tasks. Suppose task i is currently assigned to agent j and task i' is assigned to agent j'. Task i will be re-assigned to agent j' and task i' is assigned to agent j. The re-assignment will be executed only when the SWAP generates better assignment cost. For example, currently task 2 is assigned to agent 2 and task 4 is assigned to agent 3. The algorithm will re-assign task 2 to agent 3 and re-assign task 4 to agent 2. However, the SWAP will be accepted to move only when (1) both agents have enough capacity for the move and (2) the cost generated by the new move is lower than that of the current assignment. An example of the SWAP algorithm is shown in Fig. 4(a) and (b). The order of re-assignment will be done according to order Ω.

4.6.3. k-variable move algorithm

The *k*-variables move algorithm is an extended version of the SWAP algorithm. Instead of selecting 2 pairs of tasks and agents to be exchanged, *k* pairs will be selected and moved. The tasks will be moved from one agent to another by considering the cost saving and the remaining capacity of the agents. The algorithm can be explained as follows. Suppose *k* is set to be 3, currently task 1 is assigned to agent 1, task 2 is assigned to agent 2, and task 3 is assigned to agent 3. Five more possible sets of results from the re-assignment process are: (1) {1↔1, 2↔3, 3↔2}, (2) { 1↔3, 2↔1, 3↔2 }, (3) {1↔2, 2↔3, 3↔1}, (4) {1↔2, 2↔1, 3↔3} and (5) {1↔3, 2↔2, 3↔1}. The best assignment will be selected as the result of *k*-variable move. The move will be allowed only if the assignment of the task does not violate the capacity limitation of the agent. Examples of all possible results are illustrated in Fig. 5(a)–(e).

From Fig. 5, one of the possible moves having the best assignment cost (depending on whether it is a maximization or minimization

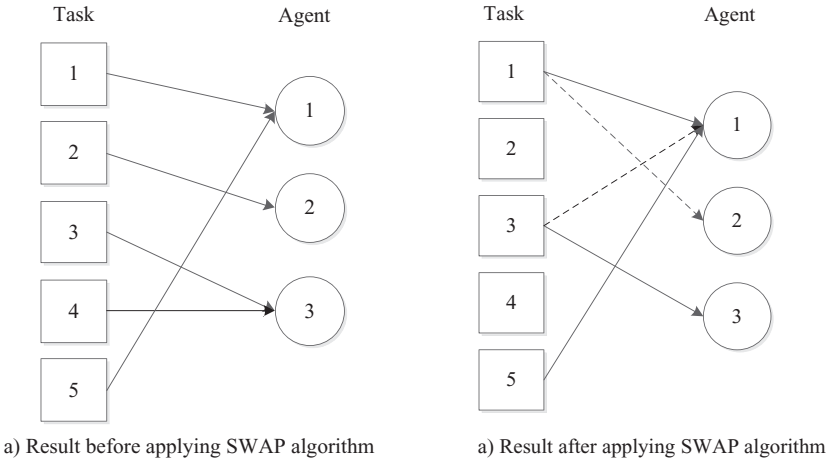


Fig. 4. Example of SWAP algorithm.

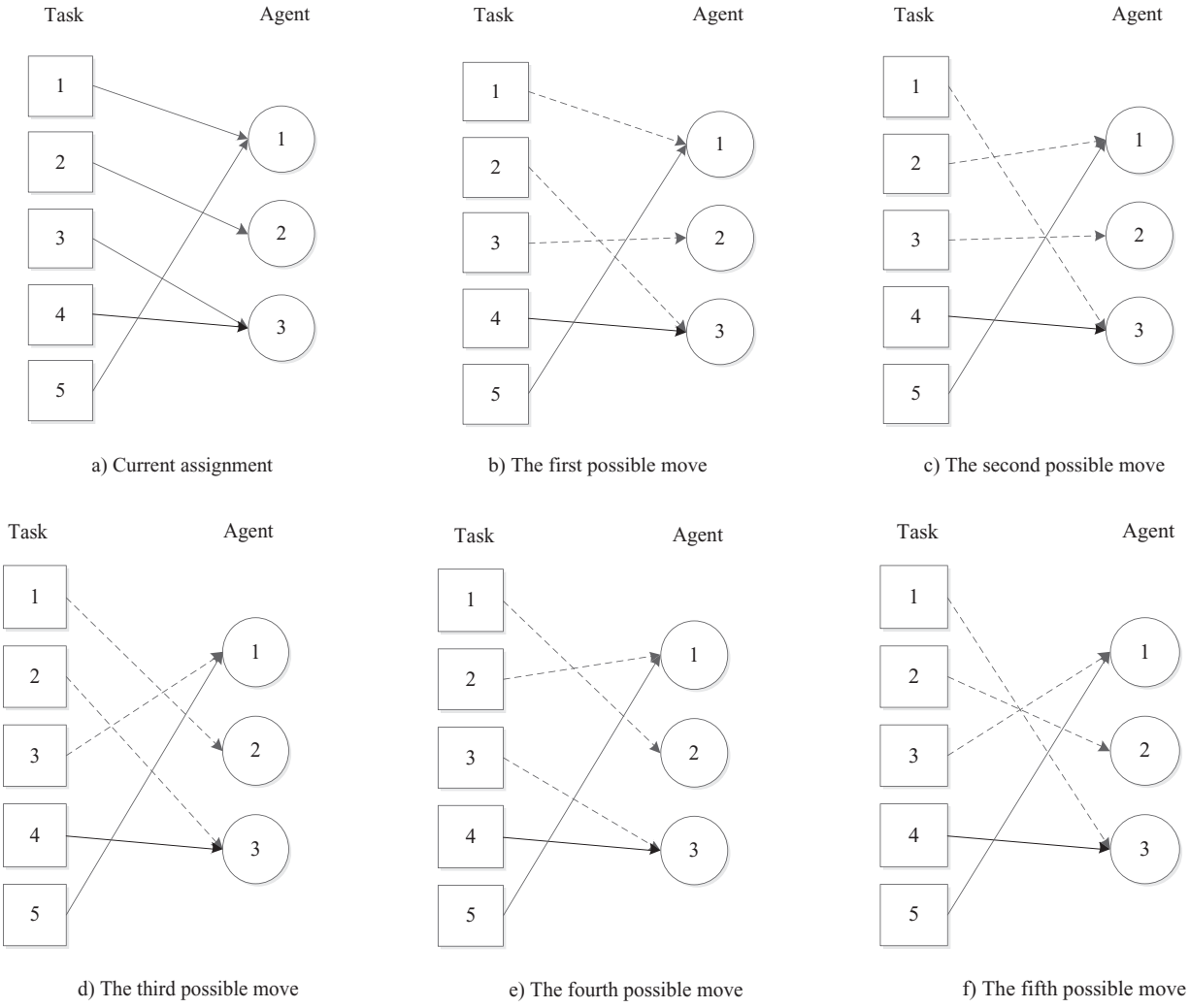


Fig. 5. Example of *k*-variables move algorithm.

3	11	10	8	4	6	1	7	9	12	14	15	13	2	5	D	E	I
0.01	0.02	0.07	0.10	0.25	0.26	0.60	0.60	0.62	0.67	0.73	0.74	0.76	0.85	0.96	0.49	0.75	0.90

Fig. 6. Example of a vector used in *k*-self-adaptive procedure.

Step 1: Initialization phase
Set the generation number G=0,
Randomly generated NP target vector ( ) when has uniformly distributed in the range [X_{min},X_{max}]
Step 2: Evaluate each target vector (the evaluation is shown in section 3.1 and apply local search which explains in section 3.6)
Step 3: Generate NP trial vectors by repeating Steps 3.1–3.2
Step 3.1: Apply mutation process by generating a mutant vector using formula (5)
Step 3.2: Apply recombination process by generating a trial vector using formula (6)
Step 3.3 : Evaluate each trial vector (the evaluation is shown in section 3.1 and apply local search already explained in section 3.6)
Step 4: Applying selection process using formula (7)
Step 5: Increment the generation count G=G+1 if the termination criterion is not satisfied, go to step3 ; Otherwise, stop the procedure.

Fig. 7. Algorithmic description of DE for GAP.

Table 3
Detail of the test instances.

Name of test instances	Number of task	Number of machine	Number of test instances	Size of problem
Gap1–12	15–60	5-10	60	Small
Gapa (type A)	10–50	5-25	6	Medium
Gapb (type B)	100–200	5-20	6	Large
Gabc (type C)	100–400	5-40	6	Large
Gabd (type D)	100–400	5-40	6	Large
Gabe (type E)	100–400	5-40	6	Large

problem) will be selected to be the representative of the *k*-variable move algorithm. Selecting *k* to perform in the proposed algorithm is important. In the proposed algorithm, *k* is selected by using a self-adaptive procedure which is integrated into the DE mechanism. The first step of the procedure is to modify the vector shown in Fig. 2 by extending three more positions (extend dimensions of the vector) and all values in extended position are randomly generated in the first generation while in the next generation of the proposed algorithm the values are obtained from DE operators. An example of a vector modified from Fig. 2 in order to enable the k-self-adaptive procedure is shown in Fig. 6.

Fig. 6 shows an example of a vector used in the proposed heuristic. The last three positions use letters instead of numbers because they represent Decrease (D), Equal (E) and Increase (I). Firstly, *k* (all vectors) is set to be 2, then each vector can have a value of *k* equal to *k* (E), $k-1$(I), or $k+1$ (D) depending on which position (D, E, and I) has the highest value. From Fig. 6, the maximum value in the last three positions is 0.90, then *k* will equal 2 + 1 or 3. The minimum value of *k* is set to be 2 in this study. If the result from the adjustment of *k* is lower than 2, the algorithm will set *k* equal to 2. The proposed DE is summarized in its entirety as shown in Fig. 7.

5. Computational experiment and results

The proposed algorithms are tested using 6 sets of test instances consisting of 99 test instances in total. Details of the test instances including number of tasks and agents are shown in Table 3. All test data is taken from Diaz and Fernandez (2001). The different test instances are generated based on different control parameters such as limitation of R_{ij}, C_{ij} and the tightness of b_j. Details of the differences of each test instance are explained in Diaz and Fernandez (2001).

The DE algorithms proposed to solve the GAP in this paper are able to enhance the solution quality by adding different local searches in the search mechanism of DE. The quantity to investigate is the performance of the heuristic algorithms, obtained by comparing their solutions to the optimal solution. For these comparisons, all combinations of the proposed algorithms with local search techniques in DE are tested. Hence, there are totally 8 algorithms in the combinations of different types of local searches in the DE listed below.

(1) DE the proposed algorithm without local search
(2) DE-S the proposed algorithm using shifting algorithm as the local search
(3) DE-SW the proposed algorithm using SWAP as the local search
(4) DE-K the proposed algorithm using *k*-variable move as the local search
(5) DE-SSW the proposed algorithm using shifting and SWAP algorithms as the local search
(6) DE-SK the proposed algorithm using shifting and *k*-variable move as the local search
(7) DE-SWK the proposed algorithm using SWAP and *k*-variable move as the local search
(8) DE-SSWK the proposed algorithm using shifting, SWAP and *k*-variable move as the local search

Details of the test are presented in Section 4.1. Additionally, the best proposed algorithm obtained from the performance comparison is taken to compare with other existing metaheuristics found in the literature. Details of the test are presented in Section 4.2.

5.1. Comparison among the proposed algorithms

Twelve sets of test instances obtained from the OR library originally described in Beasley (1990) were used. Each set is composed of 5 test instances. Hence, 60 test instances in total were used for testing to obtain the best proposed algorithm among the different combination of DE and local search techniques. The best proposed algorithm will be used as the representative of the proposed heuristics for comparing to other heuristics found in the literature. In this comparison, time limitation was used as the stopping criteria in order to check the performance of the local search techniques. The computation time was set as 30 s. Each test of instances was executed 5 times. Results of the tests in terms of (1) % average deviation from the optimal solution and (2) number of optimal solutions found from the proposed algorithm are presented in Table 4. In this table, the results are reported as *x, y*/*z*. This means this set of test instances composes *z* test instances and that algorithm can find optimal solution *y* out of *z* instances with the average of *x*% deviation from the optimal solution.

Table 4
Compare solution quality of the proposed heuristics.

Problem test	DE	DE-S	DE-SW	DE-K	DE-SSW	DE-SK	DE-SWK	DE-SSWK
Gap1	0.56, 1/5	0.30, 1/5	0.25, 1/5	0.18, 4/5	0.25, 3/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap2	0.25, 1/5	0.23, 1/5	0.35, 1/5	0.25, 3/5	0.35, 3/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap3	0.56, 0/5	0.24, 0/5	0.24, 0/5	0.18, 3/5	0.21, 3/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap4	0.26, 0/5	0.16, 0/5	0.16, 0/5	0.02, 2/5	0.08, 1/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap5	0.67, 0/5	0.15, 0/5	0.14, 0/5	0.18, 2/5	0.21, 2/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap6	0.98, 0/5	0.58, 0/5	0.54, 0/5	0.72, 1/5	0.89, 0/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap7	0.89, 0/5	0.90, 0/5	0.90, 0/5	0.72, 2/5	0.87, 1/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap8	1.20, 0/5	0.86, 0/5	0.89, 0/5	0.24, 1/5	0.98, 0/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap9	1.10, 0/5	0.76, 0/5	0.78, 0/5	0.65, 1/5	0.87, 0/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap10	0.86, 0/5	0.89, 0/5	0.98, 0/5	0.89, 1/5	0.98, 0/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap11	0.95, 0/5	0.45, 0/5	0.98, 0/5	0.15, 3/5	0.21, 0/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Gap12	0.82, 0/5	0.85, 0/5	0.89, 0/5	0.25, 2/5	0.56, 0/5	0.00, 5/5	0.00, 5/5	0.00, 5/5
Average	0.76	0.53	0.59	0.37	0.54	0.00	0.00	0.00
Total #	2	2	2	27	13	60	60	60

Note: *Average* = the average of % deviation of a proposed heuristic.
Total # = a number of test instances that a proposed heuristic can find the optimal solution.

Table 5
Computational time of the proposed algorithm had found the optimal solution.

Problem	Algorithm							
test	DE	DE-S	DE-SW	DE-K	DE-SSW	DE-SK	DE-SWK	DE-SSWK
Gap1	28.45	25.89	0.25	0.18	0.25	0.02	0.02	0.02
Gap2	27.68	26.58	0.35	0.25	0.35	0.01	0.01	0.01
Gap3	30	30	0.24	0.18	0.21	0.01	0.01	0.01
Gap4	30	30	0.16	0.02	0.08	0.02	0.02	0.02
Gap5	30	30	0.14	0.18	0.21	0.01	0.01	0.01
Gap6	30	30	0.54	0.72	0.89	0.01	0.25	0.01
Gap7	30	30	0.9	0.72	0.87	0.02	0	0.01
Gap8	30	30	0.89	0.24	0.98	0.01	0.04	0.01
Gap9	30	30	0.78	0.65	0.87	0.01	0.03	0.02
Gap10	30	30	0.98	0.89	0.98	0.02	0.01	0.02
Gap11	30	30	0.98	0.15	0.21	0.02	0.03	0.02
Gap12	30	30	0.89	0.25	0.56	0.01	0.02	0.02
Average	29.67	29.37	0.59	0.37	0.54	0.01	0.04	0.02

Table 6
Compare performance of the proposed algorithm.

Problem	Algorithm															
size	DE		DE-S		DE-SW		DE-K		DE-SSW		DE-SK		DE-SWK		DE-SSWK	
	#Opt	CT	#Opt	CT	#Opt	CT	#Opt	CT	#Opt	CT	#Opt	CT	#Opt	CT	#Opt	CT
Small	1	10	1	10	1	0.59	27	0.37	13	0.54	60	0.01	60	0.04	60	0.02
Medium	0	10	1	9.5	2	7.6	3	7.23	3	5.6	6	0.66	6	1.07	6	1.27
Large	0	350	0	350	0	350	1	301	0	350	6	134	6	161	6	165

The %deviation is measured as presented in Eq. (8):

$$\%\,deviation = \frac{Algo - Opt}{Algo} \times 100 \tag{8}$$

where *Opt* is the optimal solution represents; and *Algo* is the solution from the proposed algorithm.

From Table 4, DE, DE-S, DE-SW, DE-CY, DE-SSW, DE-SCY, DE-SWCY and DE-SSWCY yielded 0.76%, 0.53%, 0.59%, 0.37%, 0.54%, 0.00%, 0.00% and 0.00% deviation from the optimal solution, respectively. These algorithms can find 2, 2, 2, 27, 13, 60, 60 and 60 test instances for the optimal solution, respectively. From the computational results, the algorithms with the *k*-variable move in combination associated with at least one local search techniques can generate the best solution (i.e., 100% optimal solution found) among all proposed algorithms. Moreover, the experiment to determine the computational time that each combination of the proposed algorithm used to find the optimal solution was conducted. In this experiment, maximum allowable time to reach the optimal solution is set as 30 s. The computational time to reach each optimal solution of each proposed algorithm is reported in Table 5. From this table, DE-SK is the best combination since it yields the lowest computational time (0.01 s) to obtain the same solution quality as other proposed heuristics.

Finally, the experiments were conducted in the following groups: (1) gap1–12 (small-60 instances), (2) Gapa (medium-6 instances) and Gapb (large-6 instances) to check if the proposed algorithms have the same behavior (DE-SK is the best proposed heuristic) for small, medium and large test instances. The computational result is shown in Table 6. All test instances were tested 5 times. The best solution was recorded for each test instance. The number of optimal solutions found by each proposed heuristic is reported in column "#opt". The average computational time is shown in column "CT". The termination condition used for gap1–12, Gapa and Gapb is the computational time which is set to 10, 10, and 350 CPU times, respectively. From Table 6, we can see that DE-SK is the best algorithm since it can find all optimal solutions (100%) for small, medium and large set of instances with lowest computational time compared with other proposed heuristics. Thus DE-SK will be used as the representation of the

Table 7
Computational result compare with existing heuristics.

Problem	Algorithm					
	BEE		TABU		DE-SK	
	CPU	FITNESS	CPU	FITNESS	CPU	FITNESS
Gapa1	0.18	1698	n.a.	n.a.	0.11	1698
Gapa2	0.95	3235	n.a.	n.a.	0.59	3235
Gapa3	0.27	1360	n.a.	n.a.	1.02	1360
Gapa4	1.31	2623	n.a.	n.a.	0.35	2623
Gapa5	0.41	1158	n.a.	n.a.	0.65	1158
Gapa6	1.81	2339	n.a.	n.a.	1.21	2339
Gapb1	5.97	1843	95.8	1843	30.56	1843
Gapb2	45.99	1407	97.2	1407	50.89	1407
Gapb3	0.36	1166	160	1166	60.25	1166
Gapb4	315.04	3552	339.1	3552	102.4	3552
Gapb5	1.32	2827	389.8	2828	212.5	2827
Gapb6	28.65	2339	465.4	2340	350.5	2339
Gapc1	3.61	1931	83.6	1931	35.4	1931
Gapc2	18.09	1402	103.3	1402	52.98	1402
Gapc3	5.81	1243	130.7	1243	101.34	1243
Gapc4	488.89	3456	316.1	3457	202.85	3456
Gapc5	23.16	2806	403	2807	280.95	2806
Gapc6	646.18	2392	498.3	2391	420.46	2391
Gapc7	n.a.	n.a.	1823	5598	690.4	5598
Gapc8	n.a.	n.a.	2236.3	4786	750.3	4782
Gapc9	n.a.	n.a.	3442.4	4248	810.56	4244
Gapd1	916.03	6353	106.4	6357	60.89	6353
Gapd2	122.41	6356	133.4	6355	104.5	6355
Gapd3	538.29	6221	167.4	6220	209.45	6196
Gapd4	743.95	12744	313.2	12747	150.49	12744
Gapd5	1704.46	12442	397.4	12457	234.54	12442
Gapd6	922.94	12276	515.6	12351	398.67	12276
Gapd7	n.a.	n.a.	2310	25039	450.56	24996
Gapd8	n.a.	n.a.	2440.1	24747	789.2	24705
Gapd9	n.a.	n.a.	3129.3	24707	1001.2	24570
Gape1	n.a.	n.a.	124	12681	58.92	12681
Gape2	n.a.	n.a.	148.2	11581	70.89	11577
Gape3	n.a.	n.a.	197.2	8460	90.35	8460
Gape4	n.a.	n.a.	351.4	24931	80.86	24931
Gape5	n.a.	n.a.	396.4	23318	145.4	23307
Gape6	n.a.	n.a.	585.3	22422	178.2	22422
Gape7	n.a.	n.a.	891.1	45781	220.3	45781
Gape8	n.a.	n.a.	875	44579	420.2	44579
Gape9	n.a.	n.a.	1271.1	44882	650.5	44882
Average CT	272.34		755.65		242.85	
#best	21/24		17/33		39/39	
%best	87.5%		51.51%		100%	

∗n.a. = not available in the literature, #best = a/b when a is total number of lowest cost compared among all heuristics and b is total number of test instances which are simulated by an algorithm, and %best is percent that an algorithm found the lowest cost compared among other heuristics.

proposed algorithm to compare with other existing heuristics proposed in the literature.

5.2. Comparison with other heuristics found in the literature

In this section, the best proposed algorithm (DE-SK) obtained from Section 4.1 is used to compare with other heuristics found in the literature. The DE-SK algorithm is compared with BEE (Özbakir et al., 2010) and Tabu (Diaz & Fernandez, 2001). The heuristic algorithms are compared using two performance measures: (1) solution quality, and (2) computational speed. The number of iterations was used as the stopping criteria. Each experiment was run 5 times, each of which used 1000 iterations as the stopping criteria. Thirty nine test instances were used for the test. These test instances were divided into 5 sets which were set Gapa, Gapb, Gapc, Gapd, and Gape. Details of test instances were already explained in the previous section. The computation results are shown in Table 7. From this table, it can be seen that the proposed heuristic (DE-SK) yields the lowest cost (%best) for all instances, while the BEE and Tabu algorithms reach the lowest cost 21 out of 24 instances (or 87.5%), and 20 out of 39 instances (or 51.51%), respectively. Additionally, statistical test was used to evaluate if three algorithms yield the same performance. The analysis was performed using SAS Software V8 for Windows. The statistical results are reported in Table 8. From this table, it can be seen that BEE and DE-K algorithms are not significantly different, while TABU and BEE and TABU and DE-K are significantly different.

Table 8
p-Value of comparing a pair of heuristics.

Solution	BEE	TABU	DE-K
BEE	–	0.034	0.076
TABU	0.034	–	0.000
DE-K	0.076	0.000	–

6. Conclusion

In this paper, modified DE algorithms were developed to optimize the sequence of tasks assigned to agents with the minimum assignment costs. The DE was used in the combination of three local techniques which are shifting, SWAP, and k-variable move algorithms. Eight DE-based algorithms are presented and each of them uses DE with a different combination of local search techniques. From the comparison of the efficiency among the proposed algorithms we found that DE is more efficient with the local search techniques, especially the k-variable move and shifting algorithm (DE-SK). Moreover, the best proposed algorithm (DE-SK) was used to compare its performance with those of BEE and TABU algorithms as the existing heuristics found in the literature proposed by Özbakir et al. (2010) and Diaz and Fernandez (2001), respectively. The computational results reveal that the BEE and DE-SK algorithms are not significantly different while DE-SK outperformed the TABU algorithm. This is because the DE-SK is designed to enhance the search capability by improving the diversification using the DE's operators and k-variable moves are able to improve the intensification. Even though the statistical test shows that the DE-SK is not significantly different compared with the BEE algorithm, the DE-SK algorithm is able to obtain more optimal solutions (21 out of 24 test instances or 87.5%), while the BEE algorithm can obtain the optimal solutions in only 3 out of 24 test instances (or 12.5%). This means the unsolved instances that cannot find the optimal solution yet from the BEE algorithm can be solved by a combination of good diversification ability of the normal DE operators and intensification ability of the local search technique used in the proposed heuristics. Hence, the proposed algorithms, especially the DE-SK can be used to solve various practical applications of GAP and other optimization problems by enhancing the solution quality, while still maintaining fast computational time. Although the proposed DE algorithm has shown an outstanding ability to solve the problem at hand, there is a possibility to use other hybrid methods to improve the solutions of the problems.

Acknowledgments

This paper was supported by the Research Unit on System Modeling for Industry (Grant no. SMI.KKU 1/2558), Khon Kaen University, Thailand.

References

Amini, M. M., & Racer, M. (1994). A rigorous computational comparison of alternative solution methods for the generalized assignment problem. *Management Science, 40*(7), 868–890.
Avella, P., Boccia, M., & Vasilyev, I. (2010). A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization and Applications, 45*(3), 543–555.
Barcia, P., & Jörnsten, K. (1990). Improved Lagrangean decomposition: an application to the generalized assignment problem. *European Journal of Operational Research, 46*(1), 84–92.

Beasley, J. E. (1990). OR-library: distributing test problems by electronic mail. *Journal of the Operational Research Society, 41*(11), 1069–1072.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., & Zumer, V. (2006). Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation, 10*(6), 646–657.

Cao, E., & Lai, M. (2010). The open vehicle routing problem with fuzzy demands. *Expert Systems with Applications, 37*(3), 2405–2411.

Cattrysse, D. G., & Van Wassenhove, L. N. (1992). A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research, 60*(3), 260–272.

Cattrysse, D. G., Salomon, M., & Van Wassenhove, L. N. (1994). A set partitioning heuristic for the generalized assignment problem. *European Journal of Operational Research, 72*(1), 167–174.

Chamnanlor, C., Sethanan, K., Gen, M., & Chien, C. F. (2015). Embedding ant system in genetic algorithm for re-entrant hybrid flow shop scheduling problems with time window constraints. *Journal of Intelligent Manufacturing*, 1–17. doi: 10.1007/s10845-015-1078-92.

Chu, P. C., & Beasley, J. E. (1997). A genetic algorithm for the generalised assignment problem. *Computers & Operations Research, 24*(1), 17–23.

Dechampai, D., Tanwanichkul, L., Sethanan, K., & Pitakaso, R. (2015). A differential evolution algorithm for the capacitated VRP with flexibility of mixing pickup and delivery services and the maximum duration of a route in poultry industry. *Journal of Intelligent Manufacturing*, 1–20. doi:10.1007/s10845-015-1055-3.

Dıaz, J. A., & Fernández, E. (2001). A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research, 132*(1), 22–38.

Erbao, C., & Lai, M. (2009). A hybrid differential evolution algorithm to vehicle routing problem with fuzzy demands. *Journal of Computational and Applied Mathematics, 231*(1), 302–310.

Fan, Q., & Yan, X. (2015). Self-adaptive differential evolution algorithm with discrete mutation control parameters. *Expert Systems with Applications, 42*(3), 1551–1572.

Fisher, M. L., & Jaikumar, R. (1981). A generalized assignment heuristic for vehicle routing. *Networks, 11*(2), 109–124.

Fisher, M. L., Jaikumar, R., & Van Wassenhove, L. N. (1986). A multiplier adjustment method for the generalized assignment problem. *Management Science, 32*(9), 1095–1103.

Fu, Y. M., Diabat, A., & Tsai, I. T. (2014). A multi-vessel quay crane assignment and scheduling problem: Formulation and heuristic solution approach. *Expert Systems with Applications, 41*(15), 6959–6965.

Gao, L., Zhou, Y., Li, X., Pan, Q., & Yi, W. (2015). Multi-objective optimization based reverse strategy with differential evolution algorithm for constrained optimization problems. *Expert Systems with Applications, 42*(14), 5976–5987.

Gong, W., Cai, Z., & Jiang, L. (2008). Enhancing the performance of differential evolution using orthogonal design method. *Applied Mathematics and Computation, 206*(1), 56–69.

Gong, W., Cai, Z., & Ling, C. X. (2010). DE/BBO: a hybrid differential evolution with biogeography-based optimization for global numerical optimization. *Soft Computing, 15*(4), 645–665.

Hegerty, B., Hung, C. C., & Kasprak, K. (2009). A comparative study on differential evolution and genetic algorithms for some combinatorial problems. In *Proceedings of the 8th Mexican international conference on artificial intelligence* http://www.micai.org/2009/proceedings/.../cd/ws.../paper88.micai09.pdf.

Higgins, A. J. (2001). A dynamic tabu search for large-scale generalised assignment problems. *Computers & Operations Research, 28*(10), 1039–1048.

Jeet, V., & Kutanoglu, E. (2007). Lagrangian relaxation guided problem space search heuristics for generalized assignment problems. *European Journal of Operational Research, 182*(3), 1039–1056.

Jiang, Z. Z., Xia, C., Chen, X., Meng, X., & He, Q. (2013). A discrete differential evolution algorithm for the multi-objective generalized assignment problem. *Journal of Computational and Theoretical Nanoscience, 10*(12), 2819–2825.

Jörnsten, K., & Näsberg, M. (1986). A new Lagrangian relaxation approach to the generalized assignment problem. *European Journal of Operational Research, 27*(3), 313–323.

Krumke, S. O., & Thielen, C. (2013). The generalized assignment problem with minimum quantities. *European Journal of Operational Research, 228*(1), 46–55.

Laguna, M., Kelly, J. P., González-Velarde, J., & Glover, F. (1995). Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research, 82*(1), 176–189.

Li, Fachao, Li, DaXu, Jin, Chenxia, & Wang, Hong (2012). Random assignment method based on genetic algorithms and its application in resource allocation. *Expert Systems with Applications, 39*, 12213–12219.

Li, Fachao, Li, Da Xu, Jin, Chenxia, & Wang, Hong (2012). Study on solution models and methods for the fuzzy assignment problems. *Expert Systems with Applications, 39*, 11276–11283.

Litvinchev, I., Mata, M., Rangel, S., & Saucedo, J. (2010). Lagrangian heuristic for a class of the generalized assignment problems. *Computers & Mathematics with Applications, 60*(4), 1115–1123.

Liu, L., Mu, H., Song, Y., Luo, H., Li, X., & Wu, F. (2012). The equilibrium generalized assignment problem and genetic algorithm. *Applied Mathematics and Computation, 218*(11), 6526–6535.

Lorena, L. A. N., & Narciso, M. G. (1996). Relaxation heuristics for a generalized assignment problem. *European Journal of Operational Research, 91*(3), 600–610.

Mokhtari, H., & Salmasnia, A. (2015). A Monte carlo simulation based chaotic differential evolution algorithm for scheduling a stochastic parallel processor system. *Expert Systems with Applications, 42*(20), 7132–7147.

Muñoz, D. F., & Muñoz, D. F. (2012). Algorithms for the generalized weighted frequency assignment problem. *Computers & Operations Research, 39*(12), 3256–3266.

Narciso, M. G., & Lorena, L. A. N. (1999). Lagrangean/surrogate relaxation for generalized assignment problems. *European Journal of Operational Research, 114*(1), 165–177.

Nauss, R. M. (2003). Solving the generalized assignment problem: an optimizing and heuristic approach. *INFORMS Journal on Computing, 15*(3), 249–266.

Nearchou, A. C. (2005). A differential evolution algorithm for simple assembly line balancing. In *Proceedings of the paper presented in the 16th International Federation of Automatic Control (IFAC) world congress, 4–8 July, Prague, Czech Republic*.

Nearchou, A. C. (2007). Balancing large assembly lines by a new heuristic based on differential evolution method. *The International Journal of Advanced Manufacturing Technology, 34*(9–10), 1016–1029.

Nearchou, A. C. (2008). Multi-objective balancing of assembly lines by population heuristics. *International Journal of Production Research, 46*(8), 2275–2297.

Onwubolu, G., & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research, 171*(2), 674–692.

Osman, I. H. (1995). Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations-Research-Spektrum, 17*(4), 211–225.

Özbakir, L., Baykasoğlu, A., & Tapkan, P. (2010). Bees algorithm for generalized assignment problem. *Applied Mathematics and Computation, 215*(11), 3782–3795.

Pan, Q. K., Tasgetiren, M. F., & Liang, Y. C. (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering, 55*(4), 795–816.

Park, J. S., Lim, B. H., & Lee, Y. (1998). A Lagrangian dual-based branch-and-bound algorithm for the generalized multi-assignment problem. *Management Science, 44*(12-part-2), S271–S282.

Privault, C., & Herault, L. (1998). Solving a real world assignment problem with a metaheuristic. *Journal of Heuristics, 4*(4), 383–398.

Qin, A. K., Huang, V. L., & Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation, 13*(2), 398–417.

Romeijn, H. E., & Morales, D. R. (2000). A class of greedy algorithms for the generalized assignment problem. *Discrete Applied Mathematics, 103*(1), 209–235.

Ross, G. T., & Soland, R. M. (1975). A branch and bound algorithm for the generalized assignment problem. *Mathematical programming, 8*(1), 91–103.

Sangsawang, C., Sethanan, K., Fujimoto, T., & Gen, M. (2015). Metaheuristics optimization approaches for two-stage reentrant flexible flow shop with blocking constraint. *Expert Systems with Applications, 42*(5), 2395–2410.

Savelsbergh, M. (1997). A branch-and-price algorithm for the generalized assignment problem. *Operations Research, 45*(6), 831–841.

Shmoys, D. B., & Tardos, E. (1993). An approximation algorithm for the generalized assignment problem. *Mathematical Programming, 62*, 461–474.

Shtub, A., & Kogan, K. (1998). Capacity planning by the dynamic multi-resource generalized assignment problem (DMRGAP). *European Journal of Operational Research, 105*(1), 91–99.

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.

Subtil, R. F., Carrano, E. G., Souza, M. J., & Takahashi, R. H. (2010, July). Using an enhanced integer NSGA-II for solving the multiobjective generalized assignment problem. In *Proceedings of the 2010 IEEE congress on volutionary computation (CEC)* (pp. 1–7). IEEE.

Tapkan, P. N., ÖZbakıR, L., & BaykasoğLu, A. (2013). Solving fuzzy multiple objective generalized assignment problems directly via bees algorithm and fuzzy ranking. *Expert Systems with Applications, 40*(3), 892–898.

Tasgetiren, M. F., Suganthan, P. N., Chua, T. J., & Al-Hajri, A. (2009). Differential evolution algorithms for the generalized assignment problem. In *Proceedings of the 2009 IEEE congress on volutionary computation (CEC), May 2009* (pp. 2606–2613). IEEE.

Trick, M. A. (1992). A linear relaxation heuristic for the generalized assignment problem. *Naval Research Logistics, 39*, 137–151.

Wang, L., Pan, Q. K., Suganthan, P. N., Wang, W. H., & Wang, Y. M. (2010). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research, 37*(3), 509–520.

Wang, Y. Z. (2002). An application of genetic algorithm methods for teacher assignment problems. *Expert Systems with Applications, 22*(4), 295–302.

Wilson, J. M. (1997). A genetic algorithm for the generalised assignment problem. *Journal of the Operational Research Society, 48*(8), 804–809.

Wisittipanich, W., & Kachitvichyanukul, V. (2011). Differential evolution algorithm for job shop scheduling problem. *Industrial Engineering and Management Systems, 10*(3), 203–208.

Wisittipanich, W., & Kachitvichyanukul, V. (2012). Two enhanced differential evolution algorithms for job shop scheduling problems. *International Journal of Production Research, 50*(10), 2757–2773.

Yagiura, M., Ibaraki, T., & Glover, F. (2004). An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing, 16*(2), 133–151.

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*(2), 548–569.

Yagiura, M., Yamaguchi, T., & Ibaraki, T. (1998). A variable depth search algorithm with branching search for the generalized assignment problem. *Optimization Methods and Software, 10*(2), 419–441.