Applied Mathematics and Computation 215 (2010) 3782–3795

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

Bees algorithm for generalized assignment problem

Lale Özbakir[a], Adil Baykasoğlu[b,*], Pınar Tapkan[a]

[a] Erciyes University, Department of Industrial Engineering, Kayseri, Turkey
[b] University of Gaziantep, Department of Industrial Engineering, Gaziantep, Turkey

A R T I C L E   I N F O

Keywords:
Meta-heuristics
Bees algorithm
Generalized assignment problem
Ejection chain
Integer programming
Swarm intelligence

A B S T R A C T

Bees algorithm (BA) is a new member of meta-heuristics. BA tries to model natural behavior of honey bees in food foraging. Honey bees use several mechanisms like waggle dance to optimally locate food sources and to search new ones. This makes them a good candidate for developing new algorithms for solving optimization problems. In this paper a brief review of BA is first given, afterwards development of a BA for solving generalized assignment problems (GAP) with an ejection chain neighborhood mechanism is presented. GAP is a NP-hard problem. Many meta-heuristic algorithms were proposed for its solution. So far BA is generally applied to continuous optimization. In order to investigate the performance of BA on a complex integer optimization problem, an attempt is made in this paper. An extensive computational study is carried out and the results are compared with several algorithms from the literature.

1. Introduction

A relatively new branch of nature inspired algorithms which are called as swarm intelligence is focused on insect behavior in order to develop some effective meta-heuristics which can mimic insect's problem solution abilities. Interaction between insects contributes to the collective intelligence of the social insect colonies. These communication systems between insects have been adapted to scientific problems for optimization. One of the examples of such interactive behavior is the waggle dance of bees during the food procuring. By performing this dance, successful foragers share the information about the direction and distance to patches of flower and the amount of nectar within this flower with their hive mates. This is a successful mechanism which foragers can recruit other bees in their colony to productive locations to collect various resources. Bee colony can quickly and precisely adjust its searching pattern in time and space according to changing nectar sources.

The information exchange among individual insects is the most important part of the collective knowledge. Communication among bees about the quality of food sources is being achieved in the dancing area by performing waggle dance. The previous studies on dancing behavior of bees show that while performing the waggle dance, the direction of bees indicates the direction of the food source in relation to the Sun, the intensity of the waggles indicates how far away it is and the duration of the dance indicates the amount of nectar on related food source. Waggle dancing bees that have been in the hive for an extended time adjusts the angles of their dances to accommodate the changing direction of the sun. Therefore bees that follow the waggle run of the dance are still correctly led to the food source even though its angle relative to the sun has changed. So collective intelligence of bees based on the synergistic information exchange during waggle dance. Observations and studies on honey bee behaviors resulted in a new generation of optimization algorithms. Such an algorithm which is known

* Corresponding author.
E-mail address: baykasoglu@gantep.edu.tr (A. Baykasoğlu).

as bees algorithm (BA) is used in this paper in order to solve GAP. BA was initially proposed by Pham et al. [1]. In this paper, the BA algorithm is adapted to GAP problem by employing an ejection chain type neighborhood mechanism.

The main purpose in GAP is to assign a set of tasks to a set of agents with a minimum total cost. Each agent represents a single resource with limited capacity. Each task must be assigned to only one agent and it requires a certain amount of the resource of the agent. There are many application domains of GAP such as location problems, vehicle routing, group technology, scheduling. Extended review of GAP and its possible applications was presented in Martello and Toth [2,3], Cattrysse [4] and Cattrysse et al. [5]. Several exact algorithms for GAP were proposed by Ross and Soland [6], Fisher et al. [7], Martello and Toth [3], Savelsbergh [8] and more recently Nauss [9]. Several heuristics have also been proposed to solve GAP. Martello and Toth [2,3] proposed a combination of local search and greedy method. Osman [10] developed simulated annealing and tabu search algorithms to solve GAP. Chu and Beasley [11] presented a genetic algorithm for GAP that tries to improve feasibility and optimality simultaneously. Different variable depth search algorithms [12–14], ejection chain based tabu search algorithms [15–17], max–min ant system based on greedy randomized adaptive heuristic [18], path relinking approaches [19–24], ant colony optimization [25], genetic algorithm with constraint-ratio heuristic [26] can be mentioned as the other meta-heuristic approaches which were all proposed for GAP in recent years.

The purpose of this study is to present a modified BA to solve GAP. Our main interest on this problem came from its NP-hard structure that was proved by Fisher et al. [7]. Moreover, Mortello and Toth [3] presented the NP-completeness of proving that a solution is a feasible solution. The performance of modified BA on this problem might be a good indicator for its ability to tackle complex constrained combinatorial optimization problems. GAP can be formulated as an integer programming model as follows:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}, \\
\text{subject to} \quad & \sum_{i=1}^{n} a_{ij}x_{ij} \leqslant b_j \quad \forall j, \quad 1 \leqslant j \leqslant m, \\
& \sum_{j=1}^{m} x_{ij} = 1 \quad \forall i, \quad 1 \leqslant i \leqslant n, \\
& x_{ij} \in \{0,1\} \quad 1 \leqslant i \leqslant n \quad \forall i, \quad 1 \leqslant j \leqslant m \quad \forall j,
\end{aligned} \tag{1}$$

where, I is set of tasks $(i = 1,\ldots,n)$; J is set of agents $(j = 1,\ldots,m)$; b_j is resource capacity of agent $j(b_j \geqslant 0)$; a_{ij} is resource needed if task i is assigned to agent $j(a_{ij} \geqslant 0)$; c_{ij} is cost of task i if assigned to agent $j(c_{ij} \geqslant 0)$; x_{ij} is decision variable ($x_{ij} = 1$, if task i is assigned to agent j; 0, otherwise). The objective function represents the total assignment cost where the first constraint set is related to the resource capacity of agents and the second constraint set ensure that each task is assigned to only one agent.

In the following sections of this paper, a review of BA algorithm and its applications are presented in Section 2. In Section 3, the BA as proposed by Pham et al. [1] is explained in detail along with its application details to GAP. Lastly, in Section 4 computational results and comparisons are presented.

2. Behavior of bees in nature

Social insect colonies can be considered as dynamical system gathering information from the environment and adjusting their behavior in accordance to it. While gathering information and adjustment processes, individual insects do not perform all the tasks because of their specializations. Generally, all social insect colonies behave according to their own division of labors related to their morphology [27]. As also presented in authors previous work [27], bee system consists of two essential components.

2.1. Food sources

The value of a food source depends on different parameters such as its proximity to the nest, richness of energy and ease of extracting this energy.

2.1.1. Foragers

– *Unemployed foragers:* If it is assumed that a bee have no knowledge about the food sources in the search field, bee initializes its search as an unemployed forager. There are two possibilities for an unemployed forager: (1) *Scout Bee* (*S* in Fig. 1): if the bee starts searching spontaneously without any knowledge, it will be a scout bee. The percentage of scout bees varies from 5% to 30% according to the information into the nest. The mean number of scouts averaged over conditions is about 10% [28]. (2) *Recruit* (*R* in Fig. 1): if the unemployed forager attends to a waggle dance done by some other bee, the bee will start searching by using the knowledge from waggle dance.

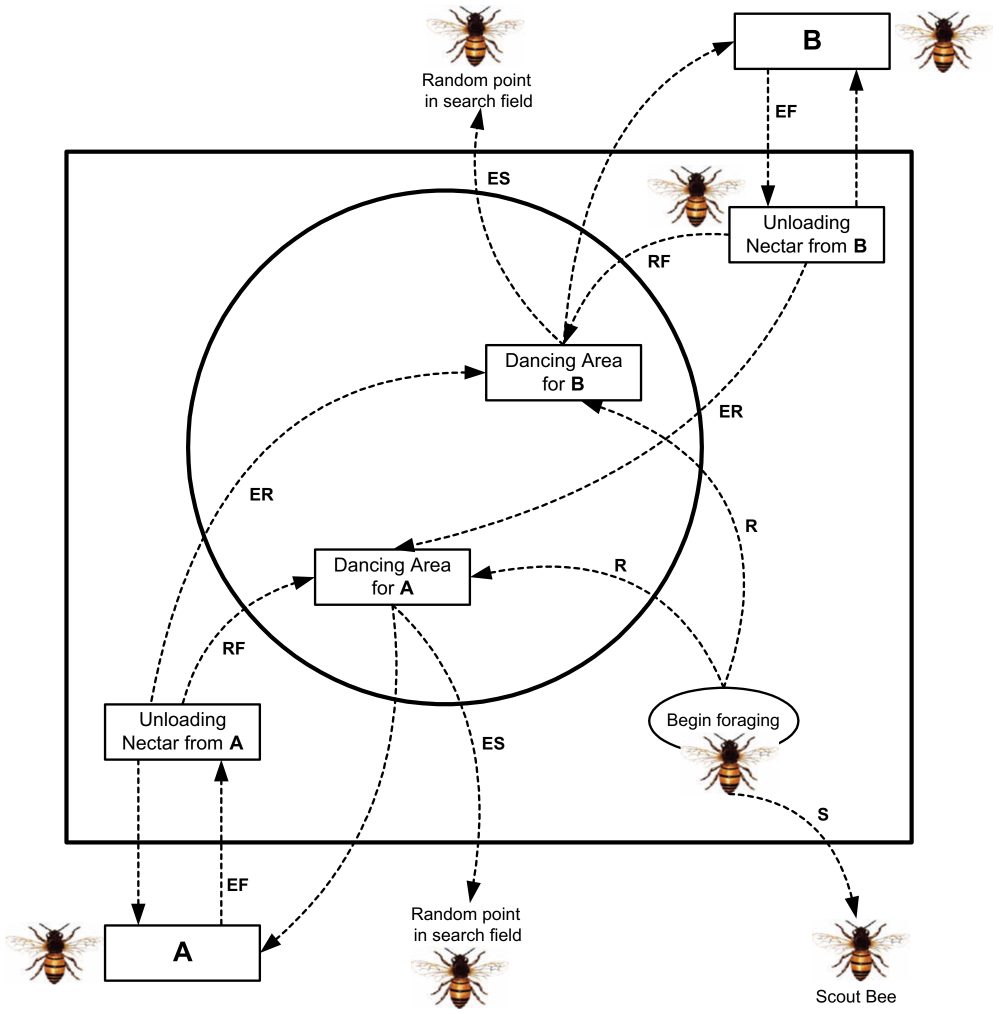

Fig. 1. Typical behavior of honey bee foraging [27].

- *Employed foragers:* (*EF* in Fig. 1): when the recruit bee finds and exploits the food source, it will raise to be an employed forager who memorizes the location of the food source. After the employed foraging bee loads a portion of nectar from the food source, it returns to the hive and unloads the nectar to the food area in the hive. There are three possible options related to residual amount of nectar for the foraging bee. (1) If the nectar amount decreased to a low level or exhausted, foraging bee abandons the food source and become an unemployed bee. (2) If there are still sufficient amount of nectar in the food source, it can continue to forage without sharing the food source information with the nest mates. (3) Or it can go to the dance area to perform waggle dance for informing the nest mates about the same food source. The probability values for these options highly related to the quality of the food source.
- *Experienced foragers:* These types of forager use their historical memories for the location and quality of food sources. It can be an inspector which controls the recent status of food source already discovered. It can be a reactivated forager by using the information from waggle dance. It tries to explore the same food source discovered by itself if there are some other bees confirm the quality of same food source (*RF* in Fig. 1). It can be scout bee to search new patches if the whole food source is exhausted (*ES* in Fig. 1). It can be a recruit bee which is searching a new food source declared in dancing area by another employed bee (*ER* in Fig. 1).

The foraging behavior, learning, memorizing and information sharing characteristics of bees have recently been one of the most interesting research areas in swarm intelligence. Studies on honey bees are in an increasing trend in the literature during the last few years. An extensive literature survey and a classification of the previous algorithms on behavioral characteristics of the honey bees were presented by Baykasoğlu et al. [27].

In this paper only "foraging behavior" based optimization algorithms are considered. Lucic and Teodorovic [29–31] and Lucic [32] explored the possible applications of collective bee intelligence in solving complex traffic and transportation engineering problems. In their study, they proposed a bee system (BS) algorithm which was applied to traditional traveling salesman problems. Lucic [32], Lucic and Teodorovic [33] combined BS and fuzzy logic approach in order to obtain good solutions for stochastic vehicle routing problems. Teodorovic and Dell'Orco [34] proposed a generalization of BS algorithm called bee colony optimization (BCO) where the proposed algorithm is capable of solving deterministic combinatorial problems, as well as combinatorial problems characterized by uncertainty. Markovic et al. [35] used the BCO algorithm to solve max-routing and wavelength assignment problem in all-optical networks. Nakrani and Tovey [36] proposed a honey bee algorithm for dynamic allocation of internet services. Wedde et al. [37] introduced a bee behavior based routing protocol for routing in telecommunication network. Bianco [38] presented a mapping paradigm for large scale precise navigation that takes inspiration from the bees' large scale navigation behavior. Chong et al. [39] presented a novel approach inspired by Nakrani and Tovey [36], to solve the job shop scheduling problems. Drias et al. [40] introduced a new intelligent approach which is inspired from the behavior of real bees especially harvesting the nectar of the easiest sources of access while always privileging the richest. Quijano and Passino [41] proposed an algorithm, based on the foraging behavior of honey bees to solve resource allocation problems. Baştürk and Karaboğa [42], Karaboğa and Baştürk [43], Yang [44], Pham et al. [1] proposed different algorithms for solving continuous optimization problems based on foraging behavior of honey bees. The bees algorithm (BA) which was developed by Pham et al. [1] is a population-based search algorithm that mimics the food foraging behavior of swarms of honey bees. In its basic version, the algorithm performs a kind of neighborhood search combined with a random search and can be used for both combinatorial optimization and functional optimization. BA has been applied to several optimization problems by its initial developers, such as: training neural networks for pattern recognition [45–49], forming manufacturing cells [50], scheduling jobs for a production machine [51], finding multiple feasible solutions to a preliminary design problem [52], data clustering [53], optimizing the design of mechanical components [54], multi-objective optimization [55], tuning a fuzzy logic controller for a robot gymnast [56].

3. Modified BA for GAP

In this section the general BA framework and the principal algorithms for the initial solution and different neighborhood structures for GAP are presented. The general steps of the basic BA [1] are presented in Table 1 and each step of the modified BA for GAP is explained in Table 2.

As discussed in Pham et al. [1] the basic BA requires a number of parameters to be set, namely: number of scout bees (n), number of patches selected out of n visited points (m), number of best patches out of m selected patches (e), number of bees recruited for e best patches (nep), number of bees recruited for the other ($m - e$) selected patches (nsp), size of patches (ngh) and the stopping criterion. The algorithm starts with n scout bees being placed randomly in the search space. The fitness of the points visited by the scout bees are evaluated in step 2. In step 4, bees that have the highest fitness are chosen as "elite bees" and those sites have been visited will be chosen for neighborhood search. Then, in steps 5 and 6, the algorithm conducts searches in the neighborhood of the selected bees in terms of more bees for the e best bees. The latter can be chosen directly according to the fitness associated with the points they are visiting. Alternatively, the fitness values are used to determine the probability of the bees being selected. Searches in the neighborhood of the e best bees which represent more promising solutions are made more detailed by recruiting more bees to follow the e best bees then other selected bees. Also within scouting, differential recruitment is one of the key operations of the BA. However, in step 7, for each site only one bee with the highest fitness will be selected to form the next bee population. In nature, there is no such a restriction. This restriction is introduced to reduce the number of points to be explored. In step 8, the remaining bees in the population are assigned randomly around the search space scouting for new potential solutions. These steps are repeated until a stopping criterion is met. At the end of each iteration, the colony will have two parts to its new population – representatives from each selected patch and other scout bees assigned to conduct random searches [1,45]. We tried to partly depict the search behavior of the basic BA in Fig. 2.

In this study, some steps of the original BA algorithm are modified for solving the GAP. One of the modifications is not to use *ngh* parameter. This parameter is replaced with *EC-length* parameter of ejection chain mechanism that is devised for GAP. Another modification is related to scout bees. In the original BA algorithm scout bees continue searching in the solution space until the algorithm is terminated. However, in the modified BA an adaptive penalty coefficient mechanism is

Table 1
Pseudo code of the basic BA.

```
1  Initialize bee population with random solutions
2  Evaluate the fitness of the population
3  While (stopping criterion not met)
           //forming new bee population
4          Select elite bees
5          Select sites for neighborhood search
6          Recruit bees around the selected sites and evaluate fitness
7          Select the fittest bee from each site
8          Assign remaining bees to search randomly and evaluate their fitness
9  End While
```

Table 2
Pseudo code of the modified BA for GAP.

```
1. Parameter initialization
2. Initialize scout bees with GRASP algorithm
3. Evaluate scout bees' fitness function (for minimization)
       $fit(\sigma^2) = \sum_{j=1}^{m}\sum_{i=1}^{n} c_{ij}x_{ij} + \left(\sum_{j=1}^{m}\alpha_j \max\left\{0, \sum_{i=1}^{n} a_{ij}x_{ij} - b_j\right\}\right)$
4. Sort $_{s=1,\ldots,S}(\sigma^s)$ in increasing order and determine best $p$ solutions as employed bees
5. I = 0
6. Do
     Select best $e$ employed bees
     Assign $nep$ onlooker bees to best $e$ employed bees
     Assign $nsp$ onlooker bees to remaining $p$-$e$ employed bees
    $k$=0
    Do
      Shift
        If fit($\sigma^{shift}$) < fit($\sigma^p$) then $\sigma^p = \sigma^{shift}$
      Double shift
        If fit($\sigma^{doubleshift}$) < fit($\sigma^p$) then $\sigma^p = \sigma^{doubleshift}$
      For each onlooker assigned to employed bee {
        Ejection Chain
           If fit($\sigma^{onlooker}$) < fit($\sigma^p$) then $\sigma^{bestonlooker} = \sigma^{onlooker}$
      }
      If fit($\sigma^{bestonlooker}$) < fit($\sigma^p$) then $\sigma^p = \sigma^{bestonlooker}$
        and limitcounter ($\sigma^p$) = 0,
      else limitcounter($\sigma^p$) = limitcounter($\sigma^p$) + 1
      Update best solution
      If ($\sigma^p$ is feasible and fit($\sigma^p$) < fit($\sigma^{best}$)) then $\sigma^{best} = \sigma^p$
      If (limitcounter($\sigma^p$) > maxlimit) then generate a new scout solution with GRASP algorithm
      Update $\alpha_j$, evaluate fit($\sigma^p$)
    k = k + 1
    While (k<p)
    Sort $_{p=1,\ldots,P}(\sigma^p)$ in increasing order
    I = Calculate CPU time
   While (I < MaxCPU)
```

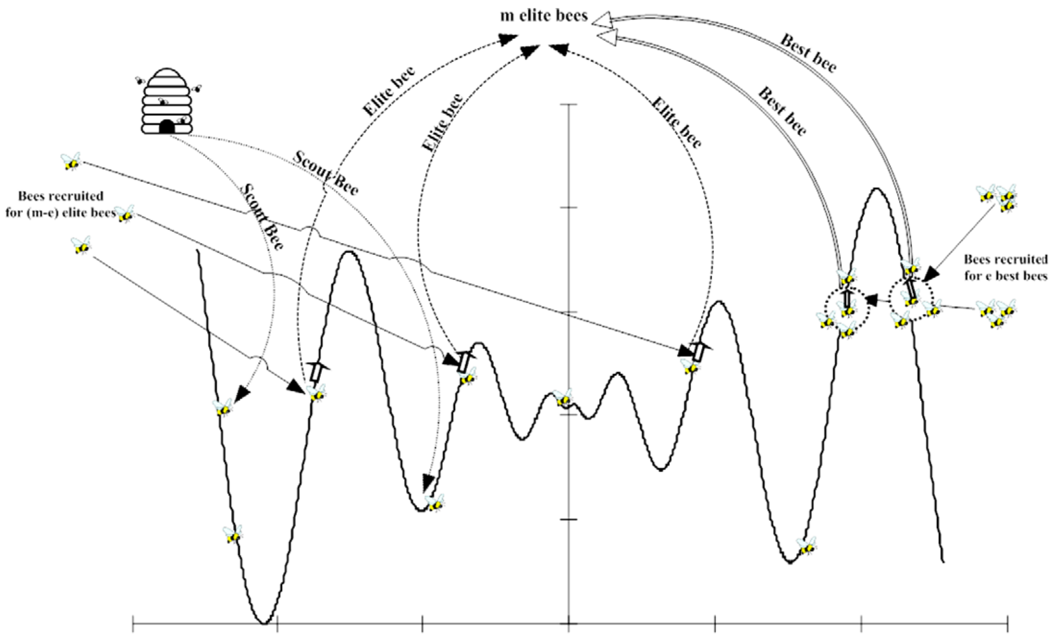

Fig. 2. Search behavior of the basic BA on a continuous function for maximization ($n = 10$ scoot bees, $m = 5$ elite bees out of $n, e = 2$ best bees out of $m, nsp = 1$ bee recruited for $m - e$ elite bees, $nep = 3$ bees recruited for e best bees).

employed. Due to starting with a small *alfa* value at the beginning, it is possible to replace the new solution with an improved solution. For that reason, instead of using scout bees which continuously perform search, new scout bees are introduced to the algorithm whenever nectar (food) is considerable decreased in a search site. In other words, a re-initialization mechanism is introduced to the algorithm for re-initializing a solution if it is not improved for some number iteration (time). In Table 2 the modified BA algorithm is presented. The following notation is used in the algorithm.

s	Scout bees $(s = 1, \ldots, S)$
p	Employed bees $(p = 1, \ldots, P)$
e	Number of best employed bees
nep	Number of onlooker bees for each $e$employed bees
nsp	Number of onlooker bees for each $P - e$ employed bees $(nsp < nep)$
MaxCPU	CPU time limit
Maxlimit	Maximum number of iteration without improvement
LimitCounter (σ^p)	Number of iteration without improvement for σ^p
EC-length	Length of ejection chain neighborhood structure
σ^p	Solution of pth employed bee
fit (σ^p)	Fitness function value of pth employed bee
σ^{ls}	A neighbor solution found by local search (*ls* = shift, double shift, onlooker, best onlooker)
σ^{best}	The best solution
α_j	The cost of using one unit of overloaded capacity of jth agent

Modified BA starts with parameter (s, p, e, *nep*, *nsp*, *EC-length*, *MaxCPU*, *Maxlimit*, α_j) initialization and continues by generating S number of initial solutions by scout bees. Initial solutions are generated by employing GRASP algorithm which is explained in the following sections of this paper. P number of good solutions within the set of generated solutions is determined as employed bees. e number of solutions are selected from the set P as the best solutions. *nep* number of onlooker bees are directed to these best solutions in order to carry out a more detailed neighborhood search. Fewer onlooker bees are directed to the remaining $P - e$ solutions. Shift and double shift neighborhood mechanisms are applied to each employed bee for local search. Employed bee is updated if an improved solution is found. Onlooker bees which were assigned to each employed bee generate neighborhood solutions via ejection chain neighborhood structure. The best onlooker bee is compared with the original employed bee. If it is better the employed bee is updated. Employed bees are compared with the best solution, if the necessary conditions are satisfied (*feasible and better than the previous best solution*) the best solution is updated. If the employed bee is not improved until *Maxlimit* iterations, a new scout bee is generated by using the GRASP algorithm. If no feasible solution is found during all these operations α_j values are increased by using an algorithm which is given in Table 7. If at least one feasible solution is found, α_j values are reduced by using the same algorithm. The details of the important substeps of the modified BA are explained in the following sub-sections. The analogy between the modified BA and the bees in nature is also shown in Table 3 in order to gain a better insight about the present algorithm.

3.1. Bee colony initialization

Initial bee colony is constructed by using the GRASP algorithm [18]. The greedy heuristic constructs a solution as follows:

- At each step, a next task to be assigned is selected.
- The agent (*the selected task is going to be assigned to*) is determined.
- Repeat these two steps until all tasks have been assigned to an agent.

In the GRASP algorithm the choice is probabilistic bias to a probability function. This function is updated at each iteration in a reinforcement way by using the features of good solutions. The main execution steps of the GRASP algorithm is summarized as shown in Table 4.

Table 3
Bees in nature versus modified BA.

	Bees in nature	Modified BA
Scout bee	Random search for food resources	Random search and re-initialization
Employed bee	Bees which found good food resources	Good solutions
Onlooker bee	Obtains information from waggle dance of employed bee to move to good food areas	Solutions which are assigned to (generated around) good solutions in order to carry out detailed search.
Waggle dance	Information sharing	Determining good solution area for detailed search
Abandon from food source	Leaving feeding areas where nectar is finished	Leaving the search area where a solution cannot be improved further

Table 4
The GRASP algorithm.

```
1.   Let, $S_j = \emptyset,\ \forall j = 1,\dots,m$ ($S_j$ is the set of task assigned to agent $j$)
2.   Construct a list of agents for each task, $L_i$, initially $L_i = \{1,\dots,m\}\ \forall i$
3.   Consider an arbitrary order of the tasks, $i = 1$
4.   While (not all tasks have been assigned) repeat
     4.1 Choose randomly an agent $j^*$ from $L_i$ following the probability function that depends on the resource of agent $j$ and the
         resource need by task $i$:
         $p_{ij} = \frac{a_j/b_{ij}}{\sum_{l \in L_i} a_l/b_{il}},\quad j \in L_i$
         (The agent with minimal cost has greater probability to be chosen)
     4.2 Assign task $i$ to agent $j^*: S_{j^*} = S_{j^*} \cup \{i\}$. Let $i = i + 1$ and if $\sum_{i \in S_{j*}} b_{ij*} > a_{j*}$ remove $j^*$ from any list. Repeat step 4 (note that the
         capacity constraint can be violated)
5.   Let $\sigma(i) = j$ if $i \in S_j$
```

3.2. Neighborhood structures

3.2.1. Shift

This type of neighbor solution is obtained from original solution by changing the agent assignment of one task. The algorithm steps are summarized in Table 5.

3.2.2. Double shift

This neighborhood structure is the special case of the ejection chain neighborhood. Since the two shift moves are performed in double shift, this is the *EC-length* = 2 state of the ejection chain. Double shift neighborhood contains the swap neighborhood, which is the interchange of agents of two different tasks assigned to, within its scope. In the ejection chain neighborhood, task for each shift move is selected from *B list*. In double shift neighborhood, new shift move is determined by using the set of all tasks because there is no restriction to achieve a new shift move. This neighborhood structure is the simplified version of double shift mechanism developed by Yagiura et al. [17].

3.2.3. Ejection chain

A neighbor solution is obtained by performing the multiple shift moves whose length is specified as chain length. A simple explanation of the neighborhood structure and main steps of the algorithm are presented here, but detailed information can be obtained from Yagiura et al. [17]. Assume that task i_0 is ejected from agent $\sigma(i_0)$ as a free task where $\sigma(i_0)$ denotes the agent that task i_0 is assigned to. The amount of resource of $\sigma(i_0)$ is increased by this ejection move. *Avail* is defined as the resulting amount of resource as shown in the following equation:

$$avail(i) = \begin{cases} a_{i,\sigma(i)} - p_{\sigma(i)}(\sigma) & \text{if} \quad a_{i,\sigma(i)} > p_{\sigma(i)}(\sigma) \\ a_{i,\sigma(i)} & \text{otherwise} \end{cases}$$

Assuming that task i_1 is the task whose shift into $\sigma(i_0)$ is most profitable among the tasks satisfying $a_{i1,\sigma(i_0)} \leqslant avail(i_0)$. Task i_1 is shifted into agent $\sigma(i_0)$. This is called as reference structure. After this first ejection move, the free task i_0 will be tried to assign into some other agents according to its effect on fitness function as shown in Table 6 (Step 4). This is called as the trial move. The next ejection move is applied to the previous reference structure, not to the solutions generated by the trial

Table 5
Shift neighborhood.

```
Shift ($\sigma^p$)
1.   Let $S = \{i | i \in \{1,\dots,n\}\}$, $k = 1$, $\sigma^{\mathbf{shift}} = \sigma^p$
2.   If $S = \emptyset$ then stop; otherwise $\sigma' = \sigma^p$, $i_k$ is ejected from $\sigma'$
     $S = S - \{i_k\}$
3.   Let $j^*$ be the agent $j$ that minimizes
     $c_{i_k j} + \alpha_j \max\left\{0, \left(\sum_{i \in I, \sigma(i)=j} a_{ij}\right) + a_{i_k j} - b_j\right\}$ among all agents $j \in J / \{\sigma(i_k)\}$
4.   Assign $i_k$ to $j^*$, output $\sigma'$, calculate fit($\sigma'$)
5.   If fit($\sigma'$) < fit($\sigma^{\text{shift}}$) then $\sigma^{\text{shift}} = \sigma'$
6.   k = k + 1, return to Step 2
7.   Output $\sigma^{\text{shift}}$
```

Table 6
Ejection chain neighborhood structure.

```
Ejection chain ($\sigma$)
1.  Let $S = \varnothing$
2.  If $S = I'$ or l = EC-length stop; otherwise randomly choose a $i_0 \in I' \setminus S$, let $S = S \cup \{i_0\}$ and $\sigma' = \sigma$ (Job $i_0$ is ejected from $\sigma(i_0)$)
3.  Let j* be the agent j that minimizes $c_{i_0 j} + \alpha_j \max\left\{0, \left(\sum_{i\in I,\sigma(i)=j} a_{ij}\right) + a_{i_{01}j} - b_j\right\}$ among all agents $j \in J/\{\sigma(i_0)\}$, and let $l = 0$.
4.  If $B(i_l) \setminus \{i_k | k \leqslant l\} = \varnothing$, return to Step 2; otherwise let $l = l + 1$ and proceed to Step 5
5.  Randomly choose $i_l \in B(i_{l-1}) \setminus \{i_k | k \leqslant l-1\}$ and let $\sigma'(i_l) = \sigma(i_{l-1})$ (an ejection move of job $i_l$). Then execute the following Steps (a) and (b) (two trial moves)
        (a) Let $\sigma'(i_0) = \sigma(i_l)$ ($i_0$ is inserted into $\sigma(i_l)$), and output $\sigma'$
        (b) Let $\sigma'(i_0) = j^*$ ($i_0$ is inserted into $i^*$), and output $\sigma'$
6.  Return to Step 4.
```

moves. Same steps are repeated until the stopping criterion is satisfied. The general mechanism of the ejection chain neighborhood is presented in Table 6.

$$score(i,j) = c_{ij}$$
$$I' = \{k \in I | \exists h \in I s.t. a_{h,\sigma(k)} \leqslant avail(k) \text{ and } \sigma(h) \neq \sigma(k)\}$$
$$bestscore(i) = \min\{score(k,\sigma(i)) | k \in I', \sigma(k) \neq \sigma(i) \text{ and } a_{k,\sigma(i)} \leqslant avail(i)\}$$
$$B(i) = \{k \in I' | score(k,\sigma(i)) = bestscore(i), \sigma(k) \neq \sigma(i) \text{ and } a_{k,\sigma(i)} \leqslant avail(i)\}$$

Yagiura et al. [17] applied shift, double shift and ejection chain neighborhood structures within a tabu search algorithm which iterates by improving a single solution. In this study, local search strategies are used differently from the previous works.

- In the present study, costs (c_{ij}) are determined as score function. Although some of the problems resulted better solutions or CPU times by using other score functions as explained in Yagiura et al. [17], minimum cost value is used in all problems for simplicity.
- Shift algorithm is executed for each job and the best improvement is accepted as the new solution.
- Double shift mechanism is simplified by implementing as the ejection chain with length two but it allows the swap trial move. It is executed for each job in I' set to determine the best improving double shift move.
- Ejection chain is restricted by a predefined ejection length. Because of the complexity of this neighborhood structure, first improvement is received as new solution. Onlooker bees assigned to the best solutions are responsible for performing a detailed neighborhood search. Each onlooker bee is assigned to the same solution, which starts with a different job while executing ejection chain and proceeds until a better solution is found.
- Adaptive control of penalty coefficients (α_j) which are explained in Table 7 is used. In this study initial values of α_j are determined as 1 to recognize the convergence capability of the algorithm.
- BA follows a population-based search strategy which denotes a parallel structure. This characteristic of the search strategy is expected to be useful as the problem complexity increases.

Table 7
Adaptive control of α_j.

1. If there is no feasible solution found in onlooker neighbors, α_j are increased for all $j \in J$ by

$$\alpha_j = \begin{cases} \alpha_j(1 + \Delta \cdot q_j^{inc}(\sigma)), & \alpha_j > 0 \\ \Delta \cdot q_j^{inc}(\sigma) \min\limits_{h\in J}\{b_h\alpha_h | b_h\alpha_h > 0\}/b_j, & \text{otherwise} \end{cases}$$

Where

$$\Delta = \begin{cases} \frac{stepsizeinc}{\max_{j\in J}|q_j^{inc}(\sigma)|}, & \text{if} \quad \max\limits_{j\in J}|q_j^{inc}(\sigma)| > 0 \\ 0, & \text{otherwise} \end{cases}$$

2. (Otherwise) If at least one feasible solution is found within onlooker neighbors, all α_j are decreased by using the same equations except that $q^{dec}(\sigma)$ instead of $q^{inc}(\sigma)$ and *stepsizedec* instead of *stepsizeinc*.

$$p_j(\sigma) = \max\left\{0, \left(\sum_{i\in I,\sigma(i)=j} a_{ij}\right) - b_j\right\}, \quad q_j^{inc}(\sigma) = p_j(\sigma)/b_j, \quad q_j^{dec}(\sigma) = \begin{cases} -1, & \text{if} \quad p_j(\sigma) = 0 \\ 0, & \text{otherwise} \end{cases}$$

4. Computational study

The proposed BA is coded in C# and tested on benchmark problems on an Intel Pentium CoreDuo PC with 1.6 GHZ CPU and 512 MB RAM. The benchmark problems range from 5 agents–100 tasks to 20 agents–200 tasks. The problems can be divided into two groups: gapa–gapb/*easy* and gapc–gapd/*difficult*. These test problems are taken from the OR-Library (http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html) and consist of four types:

Type A: r_{ij} are integers from U(5,25), c_{ij} are integers from U(10,50) and $b_i = 0.6(n/m)15 + 0.4R$, where $R = \max_{i \in I} \sum_{j \in J, I_j = i} r_{ij}$ and $I_j = \min[i | c_{ij} \leqslant c_{kj}, \ \forall k \in I]$.
Type B: r_{ij} and c_{ij} are the same as Type A and b_i is set to 70% of the value given in Type A.
Type C: r_{ij} and c_{ij} are the same as Type A and $b_i = 0.8\sum_{j=J} r_{ij}/m$.
Type D: r_{ij} are integers from U(1,100), $c_{ij} = 111 - r_{ij} + e$, where e are integers from U(−10,10) and $b_i = 0.8\sum_{j=J} r_{ij}/m$.

For each problem type, one problem for each agent/job combination (m = 5, 10, 20 and n = 100, 200) was generated, giving a total of 24 problems.

4.1. Fitness function

In computing the value of fitness function a penalty term is added to the fitness function in order to convert the constrained problem into an unconstrained one. While constructing initial solutions by using the GRASP algorithm and generating neighbor solutions via shift, double shift and ejection chain algorithms, the proposed approach may produce infeasible solutions. Therefore, there is an additional term in the fitness function which is determined by penalizing the infeasible solutions with $\alpha_j(\alpha_j > 0)$. Fitness function is formally defined as follows: $fit(\sigma^S) = \sum_{j=1}^{m}\sum_{i=1}^{n} c_{ij}x_{ij} + \left(\sum_{j=1}^{m} \alpha_j \max\{0, \sum_{i=1}^{n} b_{ij}x_{ij} - b_j\}\right)$. The first term in the fitness function denotes the total cost of assignment of tasks to agents. The second term is defined as an additional penalty function for minimization. α_j represents the cost of using one unit of overloaded capacity of jth agent. Initial values of α_j's are set by the user. If a solution is not feasible the second term will be positive and therefore the search will be directed to a feasible solution. If the capacity is not exceeded, this term will be zero to ensure not penalized. The parameter α_j can be increased during the run to penalize infeasible solutions and drive the search to feasible ones which means the adaptive control of the penalty costs.

4.2. Adaptive control of the penalty coefficient

Updating stage is adapted from Yagiura et al. [17] by using the procedure and equations which are given in Table 7. After the completion of one iteration for neighborhood structures (shift, double shift, ejection chain), α_j values are updated. Step-size increment and step-size decrement are determined as 0.01 and 0.1, respectively. These values provide that if a feasible solution is found, a detailed search is performed around its neighborhood.

4.3. Parameters setting

Parameters of the proposed bees algorithm are defined as follows:

- Number of scout bees (S)
- Number of employed bees (P)
- Number of best employed bee (e)
- Number of onlooker bees for each e employed bees (*nep*)
- Number of onlooker bees for each $P - e$ employed bees (*nsp*)
- Length of ejection chain neighborhood structure (*EC-length*)
- CPU time limit (*MaxCPU*)
- Maximum number of iteration without improvement (*Maxlimit*)
- Initial value of penalty coefficient (α_j)
- $stepsizedec > 0$
- $0 < stepsizeinc < 1$

According to the structure of the test problems, two different parameter sets are determined as shown in Table 8. These parameters are set based on the general guidelines given in the literature and the author's computational experiments with the proposed algorithm.

Minimum, average, maximum, standard deviation and the number of best for five different runs that are obtained from the proposed BA are presented in Table 9. The proposed BA results are compared with 11 different algorithms collected from the literature in Table 10. Since the best results in literature are generally obtained from five different runs, proposed algorithm is also analyzed with the same number of runs for more fair comparison. However, there is no standard approach for

Table 8
Parameters setting.

Parameters	gapa–gapb/*easy*	gapc–gapd/*difficult*
S	100	200
P	5	10
e	2	2
nep	10	10
nsp	1	1
EC-length	20	75
MaxCPU	3000	6000
Maxlimit	50	50
α_j	1	1
stepsizedec	0.1	0.1
stepsizeinc	0.01	0.01

Table 9
The results of modified BA for GAP.

Problem	Min	Average	Max	Std. dev.	Number of best
gapa1	*1698*	*1698*	*1698*	*0*	*5/5*
gapa2	*3235*	*3235*	*3235*	*0*	*5/5*
gapa3	*1360*	*1360*	*1360*	*0*	*5/5*
gapa4	*2623*	*2623*	*2623*	*0*	*5/5*
gapa5	*1158*	*1158*	*1158*	*0*	*5/5*
gapa6	*2339*	*2339*	*2339*	*0*	*5/5*
gapb1	*1843*	*1843*	*1843*	*0*	*5/5*
gapb2	*3552*	*3552*	*3552*	*0*	*5/5*
gapb3	*1407*	*1407*	*1407*	*0*	*5/5*
gapb4	*2827*	*2827*	*2827*	*0*	*5/5*
gapb5	*1166*	*1166*	*1166*	*0*	*5/5*
gapb6	*2339*	*2339*	*2339*	*0*	*5/5*
gapc1	1931	1931	1931	0	5/5
gapc2	3456	3456.6	3457	0.547723	2/5
gapc3	1402	1402	1402	0	5/5
gapc4	2806	2806.6	2807	0.547723	2/5
gapc5	1243	1243.6	1244	0.547723	2/5
gapc6	2392	2392.6	2393	0.547723	2/5
gapd1	6353	6354.4	6356	1.341641	2/5
gapd2	12744	12746.2	12748	1.788854	1/5
gapd3	6356	6358.8	6362	2.588436	1/5
gapd4	12442	12445.2	12447	2.04939	1/5
gapd5	6221	6226.6	6232	4.615192	1/5
gapd6	12276	12280	12284	2.915476	1/5

determining this number. In Tables 9 and 10 computational times performance of the proposed BA and other algorithms are also presented. In the present work, initial values of (α_j) are set to 1 for all agents. So, CPU times include the time for adjustment of α_j's to its best values for each agent.

In Table 10, results obtained from a commercial solver CPLEX 6.5 (*Commercial General Purpose Branch-And-Cut System for Solving Integer Linear Programs*) is also listed for comparing the solution qualities of the proposed BA with an exact procedure. CPLEX 6.5 results were originally presented by Yagiura et al. [24]. As it can be seen from Table 10 the proposed BA is able to generate better solutions than CPLEX 6.5 for difficult problems.

It is not easy to directly compare the performances of different algorithms due to many reasons (*computing environments, testing conditions, parameter optimizations etc.*). However, a direct comparison with the published data is usually the only way to judge the performances of the proposed algorithms. In order to have an idea about the performance of the proposed BA on solving GAP we applied a simple ranking procedure. In Table 10 the best solutions are bolded and the ranking of a solution is indicated within parentheses with its rank value. This information is aggregated and depicted in Fig. 3 where the total number of times an algorithm is ranked as 1, 2, 3 and $\geqslant 4$ can be easily visualized. Based on the results depicted in Fig. 3 PREC (*Yagiura et al. [24]'s path relinking approach based on ejection chains*) is the best performing algorithm. The proposed BA, TSEC and NBB can be classified as the second best performers in terms of solution quality. CPLEX 6.5 performance is also considerable good in *gapc* problems, however, its performance on *gabd* problems is not good. The computational time (CPU) performances of the best performer algorithms (PREC, BA, TSEC, NBB) are also considered for a direct comparison. For that purpose CPU performances of these algorithms are plotted in Fig. 4 (*as we have already mentioned this comparison might be very rough*). Based on Fig. 4 we can say that the proposed BA has a considerable better CPU performance within the best performing algorithms. We should also mention here that, in PREC and TSEC the CPU times are measured after optimizing

Table 10
Comparison of results for gapa–gapd.

Problem	Proposed BA (5 run)		DF (30 runs)		CGA (10 runs)		RLS		APT (1 run)		ACO (10 runs)	
	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost
gapa1	0.18	**1698**(1)[a,b]	–	–	253	**1698**(1)	–	–	–	–	0.17	**1698**(1)
gapa2	0.95	**3235**(1)	–	–	502	**3235**(1)	–	–	–	–	0.61	**3235**(1)
gapa3	0.27	**1360**(1)	–	–	308	**1360**(1)	–	–	–	–	0.69	**1360**(1)
gapa4	1.31	**2623**(1)	–	–	930	**2623**(1)	–	–	–	–	126.58	**2623**(1)
gapa5	0.41	**1158**(1)	–	–	350	**1158**(1)	–	–	–	–	6.71	**1158**(1)
gapa6	1.81	**2339**(1)	–	–	860	**2339**(1)	–	–	–	–	158.5	2341(2)
gapb1	5.97	**1843**(1)	–	**1843**(1)	302	**1843**(1)	–	–	10.0	**1843**(1)	191.25	1846(2)
gapb2	45.99	**3552**(1)	–	**3552**(1)	432	3601(4)	–	–	121.9	3553(2)	2605.55	3561(3)
gapb3	0.36	**1407**(1)	–	**1407**(1)	165	1410(2)	–	–	7.3	**1407**(1)	115.78	**1407**(1)
gapb4	315.04	**2827**(1)	–	2828(2)	949	2831(4)	–	–	37.6	2829(3)	5482.81	2849(5)
gapb5	1.12	**1166**(1)	–	**1166**(1)	474	**1166**(1)	–	–	11.4	**1166**(1)	613.28	**1166**(1)
gapb6	28.65	**2339**(1)	–	2340(2)	683	2347(3)	–	–	132.7	2340(2)	2865.04	2350(4)
gapc1	3.61	**1931**(1)	0.6	**1931**(1)	195	1941(2)	137.1	1942(3)	6.6	**1931**(1)	192.13	**1931**(1)
gapc2	18.09	**3456**(1)	3.7	3457(2)	405	3460(4)	1693.8	3467(6)	146.1	3458(3)	429.09	3464(5)
gapc3	5.81	**1402**(1)	3.0	**1402**(1)	203	1423(5)	178.3	1407(4)	31.5	**1402**(1)	122.43	1406(3)
gapc4	488.89	**2806**(1)	100.5	2807(2)	498	2815(4)	1086.0	2818(5)	104.3	2810(3)	1112.31	2825(6)
gapc5	23.16	**1243**(1)	21.6	**1243**(1)	479	1244(2)	309.6	1247(5)	47.5	1244(2)	193.85	1246(4)
gapc6	646.18	2392(2)	137.4	**2391**(1)	1059	2397(4)	2694.8	2405(5)	146.4	2396(3)	1119.25	2411(6)
gapd1	916.03	**6353**(1)	62.6	6357(3)	259	6479(7)	2459.2	6476(6)	71.5	6365(5)	1.81	6625(8)
gapd2	122.41	12744(3)	95.5	12747(5)	1253	12823(8)	11106.9	12923(9)	318.1	12747(5)	3.31	13197(10)
gapd3	538.29	6356(4)	107.2	6355(3)	497	6390(8)	5587.3	6469(9)	77.3	6372(5)	174.46	6613(10)
gapd4	743.95	12442(3)	129.2	12457(5)	1321	12634(8)	47538.7	12746(9)	105.2	12457(5)	1155.21	13024(10)
gapd5	1704.46	6221(5)	111.0	6220(4)	974	6280(9)	13656.8	6358(10)	108.8	6267(8)	189.92	6484(11)
gapd6	922.94	12276(3)	120.7	12351(7)	2158	12471(10)	116969.0	12617(11)	308.7	12333(5)	1270.43	12951(12)

Problem	CRH-GA (10 runs)		TSEC (5 runs)		PREC (5 runs)		PRSS (1 run)		NBB		CPLEX 6.5	
	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost
gapa1	1.0	**1698**(1)	–	–	–	–	–	–	0.12	**1698**(1)		**1698**(1)
gapa2	25.9	**3235**(1)	–	–	–	–	–	–	0.06	**3235**(1)	–	**3235**(1)
gapa3	97.1	**1360**(1)	–	–	–	–	–	–	0.05	**1360**(1)	–	**1360**(1)
gapa4	10.2	**2623**(1)	–	–	–	–	–	–	0.16	**2623**(1)	–	**2623**(1)
gapa5	22.2	**1158**(1)	–	–	–	–	–	–	0.16	**1158**(1)	–	**1158**(1)
gapa6	24.2	**2339**(1)	–	–	–	–	–	–	0.28	**2339**(1)	–	**2339**(1)
gapb1	51.2	**1843**(1)	0.81	**1843**(1)	0.95	**1843**(1)	1.41	**1843**(1)	4.18	**1843**(1)		**1843**(1)
gapb2	165.4	3553(2)	3.13	**3552**(1)	2.35	**3552**(1)	2.59	**3552**(1)	14.66	**3552**(1)		**3552**(1)
gapb3	30.9	**1407**(1)	0.22	**1407**(1)	0.33	**1407**(1)	0.18	**1407**(1)	0.11	**1407**(1)		**1407**(1)
gapb4	381.8	2829(3)	16.00	**2827**(1)	18.97	**2827**(1)	91.64	**2827**(1)	235.85	**2827**(1)		**2827**(1)
gapb5	189.3	**1166**(1)	9.00	**1166**(1)	3.64	**1166**(1)	7.24	**1166**(1)	0.17	**1166**(1)		**1166**(1)
gapb6	378.5	2340(2)	5.19	**2339**(1)	11.53	**2339**(1)	18.21	**2339**(1)	88.98	2340(2)		**2339**(1)
gapc1	39.2	**1931**(1)	0.6	**1931**(1)	0.52	**1931**(1)	1.34	**1931**(1)	0.05	**1931**(1)	2	**1931**(1)
gapc2	320.0	3457(2)	3.7	**3456**(1)	12.09	**3456**(1)	2.12	**3456**(1)	30.43	**3456**(1)	35	**3456**(1)
gapc3	54.1	1403(2)	3.0	**1402**(1)	3.20	**1402**(1)	1.86	**1402**(1)	7.25	**1402**(1)	9	**1402**(1)
gapc4	304.6	2807(2)	403.8	**2806**(1)	2710.87	**2806**(1)	48.47	2807(2)	312.64	**2806**(1)	249	**2806**(1)
gapc5	289.6	**1243**(1)	22.5	**1243**(1)	35.83	**1243**(1)	17.99	1245(3)	90.19	**1243**(1)	8	**1243**(1)
gapc6	1140.1	2396(3)	301.8	**2391**(1)	1268.83	**2391**(1)	191.32	2394(2)	968.66	**2391**(1)	296	**2391**(1)
gapd1	327.4	6365(5)	649.2	**6353**(1)	83.62	**6353**(1)	14.24	6356(2)	349.93	**6353**(1)	43	6358(4)
gapd2	814.7	12767(7)	3564.8	12743(2)	5712.49	**12742**(1)	249.89	12745(4)	2937.03	12745(4)	62	12750(6)
gapd3	472.6	6373(6)	2440.7	6349(2)	988.07	**6348**(1)	74.12	6373(6)	2831.47	6349(2)	132	6381(7)
gapd4	1909.7	12536(7)	5829.9	12440(2)	5520.51	**12433**(1)	246.32	12468(6)	1896.80	12447(4)	27	12457(5)
gapd5	902.3	6259(7)	1591.9	6206(3)	2254.88	**6192**(1)	129.33	6235(6)	2829.38	6200(2)	60	6280(9)
gapd6	3825.7	12386(8)	1757.7	12277(4)	5777.10	**12245**(1)	518.30	12334(6)	2375.42	12263(2)	297	12393(9)

Notes: Proposed BA: the proposed algorithm.
DF: Diaz and Fernandez [16]'s tabu search.
CGA: Lorena et al. [57]'s constructive genetic algorithm.
RLS: Lourenço and Serra [18]'s max–min ant system combined with local search and tabu search.
APT: Alfantari et al. [19]'s path relinking algorithm.
ACO: Randall [25]'s ant colony optimization.
CRH-GA: Feltl and Raidl [26]'s genetic algorithm with constraint-ratio heuristic.
TSEC: Yagiura et al. [17]'s tabu search based on ejection chains.
PREC: Yagiura et al. [24]'s path relinking approach based on ejection chains.
PRSS: Yagiura et al. [23]'s path relinking approach based on shift and swap.
NBB: Nauss [9]'s special purpose branch and bound with feasible solution generators.
[a] The ranking of the algorithm (i.e. BA ranks number (1) within the compared algorithms).
[b] The best solutions are bolded.

the α_j parameters. On the other hand, in the proposed BA CPU times include the time for adjustment of α_j's to its best values for each agent. So we can say that computational time requirements of PREC and TSEC is much more that the proposed BA.

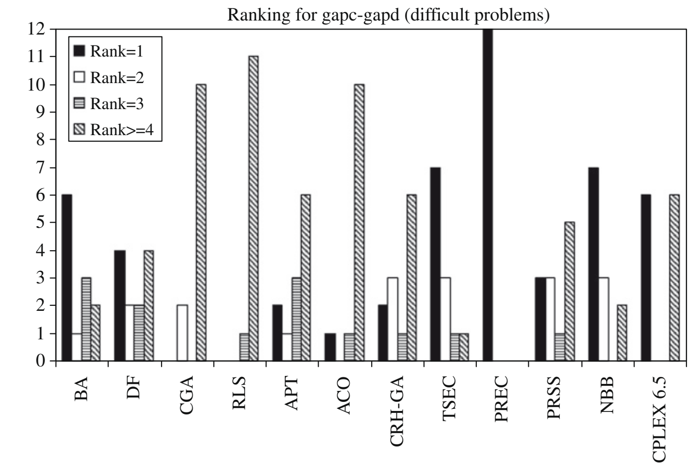

Fig. 3. Ranking of algorithms based on solution quality.

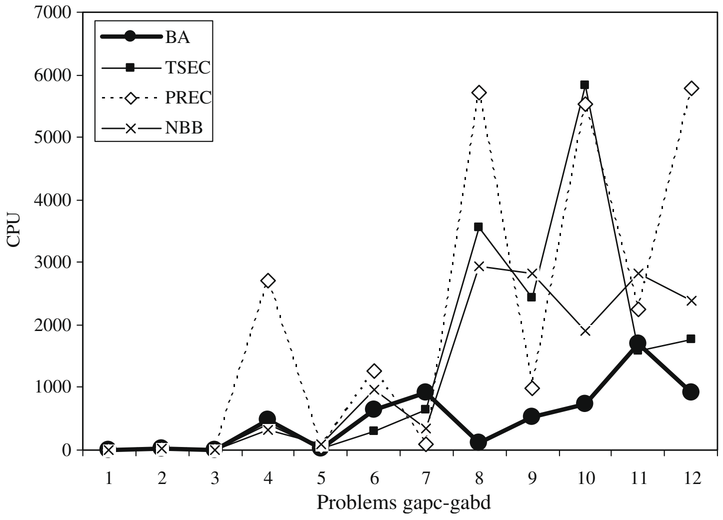

Fig. 4. CPU time performance of best performing algorithms.

This paper presents one of the first approaches in testing the performance of bees algorithm on a complex NP-hard integer optimization problem. The results obtained from the extensive computational study shows that the proposed BA has a big potential for solving complex assignment problems. We should also mention here that in the present study no serious parameter optimization for the proposed BA is carried out as this work is scheduled for future work. We expect that the performance of the proposed BA can be further improved after a detailed parameter optimization.

5. Conclusions

In this paper a relatively new member of swarm intelligence based meta-heuristics that is known as “bees algorithm, BA” is explained. Most of the studies on BA is carried out in last two years and researchers mainly concentrated on continuous optimization and TSP problems in the literature. Previous work has presented that bee inspired algorithms have a very promising potential for modeling and solving complex optimization problems. But there is still a long way to go in order to fully utilize the potential of bee inspired algorithms. Such an attempt is made in this paper to present the performance of bee inspired algorithm, BA on a NP-hard problem which is known as generalized assignment problem, GAP. The proposed bee algorithm is a modified version of the basic “bees algorithm” that was initially proposed by Pham and his colleagues. The modified BA is found very effective in solving small to medium sized generalized assignment problems. Actually, the proposed algorithm easily found all of the optimal solutions for smaller size problems. The proposed algorithm is also found very effective for solving larger size and tightly constrained generalized assignment problems. These problems are very complex, therefore their solution can be considered as a very good indicator for the potential of the bee based algorithms. Based on our computational study we have observed that the proposed BA has a potential for solving GAP. In the present paper no parameter optimization for BA is carried out. We believe that the performance of the proposed BA can be further improved for larger size generalized assignment problems if a careful parameter optimization is carried out. This is scheduled as a future work.

Acknowledgement

The second (corresponding) author is grateful to Turkish Academy of Sciences (TÜBA) for supporting his scientific studies.

References

[1] D.T. Pham, E. Koç, A. Ghanbarzadeh, S. Otri, S. Rahim, M. Zaidi, The bees algorithm – a novel tool for complex optimisation problems, in: Proceedings of the Second International Virtual Conference on Intelligent Production Machines and Systems, 2006a, pp. 454–461.

[2] S. Martello, P. Toth, An algorithm for the generalized assignment problems, in: J.P. Brans (Ed.), Operational Research, North-Holland, 1981, pp. 589–603.

[3] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, New York, 1990.

[4] D. Cattrysse, Set partitioning approaches to combinatorial optimization problems, Ph.D. Dissertation, Katholieke Universiteit Leuven, Centrum Industrieel Beleid, Belgium, 1990.

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

[6] G.T. Ross, P.M. Soland, A branch and bound based algorithm for the generalized assignment problem, Mathematical Programming 8 (1975) 91–103.

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

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

[9] R.M. Nauss, Solving the generalized assignment problem: an optimizing and heuristic approach, Informs Journal of Computing 15 (2003) 249–266.

[10] I.H. Osman, Heuristics for the generalized assignment problem: simulated annealing and tabu search approaches, OR Spektrum 17 (1995) 211–225.

[11] P.C. Chu, J.E. Beasley, A genetic algorithm for the generalized assignment problem, Computers and Operations Research 24 (1997) 17–23.

[12] M. Racer, M.M. Amini, A robust heuristic for the generalized assignment problem, Annals of Operations Research 50 (1994) 487–503.

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

[14] M. Yagiura, T. Yamaguchi, T. Ibaraki, A variable-depth search algorithm for the generalized assignment problem, in: S. Vob, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, Boston, 1999, pp. 459–471.

[15] M. Laguna, J.P. Kelly, J.L. Gonzalez-Velarde, F. Glover, Tabu search for the multilevel generalized assignment problem, European Journal of Operational Research 82 (1995) 176–189.

[16] J.A. Diaz, E. Fernandez, A tabu search heuristic for the generalized assignment problem, European Journal Operational Research 132 (2001) 22–38.

[17] M. Yagiura, T. Ibaraki, F. Glover, An ejection chain approach for the generalized assignment problem, Informs Journal of Computing 16 (2004) 131–151.

[18] H.R. Lourenço, D. Serra, Adaptive search heuristics for the generalized assignment problem, Mathware and Soft Computing 9 (2002) 209–234.

[19] L. Alfandari, A. Plateau, P. Tolla, A two-phase path relinking algorithm for the generalized assignment problem, in: Proceedings of the Fourth Metaheuristic International Conference, Porto, Portugal, July 16–20, 2001, pp. 175–179.

[20] L. Alfandari, A. Plateau, P. Tolla, A two-phase path relinking algorithm for the generalized assignment problem, Technical Report No.: 378, CEDRIC, CNAM, 2002.

[21] L. Alfandari, A. Plateau, P. Tolla, A path relinking algorithm for the generalized assignment problem, in: M.G.C. Resende, J.D. Sousa (Eds.), Metaheuristics: Computer Decision-Making, Kluwer Academic Publishers, Boston, 2004, pp. 1–17.

[22] M. Yagiura, T. Ibaraki, F. Glover, An effective metaheuristic algorithm for the generalized assignment problem, in: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Tucson, Arizona, USA, 2001.

[23] M. Yagiura, T. Ibaraki, F. Glover, A path relinking approach for the generalized assignment problem, in: Proceedings of International Symposium on Scheduling, Hamamatsu, Japan, 2002, pp. 105–108.

[24] M. Yagiura, T. Ibaraki, F. Glover, A path relinking approach with ejection chains for the generalized assignment problem, European Journal of Operational Research 169 (2006) 548–569.

[25] M. Randall, Heuristics for ant colony optimisation using the generalised assignment problem, in: Proceedings of IEEE Congress on Evolutionary Computation, Portland, Oregon, USA, 2004, pp. 1916–1923.

[26] H. Feltl, G.R. Raidl, An improved hybrid genetic algorithm for the generalized assignment problem, in: Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004, pp. 990–995.

[27] A. Baykasoğlu, L. Özbakır, P. Tapkan, Artificial bee colony algorithm and its application to generalized assignment problem, in: F.T.S. Chan, M.K. Tiwari (Eds.), Swarm Intelligence: Focus on Ant and Particle Swarm Optimization, I-Tech Education and Publishing, Vienna, Austria, 2007, pp. 113–144.

[28] T.D. Seeley, The Wisdom of the Hive, Harvard University Press, Cambridge, MA, 1995.

[29] P. Lucic, D. Teodorovic, Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence, in: Proceedings of Triennial Symposium on Transportation Analysis, Sao Miguel, Azores Islands, 2001, pp. 441–445.

[30] P. Lucic, D. Teodorovic, Transportation modeling: an artificial life approach, in: Proceedings of the 14th International Conference on Tools with Artificial Intelligence, 4–6 November 2002, Washington, DC, USA, 2002, pp. 216–223.

[31] P. Lucic, D. Teodorovic, Computing with bees: attacking complex transportation engineering problems, International Journal on Artificial Intelligence Tools 12 (3) (2003) 375–394.

[32] P. Lucic, Modeling transportation problems using concepts of swarm intelligence and soft computing, Ph.D. Dissertation, Civil Engineering, Faculty of the Virginia Polytechnic Institute and State University, 2002.

[33] P. Lucic, D. Teodorovic, Vehicle routing problem with uncertain demand at nodes: the bee system and fuzzy logic approach, in: J.L. Verdegay (Ed.), Fuzzy Sets in Optimization, Springer-Verlag, Berlin Heidelbelg, 2003, pp. 67–82.

[34] D. Teodorovic, M. Dell'Orco, Bee colony optimization – a cooperative learning approach to complex transportation problems, in: Proceedings of the 10th EWGT Meeting, Poznan, 13–16 September 2005, pp. 51–60.

[35] G.Z. Markovic, D. Teodorovic, V.S. Acimovic-Raspopovic, Routing and wavelength assignment in all-optical networks based on the bee colony optimization, AI Communications 20 (4) (2007) 273–285.

[36] S. Nakrani, C. Tovey, On honey bees and dynamic allocation in an internet server Colony, in: Proceedings of Second International Workshop Mathematics and Algorithms of Social Insects, Atlanta, Georgia, USA , 2003.

[37] H.F. Wedde, M. Farooq, Y. Zhang, BeeHive: an efficient fault-tolerant routing algorithm inspired by honey bee behavior, Lecture Notes in Computer Science 3172 (2004) 83–94.

[38] G.M. Bianco, Getting inspired from bees to perform large scale visual precise navigation, in: Proceedings of International Conference on Intelligent Robots and Systems (IEEE/RSJ'2004), Sendai, Japan, 2004, pp. 619–624.
[39] C.S. Chong, M.Y.H. Low, A.I. Sivakumar, K.Y. Gay, A bee colony optimization algorithm to job shop scheduling, in; Proceedings of the 37th Conference on Winter Simulation, Monterey, California, 2006, pp. 1954–1961.
[40] H. Drias, S. Sadeg, S. Yahi, Cooperative bees swarm for solving the maximum weighted satisfiability problem, Lecture Notes in Computer Science 3512 (2005) 318–325.
[41] N. Quijano, K.M. Passino, Honey bee social foraging algorithms for resource allocation theory and application, Submitted for publication, available on <http://wwwprof.uniandes.edu.co/~nquijano/Publications.html>.
[42] B. Baştürk, D. Karaboğa, An artificial bee colony (ABC) algorithm for numeric function optimization, in: IEEE Swarm Intelligence Symposium, Indianapolis, Indiana, USA, May 12–14, 2006.
[43] D. Karaboğa, B. Baştürk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization 39 (2007) 459–471.
[44] X.S. Yang, Engineering optimizations via nature-inspired virtual bee algorithms, Lecture Notes in Computer Science 3562 (2005) 317–323.
[45] D.T. Pham, S. Otri, A. Ghanbarzadeh, E. Koç, Application of the bees algorithm to the training of learning vector quantisation networks for control chart pattern recognition, in: Proceedings of International Conference on Information and Communication Technologies, 24–28 April 2006, Umayyad Palace, Damascus, Syria, 2006, pp. 1624–1629.
[46] D.T. Pham, E. Koç, A. Ghanbarzadeh, S. Otri, Optimisation of the weights of multi-layered perceptrons using the bees algorithm, in: Proceedings of the Fifth International Symposium on Intelligent Manufacturing Systems, Sakarya, Turkey, 2006, pp. 38–46.
[47] D.T. Pham, A.J. Soroka, A. Ghanbarzadeh, E. Koç, S. Otri, M. Packianather, Optimising neural networks for identification of wood defects using the bees algorithm, in: Proceedings of IEEE International Conference on Industrial Informatics, August 16–18, 2006, Singapore, 2006, pp. 1346–1351.
[48] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, Application of the bees algorithm to the training of radial basis function networks for control chart pattern recognition, in: Proceedings of the Fifth CIRP International Seminar on Intelligent Computation in Manufacturing Engineering, Ischia, Italy, 2006, pp. 711–716.
[49] D.T. Pham, Z. Muhamad, M. Mahmuddin, A. Ghanbarzadeh, E. Koç, S. Otri, Using the bees algorithm to optimise a support vector machine for wood defect classification, in: Proceedings of Innovative Production Machines and Systems Virtual Conference, Cardiff, UK, 2007.
[50] D.T. Pham, A. Afify, E. Koç, Manufacturing cell formation using the bees algorithm, in: Proceedings of Innovative Production Machines and Systems Virtual Conference, Cardiff, UK, 2007.
[51] D.T. Pham, E. Koç, J.Y. Lee, J. Phrueksanant, Using the bees algorithm to schedule jobs for a machine, in: Proceedings of the Eighth International Conference on Laser Metrology, CMM and Machine Tool Performance, Euspen, UK, 2007, pp. 430–439.
[52] D.T. Pham, M. Castellani, A. Ghanbarzadeh, Preliminary design using the bees algorithm, in: Proceedings of the Eighth International Conference on Laser Metrology, CMM and Machine Tool Performance, Euspen, UK, 2007, pp. 420–429.
[53] D.T. Pham, S. Otri, A. Afify, M. Mahmuddin, H. Al-Jabbouli, Data clustering using the bees algorithm, in; Proceedings of the 40th CIRP International Manufacturing Systems Seminar, Liverpool, UK, 2007.
[54] D.T. Pham, A.J. Soroka, E. Koç, A. Ghanbarzadeh, S. Otri, Some applications of the bees algorithm in engineering design and manufacture, in: Proceedings of International Conference on Manufacturing Automation, 28–30 May 2007, Singapore, 2007.
[55] D.T. Pham, A. Ghanbarzadeh, Multi-objective optimisation using the bees algorithm, in: Proceedings of Innovative Production Machines and Systems Virtual Conference, Cardiff, UK, 2007.
[56] D.T. Pham, A.H. Darwish, E.E. Eldukhri, S. Otri, Using the bees algorithm to tune a fuzzy logic controller for a robot gymnast, in: Proceedings of International Conference on Manufacturing Automation, 28–30 May 2007, Singapore, 2007.
[57] L.A.N. Lorena, M.G. Narciso, J.E. Beasley, A constructive genetic algorithm for the generalized assignment problem, Evolutionary Optimization, Submitted for publication, Available from: <http://www.lac.inpe.br/~lorena/public.html>.