Appl Intell (2015) 43:162–175
DOI 10.1007/s10489-014-0640-z

A novel approach to task assignment in a cooperative multi-agent design system

Hong Liu · Peng Zhang · Bin Hu · Philip Moore

Published online: 18 January 2015

Abstract The task assignment problem is an important topic in multi-agent systems research. Distributed real-time systems must accommodate a number of communication tasks, and the difficulty in building such systems lies in task assignment (i.e., where to place the tasks). This paper presents a novel approach that is based on artificial bee colony algorithm (ABC) to address dynamic task assignment problems in multi-agent cooperative systems. The initial bee population (solution) is constructed by the initial task assignment algorithm through a greedy heuristic. Each bee is formed by the number of tasks and agents, and the number of employed bees is equal to the number of onlooker bees. After being generated, the solution is improved through a local search process called greedy selection. This process is implemented by onlooker and employed bees. In greedy selection, if the fitness value of the candidate source is greater than that of the current source, the bee forgets the current source and memorizes the new candidate source. Experiments are performed with two test suites (TIG representing real-life tree and Fork–Join problems and randomly generated TIGs). Results are compared with other nature-inspired approaches, such as genetic and particle swarm optimization algorithms, in terms of CPU time and communication cost. The findings show that ABC improves these two criteria significantly with respect to the other approaches.

Keywords Task assignment · Swarm intelligence · Artificial bee colony algorithm · Multi-agent cooperative design · NP hard problem

H. Liu (✉) · P. Zhang · B. Hu · P. Moore
School of Information Science and Engineering, Shandong Normal University, Jinan, China
e-mail: lhsdnu@126.com

B. Hu
e-mail: bh@lzu.edu.cn

H. Liu · P. Zhang · B. Hu
Shandong Provincial Key Laboratory for Novel Distributed Computer Software Technology, Jinan, China

B. Hu · P. Moore
School of Information Science and Engineering, Lanzhou University, Lanzhou, China

1 Introduction

Task assignment is one of the core steps in effectively exploiting the capabilities of distributed or parallel computing systems. The task assignment problem (TAP) is one of the most important research topics in multi-agent systems (MAS). It is defined as the allocation of (T) tasks to (A) agents in a distributed system. General solutions (A*T) exist for the allocation of tasks to agents; the challenges are principally related to identifying optimal allocation of all tasks while accommodating the design constraints that apply to the domain of interest. Addressing these challenges is an NP-hard problem [1–4]. Kartik and Murthy in [5] demonstrated that the computational complexity of the reliability problem in distributed systems is NP-hard in a strong sense. NP-hard problems are difficult to solve, and no polynomial time algorithms have been established to solve them [6].

Addressing TAP essentially involves assigning a set of tasks to a set of agents in a distributed system; the objective is to minimize the communication and task processing cost among cooperating agents. In other words, approaches designed to manage TAP (and assign tasks to agents) attempt to determine the minimal computational costs in constrained multi-agent cooperative systems while optimizing task assignment to allow for the effective and efficient functioning of distributed systems.

Given the critical importance of task assignment, TAP has been the subject of extensive research. Allocation schemes can be classified into the following two categories [6].

- ***Exact methods***: methods that attempt to identify the optimal location for a given objective
- ***Heuristic algorithms***: methods that aim to provide a fast and effective means of obtaining sub-optimal solutions

The first category (exact methods) includes numerous exact methods, such as the family of branch and X algorithms that includes branch and bound algorithm, branch and cut algorithm, branch and price algorithm, linear programming, and dynamic programming [6] to name a few. However, as observed by Jourdan et al. [6], such approaches are limited in their capabilities to resolve moderately sized problem instances. When instances become too large for exact methods, heuristics (particularly meta-heuristic methods) are often employed [6]. Heuristic approaches have been studied more extensively than exact methods and have been demonstrated to provide an effective basis upon which sub-optimal solutions can be achieved. Examples of heuristic approaches include evolutionary algorithms (EA), evolutionary strategies (ES), genetic programming (GP), scatter search (SS), immune systems (IS), and swarm intelligence (SI), which is central to the approach posited in this study. Heuristic algorithms generally incur lower computational overhead (computation time) than exact methods and can be very useful in applications where an optimal solution is unobtainable within a critical time limit.

As discussed above, a number of approaches, including exact methods and heuristic algorithms, have been designed to address task assignment and scheduling in distributed systems [5–8, 10–12, 14–21]. SI employs nature-inspired algorithms in optimization problems [22] and has been shown to produce significant improvements in performance in comparison with alternative approaches in simulating group-based actions. This condition is attributed to the fact that SI is inspired by the activities of social animals or insects in nature. As a basis for task assignment in a multi-agent system, SI has demonstrated the capability to achieve improved performance in terms of optimization and accuracy in classifying tasks.

In this paper, we present a novel approach to solve TAP in a multi-agent system. The approach is based on SI using the artificial bee colony (ABC) algorithmic approach. The contributions provided by this proposed novel approach to task assignment are summarized below.

1. The proposed approach defines TAP in multi-agent systems and provides a dynamic resource assignment algorithm based on ABC.
2. Compared with alternative approaches, the proposed approach has the following benefits and advantages: (1) memory, (2) multi-character, (3) local search, and (4) a solution improvement mechanism. To realize these advantages, we selected the ABC algorithmic approach to model and solve complex TAP.
3. A design example of CA610 lathe is presented for task assignment, and the effectiveness of the ABC algorithm is compared with that of other popular intelligent algorithms through experiments. Computational simulations and comparative analyses are conducted via testing through the use of three test suites. The obtained results support the conclusion that the proposed approach allows for significant improvements in solving complex TAPs when compared with alternative methods, including other nature-inspired approaches such as genetic algorithms and particle swarm optimization techniques.
4. An important aspect of the proposed novel SI-based approach is that it is designed to solve NP-hard problems.

The remainder of this paper is organized as follows. Section 2 presents an overview of related research on TAP and SI, with a focus on ABC approaches. Section 3 introduces a multi-agent cooperative design system. In Section 4, a dynamic task assignment approach based on the bee colony principle and the use of the ABC algorithm is presented. Section 5 presents the experimental results obtained from tests using three test suites. The last chapter provides the conclusions and recommendations for future research.

2 Related work

This section considers task assignment and the approaches to enable the effective allocation of tasks in dynamic and complex environments. SI is considered a potential solution to TAP in dynamic environments, with a focus on the ABC algorithmic approach. The design of the proposed ABC approach is discussed in the following sections.

2.1 TAP

Resource distribution is a type of TAP in which multiple agents are required to capture and deliver resources at a number of pre-defined locations while minimizing the average waiting and delivery times for both resources and clients [6]. Agents in a resource distribution system must (1) cooperate in dynamic and complex environments, (2) implement unexpected requests (this may be viewed in terms of the lack of *a priori* knowledge related to upcoming requests for service), and (3) achieve effective performance levels in a robust system (i.e., accommodate possible failures of individual agents). Therefore, multi-agent systems designed for resource distribution tasks should be scalable, adaptive, and robust [6].

Many approaches to task assignment and scheduling in distributed systems have been proposed in literature [7]; these approaches include heuristic methods, as discussed by Lo in [8]. In the field of multi-agent coordination (which includes resource distribution and TAPs), two methods are commonly used: (a) multi-agent planning and (b) nature-inspired approaches. The disadvantage of planning-based systems is that either global information is required or agents must communicate extensively; meanwhile, nature-inspired methods have been demonstrated to perform successfully and enable the resolution of both research and real-world problems. A comprehensive overview of multi-agent planning is provided in [9].

An alternative approach initially proposed by Shatz et al. in [10] employs state space search algorithms (SSAs). The SSA approach has been subsequently improved by Kartik and Murthy [5] by using the notion of branch and bound (BB) search with underestimates and module independence to reduce the average computational effort required in establishing optimal task allocation. Although they are useful for small-scale problems, SSA approaches are limited in their capability to handle large and complex problems (such as TAP) and are therefore unsuitable for TAP in general. Meanwhile, heuristic algorithms can derive near-optimal or optimal solutions with a reasonable amount of computing time. Therefore, in recent years, research on TAP has tended to focus on developing meta-heuristic search algorithms to address the problem. A number of constructive heuristics have been proposed for meta-task assignment. Vidyarthi and Tripathi [11] presented a solution that employs a simple GA to identify a near-optimal allocation solution. Attiya and Hamam [12] developed a simulated annealing algorithm to address TAP; evaluation of the algorithm's performance (in comparison with a BB technique) produced satisfactory results.

Yin et al. [13] proposed a hybrid algorithm that combines particle swarm optimization (PSO) and a hill climbing heuristic; the researchers claimed that their solution outperforms GA in terms of effectiveness and efficiency. Given the intractable nature of TAP when taken with the constantly growing demand for distributed computing, exploring other avenues for developing good heuristic algorithms for TAP is useful.

With regard to the question *"can agents coordinate their activity using only communication within their local environment"* De Jong et al. [14] found that research has investigated nature as a source of inspiration. Hence, learning-based methods have been proposed. For example, Strens and Windelinckx [15] considered the combination of planning with reinforcement learning for multi-robot task allocation. Examples of documented research that addresses natureinspired systems can be found in [16–21]. However De Jong et al. in [14] noted that *"these systems often lack a certain degree of pragmatism required for real-world applications."*

2.2 SI and ABC approaches

SI is a research area that employs nature-inspired algorithms in optimization problems [22]. SI research has produced a number of algorithms that are based on SI; these algorithms involve modeling the behavior of a range of insects and animals, including termites, birds, ants, bees, wasps, and fish [22]. SI is based on models of collective intelligence in swarms of insects or animals and has been defined as *"the collective behavior of decentralized and self-organized swarms"* [23]. The capability of ant colony optimization (based on the swarming behavior of ants) and PSO (based on bird flocks and fish schools) to solve optimization problems in a number of areas was demonstrated in the 1990s [22]. A fertile area of research within the general area of SI is the investigation of bee swarming; ABC is a relatively recent SI algorithm.

The foraging behavior and learning, memorizing, and information sharing characteristics of bees have been one of the most interesting research areas in SI [24]. ABC algorithms can be viewed in terms of the behavioral characteristics of honey bees, namely, (1) foraging behaviors, (2) marriage behaviors, and (3) the queen bee concept. A detailed description of the behavior of bees in nature with a review and categorization of studies on artificial bee systems can be found in [24]; for a detailed exposition on the ABC approach to optimization, one may refer to [22, 24–28]. The ABC algorithm generally attempts to model the foraging behavior of honeybee colonies [23].

An enhanced ABC optimization algorithm called interactive artificial bee colony (IABC) was developed for numerical optimization problems by Tsai et al. [28]. In the IABC approach, an onlooker bee is designed to move directly to the selected coordinate indicated by an employed bee and evaluate the fitness values near it (in the original ABC algorithm) in an attempt to reduce computational complexity. Essentially, IABC is an extension of the ABC algorithm. It has been tested through the use of five benchmark functions in simulated experiments to compare the accuracy/quality of IABC, ABC, and PSO; its proponents reported that IABC exhibits superior performance in terms of accuracy when compared with other methods [28].

A novel use of the ABC algorithm was developed by [26]. In this approach, the ABC algorithm employs fuzzy clustering. The algorithm has also been tested against a range of data sets, including cancer, diabetes and cardiovascular datasets obtained from the UCI Library's database [29], which contains a collection of classification benchmark problems. Successful performance outcomes were obtained. The results revealed the good performance of the ABC optimization algorithm when used in conjunction with fuzzy clustering.

SI involves a social component in that social insect colonies can be considered dynamic systems that gather information from the environment and modify behaviors in accordance with the environmental conditions. While gathering information and setting behavioral patterns (driven by environmental conditions), individual insects do not perform all the tasks as in social insect colonies because of the existence of specialisms (e.g., the queen as well as drone and worker bees). Virtually all social insect colonies behave according to their own division of labor related to their morphology. Bees exist in highly organized colonies with a strict hierarchical structure. A colony of bees is made up of three categories of adults: the queen, the drones, and the workers. Bees are one of the most studied social insects because the highly developed organizational structure of a bee colony makes bees a very attractive source of research inspiration. Bee colonies are in fact communities divided into several social layers. Communication is interesting as bees communicate by (1) dancing and (2) producing pheromones. Bees cooperate to realize various tasks, such as construction and maintenance of the hive and harvesting of pollen. An interesting survey of algorithms inspired by the behavior of bees can be found in [22, 30].

The ABC algorithm, which was proposed by Karaboga [31] and further developed by Karaboga and Basturk [32], is a relatively new population-based meta-heuristic approach. ABC is inspired by the intelligent foraging behavior of the honeybee swarm. Karaboga and Basturk [33] and Karaboga and Akay [34] compared the performance of the ABC algorithm with that of differential evolution, which has been claimed by Sun et al. in [35] to be "*very successful in solving the global continuous optimization problem*," PSO, and evolutionary algorithms according to a set of well-known test functions. The performance of ABC was also analyzed under a change in control parameter values. The simulation results showed that the ABC algorithm performs better than the abovementioned algorithms and can be efficiently employed to solve multimodal engineering problems with high dimensionality.

Approaches have been proposed and applied as solutions to solve combinatorial-type problems [36, 37]. Specifically, in consideration of TAP, Lale et al. [38] developed an algorithm based on SI and bee behaviors and with an ejection chain neighborhood mechanism to solve generalized assignment problems [16]. Yeh and Hsieh [39] in their paper entitled "Solving reliability redundancy allocation problems using an artificial bee colony algorithm," proposed a penalty-guided artificial bee colony algorithm to solve the reliability redundancy allocation problem (RAP). Investigations have been conducted on RAP over the past four decades [39]; Yeh and Hsieh [39] stated that "*to the best of our knowledge, the ABC algorithm can search over promising feasible and infeasible regions to find the feasible optimal/near-optimal solution effectively and efficiently.*" Investigations that involved the use of numerical examples support the conclusion that the penalty-guided ABC approach performs well in the reliability–redundancy allocation design problems considered in the study; the computational results compare favorably with those of previously developed algorithms in literature [39].

This brief overview of the research related to task assignment and resource distribution considered the approaches to task assignment. The ABC optimization algorithm has been widely studied and has been successfully applied to resolve real-world problems. ABC approaches have significant advantages in terms of memory, multi-character, local search, and the solution improvement mechanism; hence, ABC methods are capable of identifying optimal and near-optimal solutions. However, notwithstanding the efficacy of the ABC approach in its basic form as discussed above and in literature, the approach is limited when used to address TAP. A meta-heuristic optimization approach that employs a penalty-guided algorithm based on the ABC approach to solve TAP in a multi-agent cooperative design system is proposed in this paper.

3 Multi-agent cooperative design system

A multi-agent collaborative design system is essentially concerned with how a group of intelligent agents can cooperate to jointly solve problems. Design is a complex process of knowledge discovery, in which information and knowledge from a broad and diverse range of sources is processed simultaneously by a team of designers involved in the life cycle of a project or product. Complex designs generally combine automated software components and human decision makers; hence, providing support for both human and computational participants in the design process is imperative.

The general architecture of a multi-agent collaborative design system is organized as a population of asynchronous semi-autonomous agents to enable the integration of design and engineering tools with human experts and specialists in an open environment. Each tool (or interface for a human specialist) can be encapsulated as an agent. These tools and human specialists are connected by a local network; they communicate via this network. Each can also communicate directly with other agents located in other local networks through the Internet. Agents exchange design data and knowledge via a local network (Internet) or the Internet through the management agent [40]. All agents in the system form an agent group, in which three classes of agents exist: (1) management, (2) tool, and (3) design agents. The agents are situated on different layers, and the hierarchical relation limits the authority of the agents within the group.

The management agent is located on a server and manages the interactions of the entire design group. The action of the management agent usually shows the inquiry made and the resultant decision for a particular problem, the control function, and the supervision functions for the lower-layer agents. The knowledge in the knowledge base (KB) of a management agent includes all relevant information (e.g., a design agent's name, address, skills or competencies, historical records for the task being undertaken, and reward in the group). When an agent is added to or deleted from the group, the corresponding knowledge of the management agent is modified.

Tool agents include design and management tools; they provide assistance to the management agent in the completion of the system management tasks. Typical tasks include communication management, task assignment, database management, knowledge management, collaboration management, and system maintenance. The role and functions of the various agents that combine to make up the design system can be summarized as follows [41].

- The *task assignment* agent helps design engineers decompose a large design task into several sub-tasks and assigns them to suitable design agents.
- The *collaborative* agent deals with conflict coordination during the collaborative design process.
- *Design tool* agents include software packages, such as AutoCAD, Pro-Engineer, Inventor, MicroStation, and SolidWorks. They also include a video conferencing system for a synchronous collaborative design that provides run-time support.
- *Communication* agents provide support for the interaction among agents and designers through e-mail, text, file, image, graph, audio, and video. The exchange of data and files is based on the file transfer protocol (FTP) and TCP/IP protocol.
- The *process monitor* agent monitors the entire design process via its event monitor and dynamically maintains information related to the current prevailing state of each design agent and the status of current design sub-tasks. The event monitor is triggered when a design event occurs, and the correlative message is passed to relevant and suitable agents.
- The *assemble agent* checks the assembly constraints for finished design components. When a constraint violation is found, this agent asks collaborative and communication agents to solve problem by coordinating with design agents.
- The *knowledge maintenance* and *database agents* maintain the knowledge base and database, respectively.

The collaborative design process is shown in Fig. 1. When a large design task arrives, the task composition agent

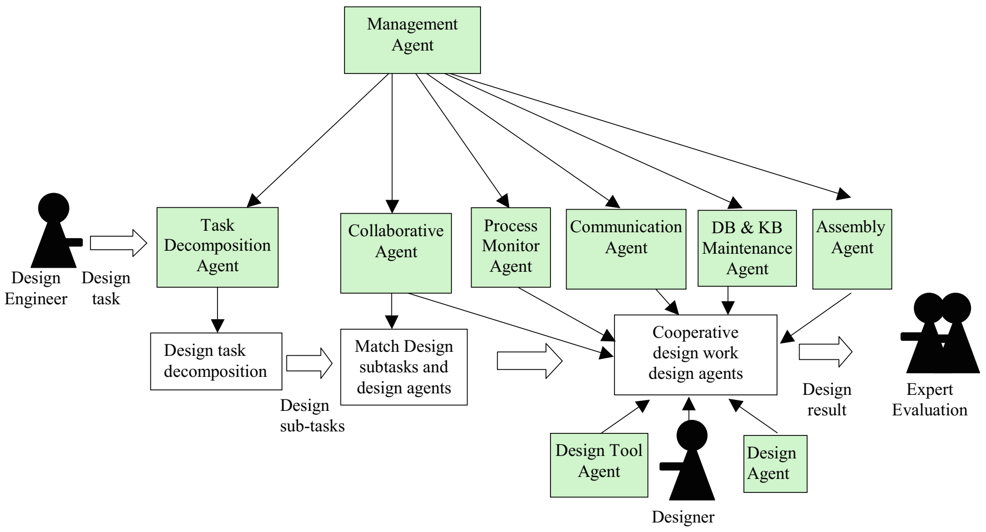

Fig. 1 Collaborative design process

helps the design engineer decompose the task into subtasks and sends the subtasks to the collaborative agent. The collaborative agent then matches the subtasks and design agents according to the latter's capability. After processing by a dynamic task assistant, the design agents and designers perform their respective assigned design tasks. During the design process, the communication agent takes charge of the interaction among agents; the knowledge maintenance and database agents maintain and update the knowledge base and database. The process monitor agent monitors the entire design process. When the assembly agent finds a constraint violation, it informs the collaborative and communication agents with the goal of solving the issue through coordination among design agents. When the design phase is over, experts evaluate the design result to determine if the design process is completed or should be restarted [42].

4 Dynamic task assignment process

The task assignment agent employs a method that divides a design task into a sequence of subtasks and assigns them to suitable design agents.

4.1 Formal description of task assignment

The formal definition of the task assignment approach posited in this study is as follows.

Definition 4.1 DA_s denotes a design agent, where D is the type of agent and s is a character string that represents which group the agent belongs to and its serial number in the group. For example, DA_{11} is a design agent with number 1 in group 1.

Definition 4.2 T_s stands for a design task, and c is a character string that represents the decomposed layer of the design task and the dependency relation. For example, an initial design task can be represented as T_1. Its subtasks are $T_{11}, T_{12}, \ldots, T_{1n}$, and the sub-processes of T_{1i} are $T_{1i1}, T_{1i2}, \ldots, T_{1im}$, i.e., the length of the string denotes the decomposed depth, and the value expresses the dependency relation (i).

The dependency relation of design tasks forms a design task tree.

Definition 4.3 T_i^j denotes that task i is being implemented by design agent j.

The group members performing task Ti can be determined by vector $\left(T_i^{j1}, T_i^{j2}, \ldots, T_i^{jk}\right)$ and the current tasks of design agent j by vector $\left(T_{i1}^j, T_{i2}^j, \ldots, T_{il}^j\right)$.

Definition 4.4 The prior relation of a design task is indicated by pair PRIOR (T_{s1}, T_{s2}), which means that T_{s2} takes the fulfillment of T_{s1} as the starting pre-condition; T_{s1} and T_{s2} are the sequences of tasks.

Definition 4.5 The concurrent relation is indicated by CONCUR (T_i, T_j), which expresses that design tasks T_i and T_j can be implemented simultaneously.

Definition 4.6 The exclusive relation is indicated by EXCLUDE (Ti, Tj), which denotes that two tasks $\left(T_i and T_j\right)$ cannot be performed simultaneously.

Definition 4.7 Event is expressed by E (i).

Table 1 presents the notations used in TAP formulation.

The general formulation of TAP can be described as follows:

$$\text{Min} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \tag{1}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} r_{ij}x_{ij} \le b_j \quad \forall j \quad 1 \ge j \le m \tag{2}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i \quad 1 \le i \le n \tag{3}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \quad 1 \le i \le n \quad \forall j \quad 1 \ge j \le m. \tag{4}$$

According to (1) and Table 1, this problem is a 0–1 quadratic integer programming problem, and its objective function is the total sum of execution and communication costs. This

Table 1 Notations used in TAP formulation

n	number of tasks in set T
m	number of agents in set A
b_{ij}	amount of available resource of agent j ($b_j \ge 0$)
r_{ij}	resource needed if task i is assigned to agent j (R is a matrix of size $n \times m$, $r_{ij} \ge 0$)
c_{ij}	cost needed if task i is assigned to agent j (C is a matrix of size $n \times m$, $c_{ij} \ge 0$)
x_{ij}	decision variable if task i is assigned to agent j, x_{ij} =1; 0, otherwise (C is a matrix of size $n \times m$)

problem is limited by two sets of constraints. The first constraint set in (2) means that the task assigned to agent j cannot exceed the resource capacity of agent j, and the second constraint set in (3) ensures that each task is assigned to only one agent.

4.2 Dynamic task assignment approach based on ABC algorithm

The proposed approach is presented in this section. After a description of bee colony initialization, we discuss the design solution for the proposed resource assignment approach.

4.2.1 Bee colony initialization

The initial bee colony is constructed using the initial task assignment algorithm, where the greedy heuristic constructs a solution as follows.

1. At each step, the next task to be assigned to an agent is selected.

```
Initial Task Assignment Algorithm
Step 1: Initialization
  1. A = {agent i}, i = 1,2,...,m;
  2. T = {task j}, j = 1,2,…,n;
  3. $S_j = \phi$, j = 1,2,...m  ($S_j$ is the set of task assigned to agent j);
  4. Construct a list of agents $L_i$ for each task i. The agents in $L_i$ possess the capability to accomplish task i;
Step 2: Select a suitable agent for each task i
  For (i=1; i<=n; i++)
    1. Select an agent j from $L_i$;
    2. Compute the probability according to the resource of agent j and the resource required by task i;
       $p_{ij} = \frac{r_j / b_{ij}}{\sum_{l \in L} r_l / b_{il}} \quad j \in L_i \; p_{ij} \quad \frac{r_j / b_{ij}}{r_l \quad b_{il}} \quad j \quad L_i$
    3. The agent with a minimal cost has a high probability of being selected;
    4. Assign task i to agent j*;
    5. $S_{j^*} = S_{j^*} \cup (j^*)$;
    6. i=i+1;
    7. If $\sum_{i \in s_j} b_{ij^*} > r_{j^*}$, remove j* from any list;
Step 3: Record assigned task i to agent j
    For (i=1; i<=n; i++),
      if $i \in s_j$, then $\sigma(i)$=j;
    Return.
```

2. The agent to whom the selected task will be assigned to is determined.
3. Steps 1 and 2 are repeated until all tasks have been assigned to agents.

In the algorithm, probabilistic bias can be implemented to a probability function. This function is updated at each iteration with reinforcement by using the features in good solutions.

4.2.2 Proposed solution design

The ABC algorithm is iterative. A colony of artificial bees in the ABC algorithm is analogous to its natural counterpart (a real colony of bees) in that three categories of bees exist: employed, onlooker, and scout bees. The first half of the bee colony comprises employed bees, and other half contains onlooker bees. The ABC algorithm proposed in this study functions as follows. Initially, all employed bees are associated with randomly generated food sources (solutions). Iterative processing then occurs; in each iteration, every employed bee determines a (new) food source in the neighborhood of its currently associated food source and evaluates its nectar value (amount), which represents fitness. If the nectar value of the new food source is greater than the nectar value of the currently associated food source, then the employed bee moves to this new food source and leaves the old (current) food source; otherwise, it retains its old food source. When all employed bees have finished this process, they share the nectar information relating to the food sources with the onlooker bees. At this stage, each of the onlooker bees selects a food source according to a probability proportional to the nectar value of that food source.

The ABC algorithm assumes that only one employed bee exists for every food source (a design parameter for the entire system). Hence, in the ABC algorithm, the number of food sources is similar to the number of employed bees. The employed bee for an abandoned food source becomes a scout bee. When it finds a new food source, it returns to being an employed bee. ABC generates a randomly distributed initial population of SN solutions (food source, positions, and the reliability) during initialization, and the SN parameter denotes the population size of the employed or onlooker bees. Below is a detailed description of the process.

Each solution $X_h (h = 1, 2, \cdots, SN)$ is a d-dimensional vector, where d is the number of optimization parameters. The population of the positions (solutions) is subject to repeated cycles [$(C = 1, 2 \ldots,$ maximum cycle number (MCN)] of the search processes for the employed, onlooker, and scout bees. After building potential solutions, the subsequent task is to evaluate the food source (solutions) using the fitness function, which is a common feature of nature-inspired approaches for optimization. The design of the fitness function is a crucial feature in ABC (as in all nature-inspired approaches) and performs a vital task in determining the optimization function in the ABC algorithm.

The fitness values are utilized to determine the probability of the individual bees being selected. In computing the value of the fitness function, a penalty term is added to the fitness function to convert the constrained problem into an unconstrained one. While constructing initial solutions (in the ABC algorithm), the proposed approach may produce infeasible solutions. Therefore, an additional term, which is determined by penalizing the infeasible solutions with $\alpha_j (\alpha_j > 0)$, is introduced to the fitness function. The fitness function is formally defined as follows:

$$fit(\sigma^p) = \sum_{j=1}^{m} \sum_{i=1}^{n} c_{ij} x_{ij} + (\sum_{j=1}^{m} \alpha_j \max \left(0, \sum_{i=1}^{n} r_{ij} x_{ij} - b_j\right), \quad (5)$$

where

σ^p	is the solution for employed bee **p**
fit(σ^p)	is the fitness function value for employed bee **p**
σ^{best}	is the best solution, and
α_j	is the cost of using one overloaded capacity of agent j.

The first term in the fitness function denotes the total cost of assigning tasks to agents. The second term is defined as an additional penalty function for minimization. α_j represents the cost of using one overloaded capacity of agent j. The initial values of α_j's are set by the user. If a solution is not feasible, the second term will be positive. Therefore, the search will be directed to a feasible solution. If the capacity is not exceeded, this term will be zero to ensure that the solution is not penalized. The parameter can be increased during the run to penalize infeasible solutions and drive the search to feasible ones; this condition denotes the adaptive control of penalty costs.

The solutions for each task are selected using the probabilities established by (6). The probability p_h of selecting a solution X_h is then determined.

$$p_h = \frac{fit(\sigma^h)}{\sum_{i=1}^{SN} fit(\sigma^i)} \quad (6)$$

With this scheme, good food sources will obtain more onlookers than the bad ones. After all onlookers have selected their food sources, each of them determines a food source in the neighborhood of his selected food source and computes its fitness. The best food source among all the neighboring food sources determined by the onlookers (associated with a particular food source i itself) will be the new location of food source i.

Local search for solution improvement After being generated, a solution is improved through a local search process called greedy selection. This process is implemented by onlooker and employed bees. In greedy selection, if the nectar value (fitness) of the candidate source is greater than that of the current source, the bee forgets the current source and memorizes the new candidate source. This condition is achieved by adding to the current value of the selected parameter the product of a uniform variable within the range [-1, 1] and the difference in values of the parameter of this food source and of several other randomly selected food sources.

We suppose that each solution consists of **d** parameters and let $X_h = (X_{h1}, X_{h2}, K, X_{hd})$ be a potential solution. To determine a new solution $NewX_h$ in the neighborhood of X_h, a solution parameter **j** and another solution $X_k = (X_{k1}, X_{k2}, K, X_{kd})$ are selected randomly. Except for the value of the selected parameter **j**, all other parameter values of $NewX_h$ are similar to X_h, i.e., $NewX_h = (X_{h1}, X_{h2}, K, X_{h,j-1}, Z_{h,j}, X_{h,j+1}, L, X_{hd})$. The value of the selected solution $NewX_h$ is determined with (7) as follows:

$$NewX_{hj} = X_{hj} + u(X_{hj} - X_{kj}), \quad (7)$$

where **u** is a uniform variable within the range [-1,1]. If the resulting value falls outside the acceptable range for parameter **j**, it is set to a corresponding extreme value in that range.

Solution intensity update The entire process is iterative in that it is repeated until the termination condition is satisfied. In the ABC algorithm, providing that a position cannot be improved further through a predetermined number of cycles, then that food source is assumed to be abandoned. Assuming that the abandoned source is X_h, the scout bee will look for a new food source to replace X_h. This operation can be defined as (8).

$$X_h^j = X_{\min}^j + rand[0.1]\left(X_{\max}^j - X_{\min}^j\right) \quad j = 1, 2, \ldots, d \quad (8)$$

Dynamic resource assignment algorithm This section introduces the dynamic resource assignment algorithm that is based on the ABC approach. The algorithm involves three steps as shown below.

```
Dynamic Task Assignment Algorithm Based on ABC

Step 1: Initialization
1. A = {agent i}, i = 1,2,...,m;
2. T = {task j}, j = 1,2,…,n;
3. Construct the initial bee population (solution) $X_{hj}$ as each
   bee is formed by the number of tasks and agents; the
   number of employed bees is equal to the number of
   onlooker bees;
4. Initiate MCN;
5. Evaluate the fitness value and make the constraints
   satisfactory for each employed bee;
6. Cycle = 1.
Step 2: Task Assignment
While (cycle<MSN),
1. Generate the new population (solution) $NewX_{hj}$ in the
   neighborhood of $X_{hj}$ for the employed bees using
   Equation (7) and evaluate them;
2. Apply a greedy selection process between $X_h$ and
   $NewX_h$;
3. Calculate the probability values $P_h$ of selecting the
   solutions $X_h$ by using Equation (6);
4. Produce new population $NewX_{hj}$ for the onlookers from
   population $X_{hj}$ depending on $P_h$ and evaluate it;
5. Apply a greedy selection process between $X_h$ and
   $NewX_h$ for the onlookers;
6. Determine the abandoned solution for the scout if it exists
   and replace it with a new randomly produced solution $X_h$
   for the scout bees using Equation (8);
7. Memorize the best solution achieved so far;
8. Cycle = cycle+1.
Step 3: Complete Task Assignment Processing
1. Each agent is assigned a suitable design task. The agents
   perform their respective design tasks.
2. After a design task is completed, the agent submits the
   completed design to the assemble agent. The assemble
   agent checks the assembly constraints for the finished
   design components. When a constraint violation is found,
   the assemble agent will ask the collaborative and
   communication agents to solve the problem through
   coordination among design agents.
```

5 Experimental results

In this section, a design example and two test suites are provided to show the performance of the proposed ABC algorithm. Figure 2 presents an image of a CA6140 lathe, and Fig. 3 presents its design sketch. This lathe design is regarded as an example in this section.

Fig. 2 CA6140 lathe

The performance criteria considered to evaluate the quality of the solutions are (1) the assignment (communication) cost and (2) amount of CPU time used for the benchmarks. G (V, E) is utilized to create the task interaction graph (TIG), where V is a set of n nodes indicating the n tasks to be executed and E is a set of edges indicating the communication requirements among these tasks. Each edge $(i, j) \in E$ is associated with a communication cost c_{ij}. Figure 4 shows an example of a TIG (related to Fig. 3) showing the communication requirements among six tasks.

Two key factors affect the complexity of the problem. One factor is the task interaction density (d) of G (V, E) that quantifies the ratio of the inter-task communication demands for a TIG. The other factor is the number of tasks (n) and agents (m). We define d as (9).

$$d = \frac{|E|}{r(r-1)/2}, \tag{9}$$

where $|E|$ calculates the number of existing communication demands in the TIG and $n(n-1)/2$ indicates the maximal number of communication demands among n tasks. To be

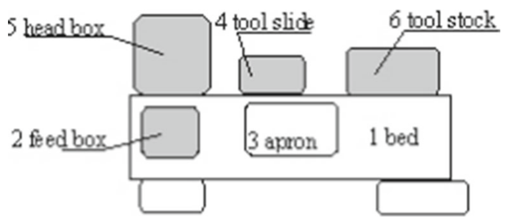

Fig. 3 Design sketch of CA6140 lathe

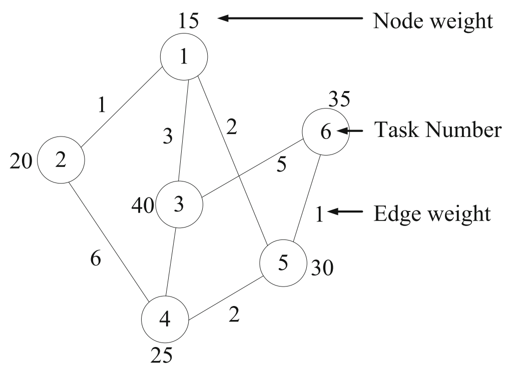

Fig. 4 Example of a TIG

specific, a brief description and the parameter values that control each algorithm are provided as follows.

1. **PSO**
   (a) The size of the population is equal to twice the number of tasks.
   (b) Inertia weight w is set to 0.9.
   (c) $c_1 = c_2 =1.0$.
2. **GA**
   (a) Crossover probability = 0.9.
   (b) Mutation probability =1/total number of tasks.
   (c) Population size = twice the number of tasks.
   (d) Elitism is used.
   (e) Roulette wheel selection.
3. **ABC**
   (a) The number of employed bees is equal to the number of onlookers and tasks.
   (b) MCN = 1000.
   (c) Limit = 20.
   (d) $\alpha_j = 1$.

5.1 Test suite 1

With this test suite, TIGs that represent real-life tree and Fork–Join problems are generated. The structures of the two TIGs are fixed, and each of them requires a different number of vertices. The size of the TIGs is varied. The cost of each node is randomly selected from a normal distribution, with the mean equal to the specified average computational cost. The cost of each edge is also randomly selected from a normal distribution, with the mean equal to the product of the average computational cost and density d equivalent to 0.1, 0.5, 1.0, 2.0, and 10.0.

Tables 2 and 3 show the results for the two graph structures in a homogeneous system with 10 agents. The experimental results are presented in Figs. 5, 6, 7, and 8 in bar line format. Each result is an average of 20 test cases. Algorithm performance varies depending on the structure of TIGs. The ABC algorithm outperforms GA and PSO in terms of task assignment problems; the ABC algorithm achieves the best solutions with the added benefit of requiring the least amount of computation time. In fact, the harder the problem is, the larger the cost difference among the three algorithms. This condition supports the conclusion obtained from the testing, which indicates that the proposed ABC algorithm is more scalable against problem complexity. These results illustrate that the proposed algorithm is a competent solution for task assignment problems at both small and large scales.

5.2 Test suite 2

To analyze the convergence behavior of the proposed algorithm, randomly generated TIGs are considered by using test suite 2. The value of (m, n) is set to (10, 6) and (20, 12), respectively, to verify the algorithm's performance at

Table 2 Comparison of results for the tree-type graph structure

	PSO		GA		ABC	
Tasks	Time (s)	Cost	Time (s)	Cost	Time (s)	Cost
10	0.00686	272.6592	0.0087	255.0182	0.00594	268.5683
20	0.07005	518.9565	0.08012	551.5127	0.06662	520.7253
30	0.24359	643.2102	0.27793	642.7952	0.21220	630.7668
40	0.63232	809.5605	0.68681	852.3117	0.5943	800.6471
50	1.16437	1029.841	1.33058	1073.758	1.12546	1010.632
60	1.94239	995.8665	2.35439	1046.949	1.87578	991.1036
70	3.06135	1372.893	3.73580	1543.903	2.89821	1320.758
80	4.71566	1267.116	5.86446	1361.481	4.27683	1189.523
90	6.87079	1471.317	8.61721	1474.531	6.05837	1479.064
100	9.72206	160.504	12.3438	1564.47	8.12220	1488.024
Total	28.4294	9941.924	35.2999	10366.73	25.16931	9699.8121

Table 3 Comparison of results for the Fork–Join type of graph structure

Tasks	PSO		GA		ABC	
	Time (s)	Cost	Time (s)	Cost	Time (s)	Cost
10	0.009158	310.571	0.009616	298.7352	0.00711	301.7529
20	0.076923	595.1565	0.08837	557.5495	0.70638	600.0979
30	0.255037	843.3817	0.303572	819.1293	0.20861	802.117
40	0.616758	1168.828	0.7587	1303.026	0.53887	1152.5683
50	16.09478	550.8643	16.26603	573.2727	16.0896	562.746
60	2.089744	1438.425	2.679029	1692.423	1.58284	1504.4828
70	3.298078	2012.21	4.243131	2099.977	2.89638	1998.0596
80	5.017399	2393.497	6.624999	2345.468	5.00008	2400.196
90	7.206502	2263.488	9.568681	2520.966	6.19675	2106.594
100	9.820055	2437.536	13.2184	2950.474	7.00967	2398.075
Total	44.48443	14013.958	53.76053	15161.021	40.2363	13826.69

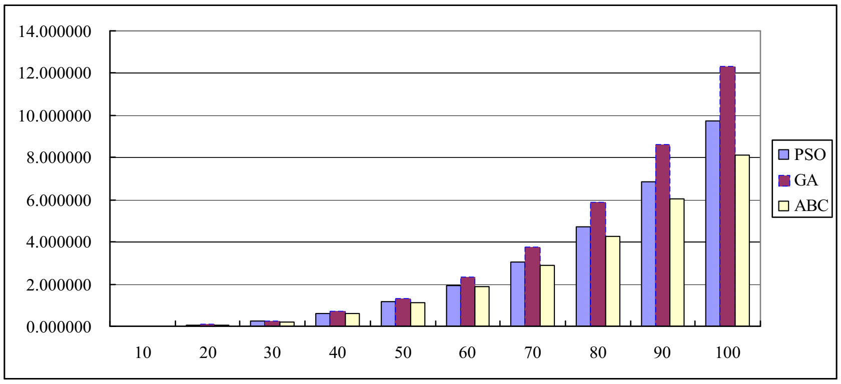

Fig. 5 Comparison of time consumed for the tree-type graph structure

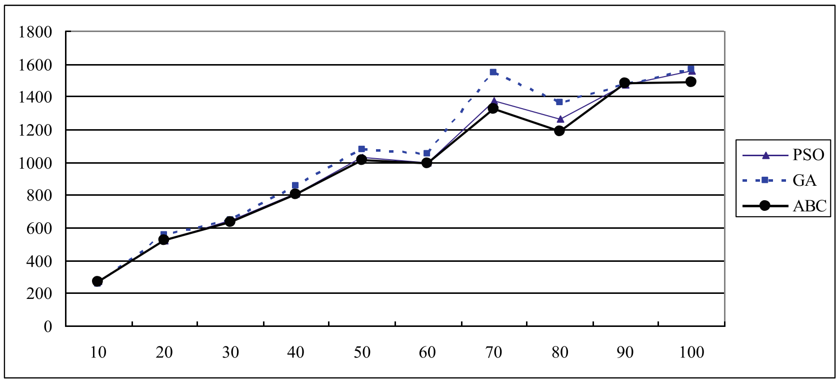

Fig. 6 Comparison of costs for the tree-type graph structure

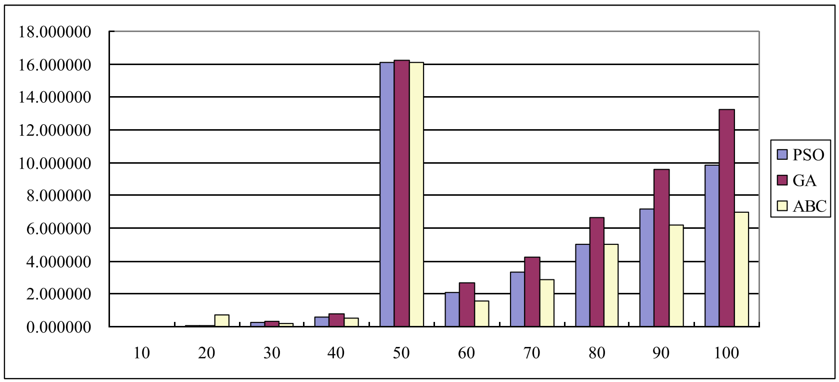

Fig. 7 Comparison of time consumed for the Fork–Join type of graph structure

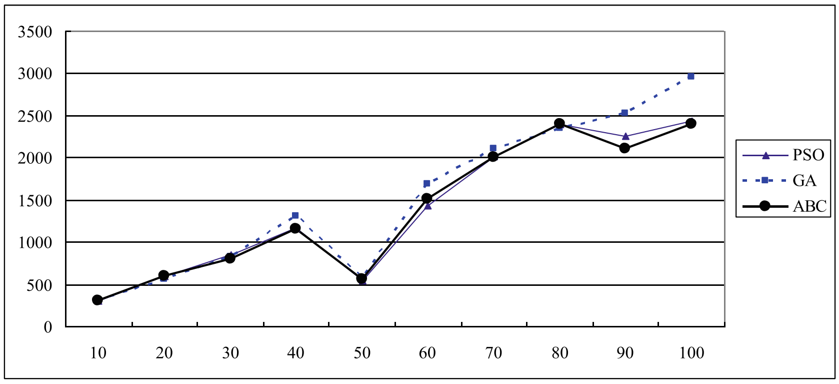

Fig. 8 Comparison of costs for the Fork–Join type of graph structure

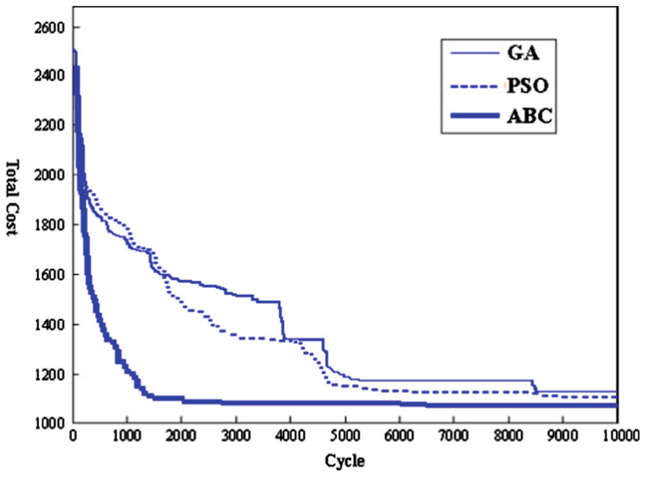

Fig. 9 Comparison for $(m, n, d) = (50, 8, 0.3)$

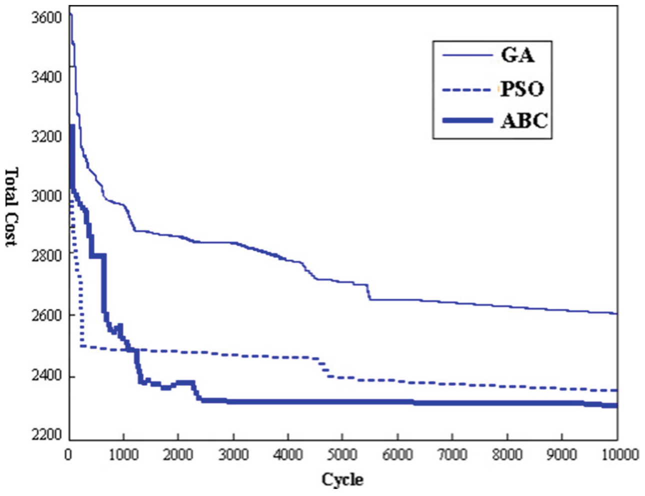

Fig. 10 Comparison for $(m, n, d) = (80, 10, 0.5)$

different problem scales. For each pair of (m, n), three different TIGs are generated randomly. The values of the other parameters are generated randomly; the execution cost is between 1 and 200, the communication cost is between 1 and 50, and the memory and processing capacity of each agent varies from 50 to 250. Figure 9 shows the typical convergence behaviors of ABC, PSO, and GA approaches for instance $(m, n, d) = (50, 8, 0.3)$, and Fig. 10 shows the same for instance $(m, n, d) = (80, 10, 0.5)$.

The results show that the proposed ABC algorithm has a clear advantage in terms of convergence. It runs faster than GA and PSO, and its solution quality is better than that of GA and PSO on the average. These results also indicate that the proposed algorithm has an advantage in addressing the demands of dynamic task assignment because of its stability, multi-character, and iteration speed. It can combine global and local search and is able to discover new optimal solutions over time. It is a potential means to solve TAP.

6 Conclusion

In this study, we considered TAPs in relation to distributed multi-agent systems. A range of potential approaches to address the demands and complexity of TAP in multi-agent systems were analyzed. A novel approach that is based on SI and employs a honeybee optimization algorithm was developed. The proposed approach employs the ABC algorithm to effectively allocate tasks to agents in a system. The algorithm is designed to provide a platform upon which system design can be achieved. The use of the ABC algorithm to realize this objective is the focus of the paper.

The proposed approach to task assignment is a modified version of the basic ABC algorithm initially proposed by Karaboga [31] and is expected to provide an effective platform upon which autonomous task assignment can be achieved to produce an effective cooperative design. The results support the conclusion that the proposed approach allows for more significant improvements in solving complex task assignment problems compared with alternative methods, including other nature-inspired approaches (e.g., GAs and PSO techniques).

Unlike traditional heuristics, approaches that employ the ABC algorithm incorporate have a number of advantages, including memory, multi-character, local search, and a solution improvement mechanism. These attributes provide an effective basis to enable the discovery of new optimal solutions over time. As demonstrated in the study of Karaboga and Basturk [32, 33], the optimal solutions produced by ABC approaches exhibit improved performance when compared with the results obtained by other heuristic methods. In the current study, the proposed method achieved the global or near-global solution in each sample problem tested.

Previous research has identified SI, particularly bee inspired algorithms, as approaches that have a very promising potential for modeling and solving complex optimization problems. Through the proposed approach, we have resolved a number of challenges. However, remaining challenges must be addressed if we are to fully utilize the potential of bee-inspired algorithms as a solution to optimization problems in general and TAP in particular. Addressing these challenges represents the direction of our future work, given that the potential of the proposed approach offers exciting opportunities for search and optimization.

Acknowledgments This project is founded by the program of Taishan scholars and supported by National Natural Science Foundation of China (No. 61272094, No. 61472232).

References

1. Tindell KW, Burns A, Wellings AJ (1992) Allocating hard real-time tasks: an NP-hard problem made easy. Real-Time Syst 4(2):145–165
2. Kanefsky B, Taylor W (1991) Where the really hard problems are. In: Proceedings of the 12th international joint conference on AI (IJCAI'91), pp 163–169
3. Burns A (1991) Scheduling hard real-time systems: a review. Softw Eng J 6(3):116–128
4. Ullman JD (1975) NP-complete scheduling problems. J Comput Syst Sci 10(3):384–393
5. Kartik S, Murthy CSR (1997) Task allocation algorithms for maximizing reliability of distributed computing systems. IEEE Trans Comput 46:719–724
6. Jourdan L, Basseur M, Talbi EG (2009) Hybridizing exact methods and metaheuristics: a taxonomy. Eur J Oper Res 199(3):620–629
7. Pentico DW (2007) Assignment problems: a golden anniversary survey. Eur J Oper Res 176:774–793
8. Lo VM (1988) Heuristic algorithms for task assignment in distributed systems. IEEE Trans Comput 37(11):1384–1397
9. Weerdt DM, Mors TA, Witteveen C (2005) Multi-agent planning: an introduction to planning and coordination. In: Handouts of the european agent summer school, pp 1–32
10. Shatz SM, Wang JP, Goto M (1992) Task allocation for maximizing reliability of distributed computer systems. IEEE Trans Comput 41:1156–1168
11. Vidyarthi DP, Tripathi AK (2001) Maximizing reliability of distributed c computing systems with task allocation using simple genetic algorithm. J Syst Archit 47:549–554
12. Attiya G, Hamam Y (2006) Task allocation for maximizing reliability of distributed systems: a simulated annealing approach. J Parallel Distrib Comput 66:1259–1266
13. Yin PY et al (2007) Task system using hybrid particle swarm optimization. J Syst Softw 80:724–735
14. Jong Sd, Tuyls K, Sprinkhuizen-Kuyper IG (2006) Robust and scalable coordination of potential-field driven agents. Proceeding of: computational intelligence for modelling, control and automation. In: IEEE proceedings of international conference on computational intelligence for modelling control and automation and international conference on intelligent agents web technologies and international commerce, pp 230/1-230/8
15. Strens MJA, Windelinckx N (2005) Combining planning with reinforcement learning for mult-robot task allocation. In: Proceedings of adaptive agents and multi-agent systems, pp 260–274
16. Brueckner S, Parunak H (2000) Multiple pheromones for improved guidance. In: Proceedings of symposium on advanced enterprise control, Minneapolis
17. Weyns D, Boucke N, Holvoet T, Schols W (2005) Gradient field-based task assignment in an AGV transportation system. In: Proceedings of 5th international joint conference on autonomous agents and multiagent systems, pp 447–458
18. Panait L, Luke S (2004) A pheromone-based utility model for collaborative foraging. In: Proceedings of the 3rd international joint conference on autonomous agents and multi-agent systems, pp 36–43
19. Parunak H, Brueckner S, Sauter J (2002) Synthetic pheromone mechanisms for coordination of unmanned vehicles. In: Proceedings of the 1st international joint conference on autonomous agents and multiagent systems. ACM Press, Bologna, pp 448–450
20. Dorigo M, Stuetzle T (2004) Ant colony optimization. MIT Press, Cambridge
21. Holland OE, Melhuish C (1999) Stigmergy, self-organisation, and sorting in collective robotics. Artif Life 5(2):173–202
22. Karaboga D, Akay B (2009) A survey: algorithms simulating bee swarm intelligence. Artif Intell Rev 31(1-4):61–85
23. Akay B, Karaboga D (2012) A modified artificial bee colony algorithm for real-parameter optimization. Inf Sci Swarm Intell Appl 192(1):120–142
24. Baykasoglu A, Ozbakir L, Tapkan P (2007) Artificial bee colony algorithm and its application to generalized assignment problem. In: Felix T, Chan S, Tiwari MK (eds) Swarm intelligence: focus on Ant and particle swarm optimization. Tech Education and Publishing, pp 113–144
25. Karaboga D, Gorkemli B, Ozturk C, Karaboga N (2014) A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artif Intell Rev 42(1):21–57
26. Karaboga D, Ozturk C (2010) Fuzzy clustering with artificial bee colony algorithm. Sci Res Essays 5(41):1899–1902

27. Ozturk C, Karaboga D, Gorkemli B (2011) Probabilistic dynamic deployment of wireless sensor networks by artificial bee colony algorithm. Sensors 11(6):6056–6065
28. Tsai PW, Pan JS, Liao BY, Chu SC (2009) Enhanced artificial bee colony optimization. Int J Innov Comput Inf Control 5(12):5081–5092
29. UCI (2014) UCI Libraries. [online]. Available from: http://www.lib.uci.edu/collections/databases-a-to-z.html
30. Teodorovic D (2009) Bee colony optimization (BCO). In: Lim CP, Jain LC, Dehuri S (eds) Innovations in swarm intelligence. Springer-Verlag, Berlin, Heidelberg, pp 39-60
31. Karaboga D (2005) An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department
32. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471
33. Karaboga D, Basturk B (2008) On the performance of artificial bee colony(ABC) algorithm. Appl Soft Comput 8:687–697
34. Karaboga D, Akay B (2009) A comparative study of artificial bee colony algorithm. Appl Math Comput 214:108–132
35. Sun J, Zhang Q, Tsang EP (2005) DE/EDA: a new evolutionary algorithm for global optimization. Inf Sci 169(3):249–262
36. Lu ML, Xu BL, Sheng AD et al (2014) Modeling analysis of ant system with multiple tasks and its application to spatially adjacent cell state estimate. Appl Intell 41(1):13–29
37. Zhang R, Wu C (2010) A hybrid approach to large-scale job shop scheduling. Appl Intell 32(1):47–59
38. Özbakir L, Baykasoğlu A, Tapkan P (2010) Bees algorithm for generalized assignment problem. Appl Math Comput 215:3782–3795
39. Yeh WC, Hsieh TJ (2011) Solving reliability redundancy allocation problems using an artificial bee colony algorithm. Comput Oper Res 38:1465–1473
40. Liu H, Tang MX (2006) Evolutionary design in a multi-agent design environment. Appl Soft Comput 6(2):207–220
41. Liu H (2010) Context-aware agents in cooperative design environment. Int J Comput Appl Technol 39(4):187–198
42. Liu H, Tang MX, Frazer JH (2004) Supporting dynamic management in a multi-agent collaborative design system. Adv Eng Softw 35(8-9):493–502

Hong Liu is now a Professor of computer science at the School of Information Science and Engineering, Shandong Normal University. She received PhD degree from the Chinese Academy of Sciences in 1998. Her main research interests include computational intelligence and cooperative design.

Peng Zhang is a postgraduate of School of Information Science and Engineering, Shandong Normal University, Jinan, China. His current research interests include Swarm Intelligence, Optimization Algorithm and Computer Aided Design.

Bin Hu is now a Professor at the School of Information Science and Engineering, Lanzhou University. He is also a Taishan Scholar in Shandong Normal University. He received PhD degree from the Chinese Academy of Sciences in 1998. His main research interests include pervasive aware computing and cooperative design.

Philip Moore is now a Professor at the School of Information Science and Engineering, Lanzhou University. He is also a guest professor in Shandong Normal University. He received PhD degree from Fukuoka Institute of Technology, Japan. His main research interests include intelligent context aware systems and data fusion with intelligent context processing.