A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization Problems

Wei-Neng Chen, *Student Member, IEEE,* Jun Zhang, *Senior Member, IEEE,* Henry S. H. Chung, *Senior Member, IEEE,* Wen-Liang Zhong, Wei-Gang Wu, *Member, IEEE,* and Yu-hui Shi, *Senior Member, IEEE*

***Abstract*—Particle swarm optimization (PSO) is predominately used to find solutions for continuous optimization problems. As the operators of PSO are originally designed in an *n*-dimensional continuous space, the advancement of using PSO to find solutions in a discrete space is at a slow pace. In this paper, a novel set-based PSO (S-PSO) method for the solutions of some combinatorial optimization problems (COPs) in discrete space is presented. The proposed S-PSO features the following characteristics. First, it is based on using a set-based representation scheme that enables S-PSO to characterize the discrete search space of COPs. Second, the candidate solution and velocity are defined as a crisp set, and a set with possibilities, respectively. All arithmetic operators in the velocity and position updating rules used in the original PSO are replaced by the operators and procedures defined on crisp sets, and sets with possibilities in S-PSO. The S-PSO method can thus follow a similar structure to the original PSO for searching in a discrete space. Based on the proposed S-PSO method, most of the existing PSO variants, such as the global version PSO, the local version PSO with different topologies, and the comprehensive learning PSO (CLPSO), can be extended to their corresponding discrete versions. These discrete PSO versions based on S-PSO are tested on two famous COPs: the traveling salesman problem and the multidimensional knapsack problem. Experimental results show that the discrete version of the CLPSO algorithm based on S-PSO is promising.**

***Index Terms*—Combinatorial optimization problem, discrete space, multidimensional knapsack problem, particle swarm optimization, traveling salesman problem.**

I. Introduction

PARTICLE swarm optimization (PSO), which was first developed by Kennedy and Eberhart in 1995 [1], is a population-based stochastic algorithm for continuous optimization. The algorithm is inspired by the social interaction behavior of birds flocking and fish schooling. To search for the optimal solution, each individual, which is typically called a "particle," updates its flying velocity and current position iteratively according to its own flying experience and the other particles' flying experience. By now, PSO has become one of the most popular optimization techniques for solving continuous optimization problems [1]–[6].

The original PSO is simple and efficient, but it is predominately used to find solutions in a continuous space. As many optimization problems are defined in the discrete space, research on extending the PSO to solve discrete combinatorial optimization problems (COPs) has become an attractive subject in recent years.

The first attempt to extend PSO for discrete problems is the binary PSO (BPSO) algorithm proposed by Kennedy and Eberhart [7] based on the binary coding scheme. Afterward, the algorithm was improved by the discrete multiphase PSO [8] and the angle modulated PSO [9]. Recently, Clerc designed several BPSO algorithms with different strategies (please refer to the source codes in http://clerc.maurice.free.fr/pso/). These BPSO algorithms have shown promising performance on some benchmark instances, but the binary coding scheme has applications in limited types of optimization problems in the discrete space.

In order to define a more general frame for a discrete PSO (DPSO), several approaches have been developed in the literature. These approaches can be classified into four types. The first type is the swap-operator-based DPSO algorithm outlined by Clerc [10]. The algorithm defines the position of a particle as a permutation of numbers and the velocity as a set of swaps. A similar approach was also proposed by Wang *et al.* [11] for the traveling salesman problem (TSP). The second type of DPSO algorithms is characterized by a space transformation technique [12]–[14]. In these algorithms, the position is defined as a real vector, and thus a space transformation technique is used to convert the position into its corresponding solution. A common space transformation method is to view the position as a priority-based list [12]–[14]. The third type of DPSO algorithms defines the position and velocity as a fuzzy matrix [15]–[17]. To obtain a feasible solution to the problem, the algorithms need a defuzzification

Manuscript received December 29, 2007; revised July 06, 2009. First version published October 30, 2009; current version published March 31, 2010. This paper was supported in part by the National Science Foundation of China under Project 60573066, the National Natural Science Foundation of China Joint Fund with Guangdong under Key Project U0835002, the National High Technology Research and Development Program of China, No. 2009AA01Z208, and the Suzhou Science and Technology Project under Grant SYJG0919.

W.-N. Chen, J. Zhang, and W.-G. Wu are with the Department of Computer Science, Sun Yat-sen University, Guangzhou 510275, China (e-mail: junzhang@ieee.org).

H. S. H. Chung is with the Department of Electronic Engineering, City University of Hong Kong, Kowloon Tong, Hong Kong, and also with e.Energy Technology Ltd., Kowloon, Hong Kong.

W.-L. Zhong is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong.

Y.-h. Shi is with the Department of Electrical and Electronic Engineering, Xi'an Jiaotong-Liverpool University, Suzhou 215123, China (e-mail: yuhuishi@hotmail.com).

Digital Object Identifier 10.1109/TEVC.2009.2030331

method to decode the fuzzy matrix into a feasible solution. In addition to the above three types of "pure" DPSO approaches, the fourth type of DPSO approaches works by incorporating some meta-heuristics [18], [19] or problem-dependent local search techniques [20]–[27]. In other words, they are hybrid DPSO approaches.

Although various DPSO algorithms have been proposed, their performance is generally not satisfactory. The first three types of pure DPSO algorithms manage to follow the simple structure of the original PSO on the whole. However, compared with the other meta-heuristics for discrete optimization, the performance is not competitive. For example, in the TSP, the best results obtained by the pure DPSO approaches [10], [11], [14], [15], [17] are still far behind the ones obtained by the ant colony optimization (ACO) algorithm [28]. On the other hand, the hybrid DPSO algorithms perform better than the pure DPSO approaches. But as these hybrid DPSO algorithms are generally designed for specific problems, their structures are more complicated, and it is not easy to apply these hybrid algorithms to other COPs.

This paper proposes a set-based PSO (S-PSO) method to extend the application of PSO for solving some optimization problems in discrete space. Based on the concept of sets and the possibility theory, S-PSO has the following two features. First, a set-based representation scheme is designed to characterize the discrete search space as a universal set of elements. Second, each candidate solution in S-PSO corresponds to a crisp subset out of the universal set. A velocity is defined as a set with possibilities, that is, each element in a velocity is assigned with a possibility. All related arithmetic operators in the velocity and position updating rules in the original PSO are replaced by the operators and procedures defined on crisp sets and sets with possibilities. These features enable S-PSO to follow the simple structure of the original PSO to search solutions in a discrete space. As a result, the search behavior of S-PSO is very similar to that of the original PSO in continuous space. The parameters in the original PSO (e.g., the acceleration coefficients and the inertia weight [2], [29]) play a similar role in the proposed S-PSO. Moreover, different improved variants of the original PSO, e.g., the PSO with different topologies [30], [31] and the comprehensive learning PSO (CLPSO) [6] can be extended to their corresponding discrete versions based on S-PSO. The discrete PSO versions based on S-PSO are tested on two famous COPs, the TSP and the multidimensional knapsack problem (MKP). In the experiments, the algorithms are compared with both existing PSO-based approaches and some meta-heuristic algorithms [28]. Experimental results show that the discrete version of the CLPSO algorithm [6] based on S-PSO is promising.

The rest of this paper is organized as follows. Section II gives a brief review on the original PSO algorithms in the continuous space. In Section III, the proposed S-PSO is presented. Section IV further discusses the search behavior of S-PSO. Comparison studies are shown in Section V, and the conclusions are finally summarized in Section VI.

II. Particle Swarm Optimizers in Continuous Space

A. The Original PSO

In the original PSO [1], M particles cooperate to search for the global optimum in the n-dimensional search space. The ith ($i = 1, 2, \ldots, M$) particle maintains a position $\boldsymbol{X}_i(x_i^1, x_i^2, \ldots, x_i^n)$ and velocity $\boldsymbol{V}_i(v_i^1, v_i^2, \ldots, v_i^n)$. In each iteration, each particle uses its own search experience (self-cognitive) and the whole swarm's search experience (social-influence) to update the velocity and flies to a new position. The updating rules in [1] are as follows:

$$v_i^j \leftarrow v_i^j + c_1 r_1^j (pbest_i^j - x_i^j) + c_2 r_2^j (gbest^j - x_i^j) \tag{1}$$

$$x_i^j \leftarrow x_i^j + v_i^j \tag{2}$$

where $\boldsymbol{PBest}_i(pbest_i^1, pbest_i^2, \ldots, pbest_i^n)$ is the best solution yielded by the ith particle and $\boldsymbol{GBest}(gbest^1, gbest^2, \ldots, gbest^n)$ is the best-so-far solution obtained by the whole swarm. c_1 and c_2 are two parameters to weigh the importance of self-cognitive and social-influence, respectively. r_1^j and r_2^j are random numbers uniformly distributed in [0, 1], and j ($j = 1, 2, \ldots, n$) represents the ith dimension.

B. Successors of the Original PSO

As the original PSO is easy to implement and effective, a considerable amount of research effort has been made to improve its performance in various ways. Shi and Eberhart [2] first introduced an inertia weight ω to the velocity updating rule as follows:

$$v_i^j \leftarrow \omega v_i^j + c_1 r_1^j (pbest_i^j - x_i^j) + c_2 r_2^j (gbest^j - x_i^j). \tag{3}$$

Their work indicates that a relatively large inertia weight is better for global search, while a small inertia weight enhances the ability of local search. A scheme to decrease ω linearly from 0.9 to 0.4 over the course of search was also proposed in [2]. Later, Shi and Eberhart further designed fuzzy methods to adjust ω nonlinearly [3]. In [4], Clerc and Kennedy analyzed the convergence behavior of PSO in detail and introduced a constriction factor. The constriction factor is used to guarantee the convergence of the algorithm.

Another active research area is to develop different topologies. Kennedy and Mendes [30], [31] suggested using the local best position $\boldsymbol{LBest}_i(lbest_i^1, lbest_i^2, \ldots, lbest_i^n)$ of a neighborhood and modified the velocity updating rule into

$$v_i^j \leftarrow \omega v_i^j + c_1 r_1^j (pbest_i^j - x_i^j) + c_2 r_2^j (lbest_i^j - x_i^j). \tag{4}$$

The neighborhood can be defined by different topologies, such as Ring, URing, von Neumann [30], [31], random (refer to the random topology in the Standard PSO 2007 provided in http://www.particleswarm.info/Programs.html), and so on. In this paper, we term the global version GPSO, the local version with URing topology ULPSO, the local version with von Neumann topology VPSO, and the local version with random topology RPSO. In general, a large neighborhood is better for simple problems, and a small neighborhood is more suitable for complex problems [2].

```
procedure S-PSO
  initialization;
  while terminal condition not met
      for each particle i  (i = 1,2,···, M)
          velocity updating;
          position updating;
      end for
  end while
end procedure
```

Fig. 1. Structure of the S-PSO.

Many researchers have developed variants of PSO to prevent premature convergence by modifying the learning strategies of particles [6] or combining PSO with other search techniques [33]–[35]. A representative PSO variant is the CLPSO proposed by Liang *et al*. [6]. The algorithm updates velocity using the following rule:

$$v_i^j \leftarrow \omega v_i^j + cr^j(pbest_{f_i(j)}^j - x_i^j) \tag{5}$$

where c is a parameter, r^j is a random number in [0, 1], and $pbest_{f_i(j)}^j$ means the jth dimension of the ***PBest*** position of the particle $f_i(j)$. $f_i(j)$ is given as follows. First, a random number $ran \in [0, 1]$ is generated. If ran is larger than a parameter Pc, then $f_i(j) = i$. Otherwise, the algorithm applies the tournament selection to two randomly selected particles. The particle with a better fitness value is selected as $f_i(j)$. In this sense, $pbest_{f_i(j)}^j$ can be the corresponding dimension of any particle's ***PBest*** position. CLPSO has been shown to be excellent for complex multimodal function optimization problems [6].

III. The Set-Based PSO

In this section, the set-based PSO method is described. To follow the idea of PSO to solve discrete problems, S-PSO applies a set-based representation scheme and redefines the term "position" and "velocity" and all related operators for the discrete space. The structure of the algorithm is similar to the original PSO, which is shown in Fig. 1. To facilitate understanding, the rest of this section will take the TSP and the MKP for example in the description. The definitions of the TSP and the MKP are given in Fig. 2.

A. Representation Scheme

We first give a formal description of the representation scheme in S-PSO. In general, a COP can be defined by a triple (PS, f, Ω), where PS is the set of all candidate solutions, f is the objective function, and Ω is the set of constraints. The goal of the problem is to find a global optimal feasible solution $X^* \in PS$ that satisfies Ω and optimizes f.

According to [36], many COPs can be formulated in the abstract as "find from a set E a subset X that satisfies some constraints Ω and optimizes the objective function f." In terms of this formulation scheme, in S-PSO, the problem (PS, f, Ω) is mapped to a problem that includes the following characteristics.

1) A universal set E of elements is given. The universal set E can be divided into n dimensions, i.e., $E = E^1 \bigcup E^2 \bigcup \cdots \bigcup E^n$ (see the examples followed for illustration).
2) A candidate solution to the problem $X \in PS$ corresponds to a subset of E, that is, $X \subseteq E$. X can also be divided into n dimensions, i.e., $X = X^1 \bigcup X^2 \bigcup \cdots \bigcup X^n$, where $X^j \subseteq E^j$.
3) X is a feasible solution only if X satisfies the constraints Ω.
4) The objective of the problem is to find a feasible solution X^* that optimizes f.

According to the above characteristics, the COP is converted into a problem of choosing a subset of elements to form a subset of E that optimizes the objective function.

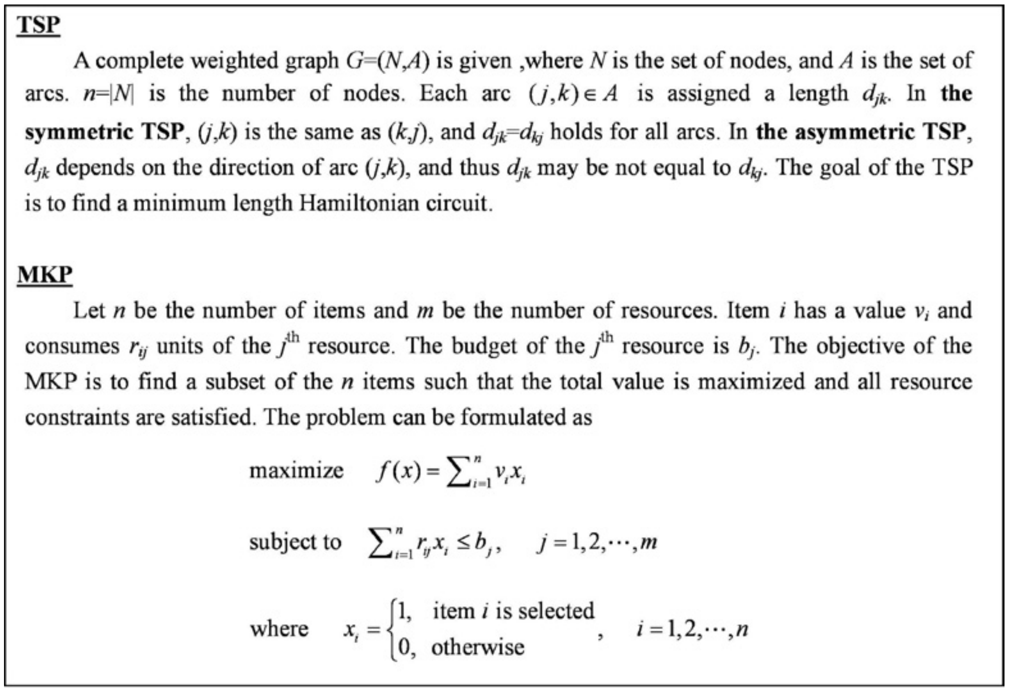
With the above representation scheme, for example, in the symmetric TSP, each arc (j, k) can be viewed as an element. The universal set E corresponds to the set A of arcs. Dimension E^j is composed of the arcs that are connected with node j. A candidate solution $X \subseteq E$ is a subset of arcs. $X = X^1 \bigcup X^2 \bigcup \cdots \bigcup X^n$, where X^j is a subset of E^j with two arcs. This is because there are always two arcs connected with j in a feasible solution. X is feasible if the arcs in X form a Hamiltonian circuit of the graph. Note that since arc (j, k) is equal to arc (k, j), dimensions E^j and E^k share the same arc (j, k). In this situation, different dimensions X^j (j = 1, 2, ..., n) must be consistent, i.e., if arc (j, k) belongs to X^j, (j, k) must belong to X^k. Fig. 3(a) gives an example of the representation scheme for the symmetric TSP. Differently, in the asymmetric TSP, as (j, k) is not equal to (k, j), dimension E^j only includes the arcs that start from node j, and thus different dimensions no longer share the same arc. An example of the representation scheme for the asymmetric TSP is given in Fig. 3(b). In the MKP, each dimension E^j can be defined as $E^j = \{(j, 0), (j, 1)\}$, where (j, 1) means that item j is chosen and (j, 0) indicates the opposite. A candidate solution is $X = X^1 \bigcup X^2 \bigcup \cdots \bigcup X^n$, where $X^j = \{(j, 0)\}$ or $X^j = \{(j, 1)\}$. X is feasible if it satisfies the capacity constraints. An example of the representation scheme for the MKP is given in Fig. 3(c).

In fact, the TSP and the MKP are two representative COPs of two different types. A solution of the TSP is a permutation of n components and the complexity is $O[(n - 1)!)]$. A solution of the MKP is a subset of n components and the complexity is $O(2^n)$. The COPs of these types can also be mapped to the problems to which S-PSO can apply in a similar way.

B. Velocity Updating

In PSO, a velocity gives the moving direction and tendency of a particle. In the original PSO algorithms in the continuous space, particles use the information of the best solutions found previously to adjust their velocities so that the velocities can direct particles to move to better positions. The velocity updating rule in S-PSO follows the same idea. For example, the velocity updating rule in the discrete version of GPSO based on S-PSO is given by

$$V_i^j \leftarrow \omega V_i^j + c_1 r_1^j(PBest_i^j - X_i^j) + c_2 r_2^j(GBest^j - X_i^j) \tag{6}$$



TSP

A complete weighted graph $G=(N,A)$ is given ,where N is the set of nodes, and A is the set of arcs. $n=|N|$ is the number of nodes. Each arc $(j,k)\in A$ is assigned a length d_{jk}. In **the symmetric TSP**, (j,k) is the same as (k,j), and $d_{jk}=d_{kj}$ holds for all arcs. In **the asymmetric TSP**, d_{jk} depends on the direction of arc (j,k), and thus d_{jk} may be not equal to d_{kj}. The goal of the TSP is to find a minimum length Hamiltonian circuit.

MKP

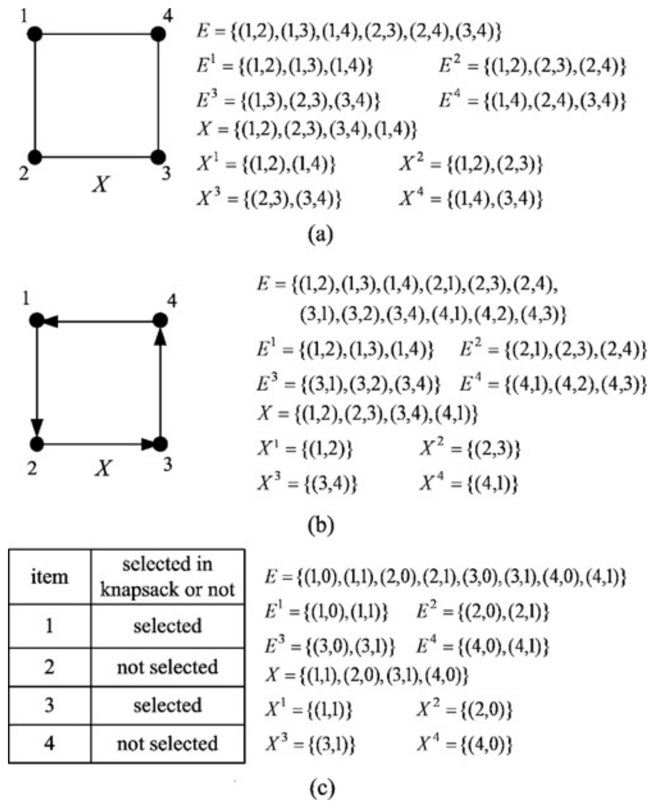
Let n be the number of items and m be the number of resources. Item i has a value v_i and consumes r_{ij} units of the j^{th} resource. The budget of the j^{th} resource is b_j. The objective of the MKP is to find a subset of the n items such that the total value is maximized and all resource constraints are satisfied. The problem can be formulated as

$$\text{maximize} \quad f(x)=\sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_{ij} x_i \le b_j, \quad j=1,2,\cdots,m$$

$$\text{where} \quad x_i=\begin{cases}1, & \text{item } i \text{ is selected}\\ 0, & \text{otherwise}\end{cases}, \quad i=1,2,\cdots,n$$

Fig. 2. Description of the TSP and the MKP.


1
4
2
3
X
$E=\{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$
$E^1=\{(1,2),(1,3),(1,4)\}$ $E^2=\{(1,2),(2,3),(2,4)\}$
$E^3=\{(1,3),(2,3),(3,4)\}$ $E^4=\{(1,4),(2,4),(3,4)\}$
$X=\{(1,2),(2,3),(3,4),(1,4)\}$
$X^1=\{(1,2),(1,4)\}$ $X^2=\{(1,2),(2,3)\}$
$X^3=\{(2,3),(3,4)\}$ $X^4=\{(1,4),(3,4)\}$
(a)
1
4
2
3
X
$E=\{(1,2),(1,3),(1,4),(2,1),(2,3),(2,4),(3,1),(3,2),(3,4),(4,1),(4,2),(4,3)\}$
$E^1=\{(1,2),(1,3),(1,4)\}$ $E^2=\{(2,1),(2,3),(2,4)\}$
$E^3=\{(3,1),(3,2),(3,4)\}$ $E^4=\{(4,1),(4,2),(4,3)\}$
$X=\{(1,2),(2,3),(3,4),(4,1)\}$
$X^1=\{(1,2)\}$ $X^2=\{(2,3)\}$
$X^3=\{(3,4)\}$ $X^4=\{(4,1)\}$
(b)
$E=\{(1,0),(1,1),(2,0),(2,1),(3,0),(3,1),(4,0),(4,1)\}$
$E^1=\{(1,0),(1,1)\}$ $E^2=\{(2,0),(2,1)\}$
$E^3=\{(3,0),(3,1)\}$ $E^4=\{(4,0),(4,1)\}$
$X=\{(1,1),(2,0),(3,1),(4,0)\}$
$X^1=\{(1,1)\}$ $X^2=\{(2,0)\}$
$X^3=\{(3,1)\}$ $X^4=\{(4,0)\}$
(c)

item	selected in knapsack or not
1	selected
2	not selected
3	selected
4	not selected

Fig. 3. Examples of the representation scheme for the TSP and the MKP. (a) Scheme for the symmetric TSP. (b) Scheme for the asymmetric TSP. (c) Scheme for the MKP.

where V_i^j is the jth dimension of the ith particle's velocity and X_i^j is the jth dimension of the ith particle's position. $PBest_i^j$ is the jth dimension of the best-so-far solution found by particle i, and $GBest^j$ is the jth dimension of the best-so-far solution found by all particles. $c_1 > 0, c_2 > 0$, and $\omega \in [0,1]$ are parameters, and $r_1^j \in [0,1]$ and $r_2^j \in [0,1]$ are random numbers. Obviously, the velocity updating rule (6) is in the same format as rule (3), but the positions, velocities and all related arithmetic operators in rule (6) are redefined in the discrete space as follows.

1) *Position:* A position is a feasible solution to the problem. In terms of the representation scheme given above, a feasible solution corresponds to a subset of elements. We denote the position of the ith particle as $X_i(X_i \subseteq E)$. The position is composed of n dimensions, i.e., $X_i = X_i^1 \bigcup X_i^2 \bigcup \cdots \bigcup X_i^n$, and $X_i^j \subseteq E^j$ $(j=1,2,\ldots,n)$. Similarly, $PBest_i \subseteq E$, $GBest \subseteq E$, and $LBest_i \subseteq E$ are the ith particle's best-so-far position, global best-so-far position, and the ith particle's local best-so-far position, respectively. At the beginning of the algorithm, X_i is initialized with a randomly-generated feasible solution.
2) *Velocity:* In S-PSO, a velocity is defined as a set with possibilities.

   Definition 1 (Set With Possibilities): Let E be a crisp set. A set with possibilities V defined on E is given by

   $$V = \{e/p(e)|e \in E\} \tag{7}$$

   that is, each element $e \in E$ has a possibility $p(e) \in [0,1]$ in V. In the representation, if $p(e)=0$, we usually omit the item $e/p(e)$ in the set for short.

   Based on this definition, in S-PSO, the overall velocity $V_i = \{e/p(e)|e \in E\}$ for particle i is a set with possibilities defined on E. In the jth dimension, $V_i^j = \{e/p(e)|e \in E^j\}$ is a set with possibilities defined

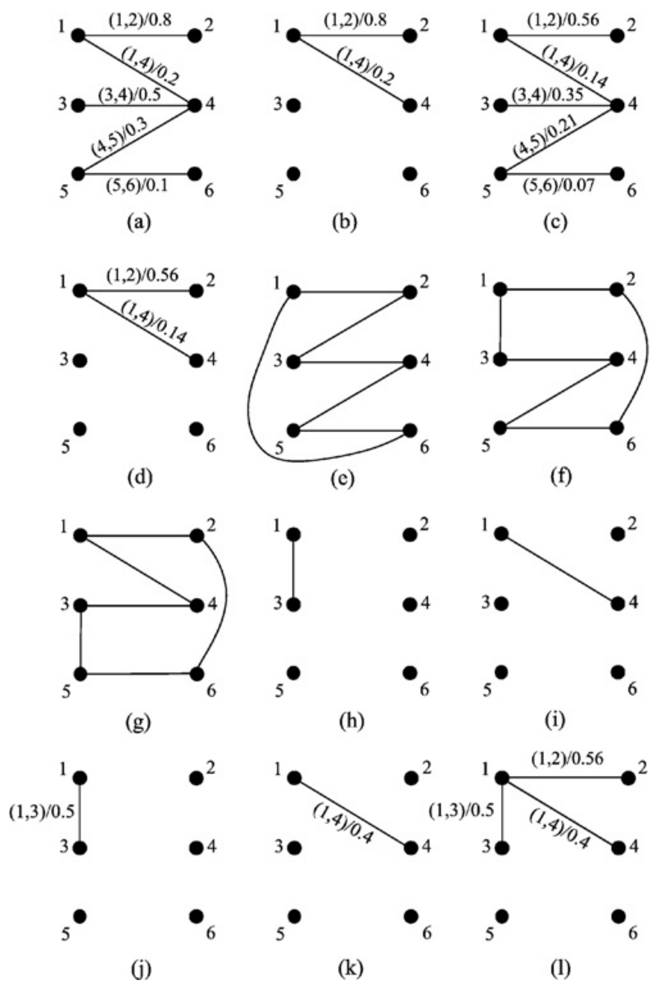


Fig. 4. Examples of the arithmetic operators in the velocity updating rule in S-PSO. (a) V_i. (b) V_i^1. (c) ωV_i (ω = 0.7). (d) ωV_i^1 (ω = 0.7). (e) X_i. (f) $GBest$. (g) $PBest_i$. (h) $GBest^1 - X_i^1$. (i) $PBest_i^1 - X_i^1$. (j) $c_2r_2\,(GBest^1 - X_i^1)$ (c_2r_2 = 0.5). (k) $c_1r_1\,(PBest_i^1 - X_i^1)$ (c_1r_1 = 0.4). (l) V_i^1 after updated.

on E^j. We will see later that the possibility $p(e)$ in V_i actually gives the possibility that particle i will learn from element e to build new positions.

Take the TSP for example, in Fig. 4(a), we have a velocity $V_i = \{(1, 2)/0.8, (1, 4)/0.2, (3, 4)/0.5, (4, 5)/0.3, (5, 6)/0.1\}$. The jth dimension of velocity V_i is composed of the arcs connected with node j and their corresponding possibilities, i.e., V_i^1={(1, 2)/0.8, (1, 4)/0.2} [Fig. 4(b)], V_i^2={(1, 2)/0.8}, V_i^3={(3, 4)/0.5}, and so on. At the beginning of the algorithm, V_i is initialized by randomly selecting n elements from the universal set E and assigning each of these elements a random possibility $p(e) \in (0, 1]$. The possibilities of the other unselected elements are set to 0. Basically, the initialization of velocities does not influence the search behavior and the performance of the algorithm, which will be discussed in the next section.

3) *Coefficient × Velocity:* the term coefficient here is used to denote a parameter or a random number which is a nonnegative real number. In S-PSO, the product of a coefficient and a set with possibilities is defined as follows.

Definition 2 (Multiplication Operator Between a Coefficient and a Set With Possibilities): Given a coefficient c ($c \geq 0$) and a set with possibilities $V = \{e/p(e)|e \in E\}$, their product is defined as

$$cV = \{e/p'(e)|e \in E\},$$
$$p'(e) = \begin{cases} 1, & \text{if } c \times p(e) > 1 \\ c \times p(e), & \text{otherwise} \end{cases} \quad (8)$$

where, for example, suppose the velocity in Fig. 4(a) is multiplied by ω = 0.7, and we have ωV_i={(1, 2)/0.56, (1, 4)/0.14, (3, 4)/0.35, (4, 5)/0.21, (5, 6)/0.07} [Fig. 4(c)], ωV_i^1= {(1, 2)/0.56, (1, 4)/0.14} [Fig. 4(d)], ωV_i^2={(1, 2)/0.56}, and so on.

4) *Position−Position:* in the representation scheme of S-PSO, a position is given by a crisp set. S-PSO follows the traditional definition of the minus operator between two crisp sets. Given two crisp sets A and B, the relative complement A–B of B in A is given by

$$A - B = \{e|e \in A \text{ and } e \notin B\}. \quad (9)$$

In terms of this definition, for example, given X_i^1={(1, 2), (1, 6)} and $GBest^1 = \{(1, 2), (1, 3)\}$, we have $GBest^1 - X_i^1 = \{(1, 3)\}$. In fact, the effect of $GBest^j - X_i^j$ and $PBest_i^j - X_i^j$ in (6) is to find out the elements used by the $GBest$ (or $PBest$) position but not used by the current position X_i. Such elements may have great potential to improve X_i.

5) *Coefficient × (Position − Position):* the result of "Position − Position" operation is a crisp set. The multiplication operator between a coefficient and a crisp set is defined as follows.

Definition 3 (Multiplication Operator Between a Coefficient and a Crisp Set): Given a coefficient c ($c \geq 0$) and a crisp set E', E' is a subset of the universal set E. The multiplication operator between c and E' is defined as follows:

$$cE' = \{e/p'(e)|e \in E\},$$
$$p'(e) = \begin{cases} 1, & \text{if } e \in E' \text{ and } c > 1 \\ c, & \text{if } e \in E' \text{ and } 0 \leq c \leq 1 \\ 0, & \text{if } e \notin E'. \end{cases} \quad (10)$$

The effect of (10) is to convert the crisp set E' into a set with possibilities with respect to the coefficient c. Based on this definition, suppose $c_1r_1 = 0.4$ and $c_2r_2 = 0.5$ are two coefficients, $PBest_i^1 - X_i^1 = \{(1, 4)\}$ and $GBest^1 - X_i^1 = \{(1, 3)\}$, we have $c_1r_1(PBest_i^1 - X_i^1) = \{(1, 4)/0.4\}$ [Fig. 4(k)] and $c_2r_2(GBest^1 - X_i^1) = \{(1, 3)/0.5\}$ in [Fig. 4(j)].

6) *Velocity + Velocity:* finally, we define the plus operator between two sets with possibilities.

Definition 4 (Plus Operator Between Two Sets With Possibilities): Given two sets with possibilities $V_1 = \{e/p_1(e)|e \in E\}$ and $V_2 = \{e/p_2(e)|e \in E\}$ defined on E, $V_1 + V_2$ is defined as

$$V_1 + V_2 = \{e/\max(p_1(e), p_2(e))|e \in E\} \quad (11)$$

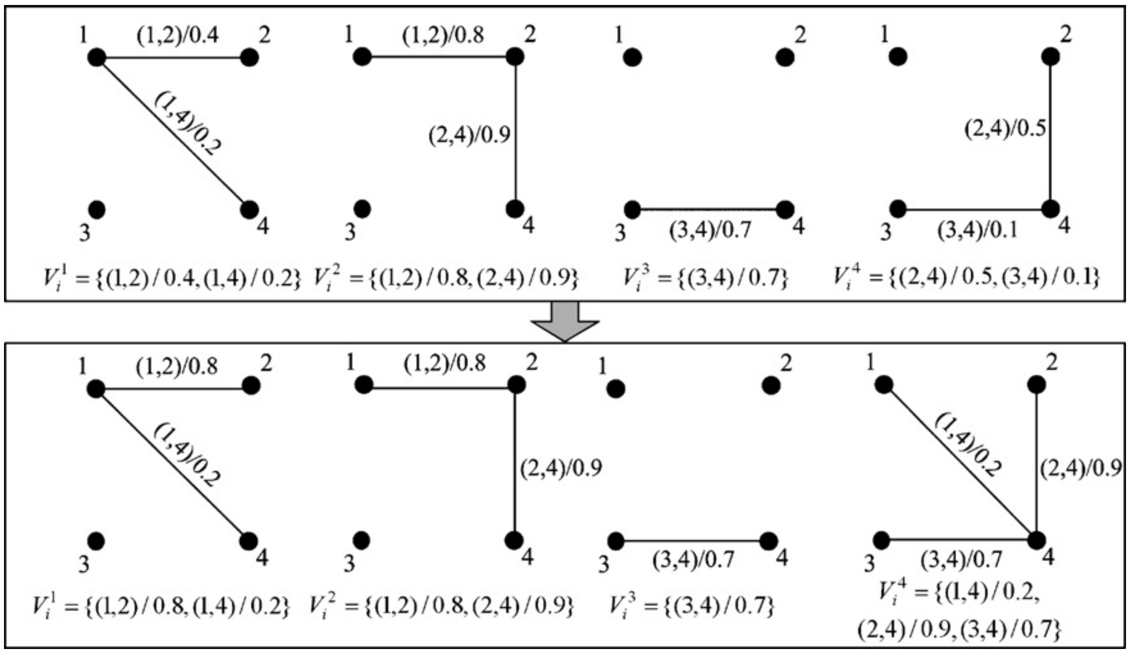


Fig. 5. Example of the union of velocities to avoid inconsistency.

that is, the possibility $p(e)$ in $V_1 + V_2$ is set to the larger one between $p_1(e)$ and $p_2(e)$.

For example, in Fig. 4, $\omega V_i^1 = \{(1, 2)/0.56, (1, 4)/0.14\}$, $c_1r_1(PBest_i^1 - X_i^1) = \{(1, 4)/0.4\}$, and $c_2r_2(GBest^1 - X_i^1)$= $\{(1, 3)/0.5\}$ are given. We have

$$\omega V_i^1 + c_1r_1(PBest_i^1 - X_i^1) + c_2r_2(GBest^1 - X_i^1)$$
$$= \{(1, 2)/0.56, (1, 3)/0.5, (1, 4)/0.4\}.$$

7) *Avoiding Inconsistency of the Velocities of Different Dimensions:* In some special COPs, e.g., in the symmetric TSP, an element (arc) (j, k) belongs to both dimensions j and k. In this situation, after updating all V_i^j $(j = 1, 2, \ldots, n)$ for particle i, different dimensions of V_i may be inconsistent. That is, for an element (j, k) [$(j, k) \in E^j$ and $(j, k) \in E^k$], it may occur that the possibility for (j, k) in V_i^j is not equal to the one in V_i^k. Suppose p_j (j, k) and p_k(j, k) are the possibilities for (j, k) in V_i^j and V_i^k respectively, we unify them as $p_j(j, k) = p_k(j, k) = p(j, k) = \max\{p_j(j, k), p_k(j, k)\}$. So the possibility for (j, k) in the new velocity V_i after implementing the velocity updating rule can be kept consistent.

An example is given in Fig. 5. In a 4-D symmetric TSP, after implementing the velocity updating rule, we have V_i^1=$\{(1, 2)/0.4, (1, 4)/0.2\}$, V_i^2=$\{(1, 2)/0.8, (2, 4)/0.9\}$, $V_i^3 = \{(3, 4)/0.7\}$, and V_i^4=$\{(2, 4)/0.5, (3, 4)/0.1\}$. In order to keep the possibilities consistent, the velocities are modified to $V_i^1 = \{(1, 2)/0.8, (1, 4)/0.2\}$, $V_i^2 = \{(1, 2)/0.8, (2, 4)/0.9\}$, $V_i^3 = \{(3, 4)/0.7\}$, and $V_i^4 = \{(1, 4)/0.2, (2, 4)/0.9, (3, 4)/0.7\}$.

8) *The Velocity Updating Rules for Different Discrete PSO Variants:* Based on the above definitions, the velocity updating rule for the discrete version of GPSO based on S-PSO is summarized in Fig. 6.

The velocity updating rule of the other PSO variants can also be extended to discrete versions based on S-PSO in the same way. The velocity updating rules in the discrete PSO with different topologies and the CLPSO are as follows:

$$V_i^j \leftarrow \omega V_i^j + c_1 r_1^j (PBest_i^j - X_i^j) + c_2 r_2^j (LBest_i^j - X_i^j) \quad (12)$$

$$V_i^j \leftarrow \omega V_i^j + c r^j (PBest_{f_i(j)}^j - X_i^j) \quad (13)$$

where $LBest_i$ is the local best position of a neighborhood and $PBest_{f_i(j)}^j$ is the jth dimension of particle $f_i(j)$'s *PBest* position. The neighborhood can be defined by any type of topologies. The parameters and the function $f_i(j)$ in (12) and (13) are the same as the ones in (4) and (5). In the rest of this paper, we name the discrete version of GPSO, VPSO, ULPSO, RPSO [30], [31], and CLPSO [6] based on the proposed S-PSO as S-GPSO, S-VPSO, S-ULPSO, S-RPSO, and S-CLPSO, respectively.

C. Position Updating

After updating the velocity, particle i uses the new velocity *Vi* to adjust its current position X_i and builds a new position NEW_X_i. Different from the case in the continuous space, the positions in the discrete space must satisfy the constraints Ω. To ensure the feasibility of the newly generated position *NEW_Xi*, in S-PSO, the ith particle applies the position updating procedure $position\_updating(X_i, V_i)$ given in Fig. 7 to build new positions. That is, in S-PSO, particle i updates its position by

$$X_i \leftarrow position\_updating(X_i, V_i). \quad (14)$$

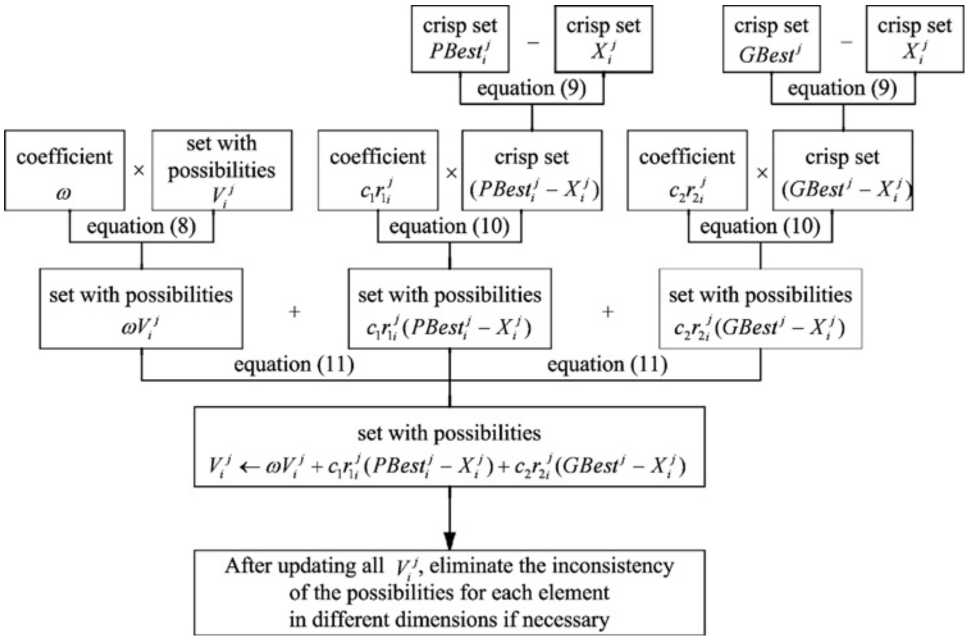

crisp set $PBest_i^j$
−
crisp set X_i^j
crisp set $GBest^j$
−
crisp set X_i^j
equation (9)
equation (9)
coefficient ω
×
set with possibilities V_i^j
coefficient $c_1 r_{1i}^j$
×
crisp set $(PBest_i^j - X_i^j)$
coefficient $c_2 r_{2i}^j$
×
crisp set $(GBest^j - X_i^j)$
equation (8)
equation (10)
equation (10)
set with possibilities ωV_i^j
+
set with possibilities $c_1 r_{1i}^j (PBest_i^j - X_i^j)$
+
set with possibilities $c_2 r_{2i}^j (GBest^j - X_i^j)$
equation (11)
equation (11)
set with possibilities $V_i^j \leftarrow \omega V_i^j + c_1 r_{1i}^j (PBest_i^j - X_i^j) + c_2 r_{2i}^j (GBest^j - X_i^j)$
After updating all V_i^j, eliminate the inconsistency of the possibilities for each element in different dimensions if necessary

Fig. 6. Arithmetic operators in the velocity updating rule of the S-GPSO algorithm.

```
procedure position_updating($X_i$, $V_i$)
    generate a random number $\alpha \in (0,1)$;
    for each dimension $j$ $(j = 1,2,\cdots,n)$
        $cut_\alpha(V_i^j) = \{e \mid e/p(e) \in V_i^j$ and $p(e) \geq \alpha\}$;
    end for
    $NEW\_X_i = \Phi$;
    for each dimension $j$ $(j = 1,2,\cdots,n)$
        $Candidate\_Set_i^j = \{e \mid e \in cut_\alpha(V_i^j)$ and $e$ satisfies $\Omega\}$;
        while the construction of $NEW\_X_i^j$ is not finished and $Candidate\_Set_i^j \neq \Phi$
            select an element from $Candidate\_Set_i^j$ and add it to $NEW\_X_i^j$;
            update $Candidate\_Set_i^j$;
        end while
        if the construction of $NEW\_X_i^j$ is not finished
            $Candidate\_Set_i^j = \{e \mid e \in X_i^j$ and $e$ satisfies $\Omega\}$;
            while the construction of $NEW\_X_i^j$ is not finished and $Candidate\_Set_i^j \neq \Phi$
                select an element from $Candidate\_Set_i^j$ and add it to $NEW\_X_i^j$;
                update $Candidate\_Set_i^j$;
            end while
        end if
        if the construction of $NEW\_X_i^j$ is not finished
            $Candidate\_Set_i^j = \{e \mid e \in E^j$ and $e$ satisfies $\Omega\}$;
            select the elements from $Candidate\_Set_i^j$ to complete $NEW\_X_i^j$;
        end if
    end for
    $X_i = NEW\_X_i$;
end procedure
```

Fig. 7. Pseudocode for the position updating procedure.

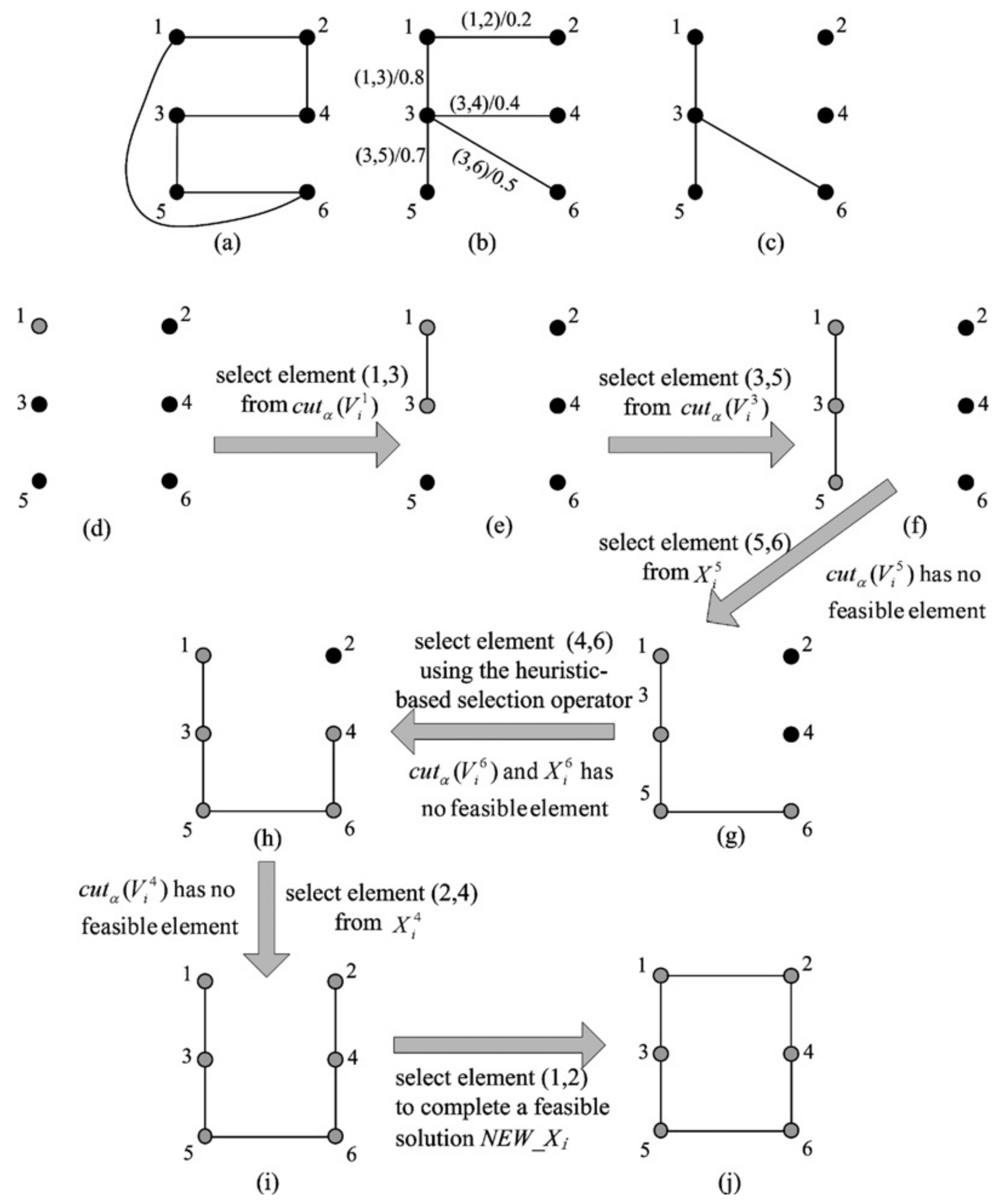

(a)
(1,2)/0.2
(1,3)/0.8
(3,4)/0.4
(3,5)/0.7
(3,6)/0.5
(b)
(c)
(d)
select element (1,3) from $cut_\alpha(V_i^1)$
(e)
select element (3,5) from $cut_\alpha(V_i^3)$
(f)
select element (5,6) from X_i^5
$cut_\alpha(V_i^5)$ has no feasible element
select element (4,6) using the heuristic-based selection operator
$cut_\alpha(V_i^6)$ and X_i^6 has no feasible element
(g)
(h)
$cut_\alpha(V_i^4)$ has no feasible element
select element (2,4) from X_i^4
(i)
select element (1,2) to complete a feasible solution NEW_X_i
(j)

Fig. 8. Example of the position updating procedure in the TSP.

1) *The Set with Possibilities V_i Is Converted Into a Crisp Set:* In the position updating procedure, particle i first finds out some elements from V_i which are desirable to be learnt from. In each iteration, a random number $\alpha \in (0, 1)$ is generated for each particle. For each element e in the jth dimension, if its corresponding possibility $p(e)$ in V_i^j is not smaller than α, element e is reserved in a crisp set, that is

$$cut_\alpha(V_i^j) = \{e|e/p(e) \in V_i^j \text{ and } p(e) \geq \alpha\}. \qquad (15)$$

This step is done in lines 1 to 4 in Fig. 7. Obviously, an element e with a larger $p(e)$ has a better chance to be reserved in $cut_\alpha(V_i^j)$. We will see later that particle i will learn from the elements in $cut_\alpha(V_i^j)$ to build a new position.

For example, in Fig. 8(b), given $V_i^1 = \{(1, 2)/0.2, (1, 3)/0.8\}$, if $\alpha = 0.5$, then we have $cut_\alpha(V_i^1) = \{(1, 3)\}$ [Fig. 8 (c)].

2) *Particle i Learns from the Elements in $cut_\alpha(V_i^j)$ to Build a New Position:* After generating $cut_\alpha(V_i^j)$, particle i builds a new position NEW_X_i by learning from the elements in $cut_\alpha(V_i^j)$. In S-PSO, the new position is built in a constructive way. The constraints Ω must be taken into account during the construction. At the beginning, the new position is set as an empty set (line 5 in Fig. 7). We denote the jth dimension of NEW_X_i as $NEW\_X_i^j$. For each dimension j, particle i first learns from the elements in $cut_\alpha(V_i^j)$ and adds them to $NEW\_X_i^j$ (lines 7 to 11 in Fig. 7). If the construction of $NEW\_X_i^j$ is not finished and there is no available element in $cut_\alpha(V_i^j)$, particle i reuses the elements in the previous X_i^j to build $NEW\_X_i^j$ (lines 12 to 18 in Fig. 7). If the construction of $NEW\_X_i^j$ is still not finished and there is no available element in the X_i^j, particle i uses the other available

elements to complete $NEW\_X_i^j$ (lines 19 to 22 in Fig. 7). After all $NEW\_X_i^j$ $(j = 1, 2, \ldots, n)$ have been completed, the construction of NEW_X_i is finished.

Note that there is a selection operator in the position updating rule (lines 9, 15, and 21 in Fig. 7). The operator can be a random selection, where the elements are randomly chosen, or a heuristic-based selection, where some problem-dependent information is applied to prefer better elements. Taking the TSP for example, we can employ the length of each arc as the heuristic information, and select the shortest arc from the candidate set. The performance of different selection operators will be discussed in the next section.

Based on the position updating procedure given in Fig. 7, we take the symmetric TSP for example to further illustrate the procedure. Note that there is a special phenomenon in the symmetric TSP. That is, an arc (j, k) is shared by both dimensions j and k. If (j, k) is added to $NEW\_X_i^j$, it is also added to $NEW\_X_i^k$. Though each dimension in a feasible solution has two arcs (each node is connected with two arcs), the whole solution (a Hamilton circuit) has only n arcs. In this situation, a special technique is applied to build a Hamilton circuit in a more natural way. At the beginning, the particle randomly chooses a dimension (node) to start the construction procedure. If the particle is located on node j, it follows the procedure given in Fig. 7 to choose only one arc (j, k) to add to $NEW\_X_i^j$. [Note that (j, k) is also added to $NEW\_X_i^k$.] Then the particle moves to node k and selects an arc (k, l) to add to $NEW\_X_i^k$ (and $NEW\_X_i^l$) in the same way. After all $NEW\_X_i^j$ $(j = 1, 2, \ldots, n)$ have been completed, the new Hamilton circuit NEW_X_i is built. To facilitate understanding, an example is illustrated in Fig. 8.

In the symmetric TSP, to build a Hamilton circuit in a natural way, we can start with a randomly selected dimension j and the next dimension k to be processed is determined by the selected arc (j, k). In the other COPs, the processing order of different dimensions can be specifically defined according to the characteristics of the COPs.

In the position updating procedure, Ω is considered during the construction of NEW_X_i. That is, e can be added to NEW_X_i only if Ω is satisfied. Thus the feasibility of NEW_X_i is guaranteed. In some special COPs, if it is not possible (or suitable) to judge whether a single element is feasible, or if there has been an effective method to convert an infeasible solution into a feasible one, we can also first build a new position NEW_X_i regardless of the constraints Ω, and then modify NEW_X_i into a feasible solution. In this case, the condition "e satisfies Ω" in lines 7 and 13 in Fig. 7 can be ignored. For example, in [32], an effective repair procedure has been proposed for the MKP. Based on this repair procedure, particles are able to first build solutions regardless of the capacity constraints, and then apply this method to ensure the feasibility of the solutions.

Note that different from the traditional PSO algorithms for the continuous space, in S-PSO, given two feasible solutions X and X' and a velocity $V = X - X'$, the position updating procedure cannot always guarantee $X = position\_updating(X', V)$. For example, in a 6-city symmetric TSP instance, suppose $X=\{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6),$ $(1, 6)\}$ and $X' = \{(1, 2), (1, 3), (3, 4), (4, 5), (5, 6), (2, 6)\}$, then we have $V = \{(1, 6)/1.0, (2, 3)/1.0\}$. According to the position updating procedure in Fig. 7, when building the first dimension of the new position, as $cut_\alpha(V^1) = \{(1, 6)\}$ $(\alpha \in (0, 1))$, the arc $(1, 6)$ will be added to the new position first. Because there are two arcs in each dimension of a feasible solution to the symmetric TSP, the other arc in the first dimension is selected from $X'_1 = \{(1, 2), (1, 3)\}$ based on the selection operator in line 15 in Fig. 7. In this case, if the arc $(1, 3)$ is chosen, the resulting new position will not be equal to X. In fact, due to various constraints in different COPs, it is sometimes not easy to rebuild the solution X by a simple procedure when only X' and V $(V = X - X')$ are given. Therefore, to make the procedure implementable, S-PSO applies the position updating procedure in Fig. 7 and $X = position\_updating(X', V)$ is not always guaranteed. However, as the position updating procedure first uses the elements in V and X' to build new solutions, the resulting new position of $position\_updating(X', V)$ is still close to (or sometimes equal to) X in general, and S-PSO still follows the idea of the original PSO to improve X' by V.

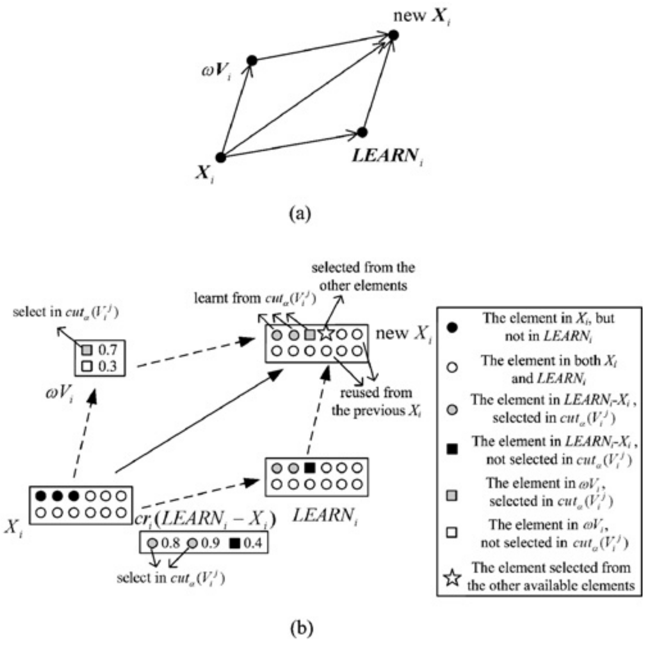

new X_i
ωV_i
$LEARN_i$
X_i
(a)
selected from the other elements
learnt from $cut_\alpha(V_i^j)$
select in $cut_\alpha(V_i^j)$
0.7
0.3
new X_i
ωV_i
reused from the previous X_i
X_i
$cr_i(LEARN_i - X_i)$
$LEARN_i$
0.8
0.9
0.4
select in $cut_\alpha(V_i^j)$
● The element in X_i, but not in $LEARN_i$
○ The element in both X_i and $LEARN_i$
The element in $LEARN_i$-X_i, selected in $cut_\alpha(V_i^j)$
■ The element in $LEARN_i$-X_i, not selected in $cut_\alpha(V_i^j)$
The element in ωV_i, selected in $cut_\alpha(V_i^j)$
□ The element in ωV_i, not selected in $cut_\alpha(V_i^j)$
☆ The element selected from the other available elements
(b)

Fig. 9. Comparison of the search mechanisms between the original PSO and the S-PSO. (a) Search mechanism of the original PSO. (b) Search mechanism of the S-PSO.

IV. Behaviors of S-PSO and Further Discussions

In this section, we further discuss the search behaviors of the proposed S-PSO. In order to analyze the characteristics of different DPSO variants based on the proposed S-PSO, we implemented the S-GPSO, S-VPSO, S-ULPSO, S-RPSO, and S-CLPSO algorithms using the TSP as an application problem. The performance of these five discrete PSO variants is compared. In the studies in this section, the default parameter settings are as follows: $c1 = c2 = 2.0$ ($c = 2.0$ in S-CLPSO), ω

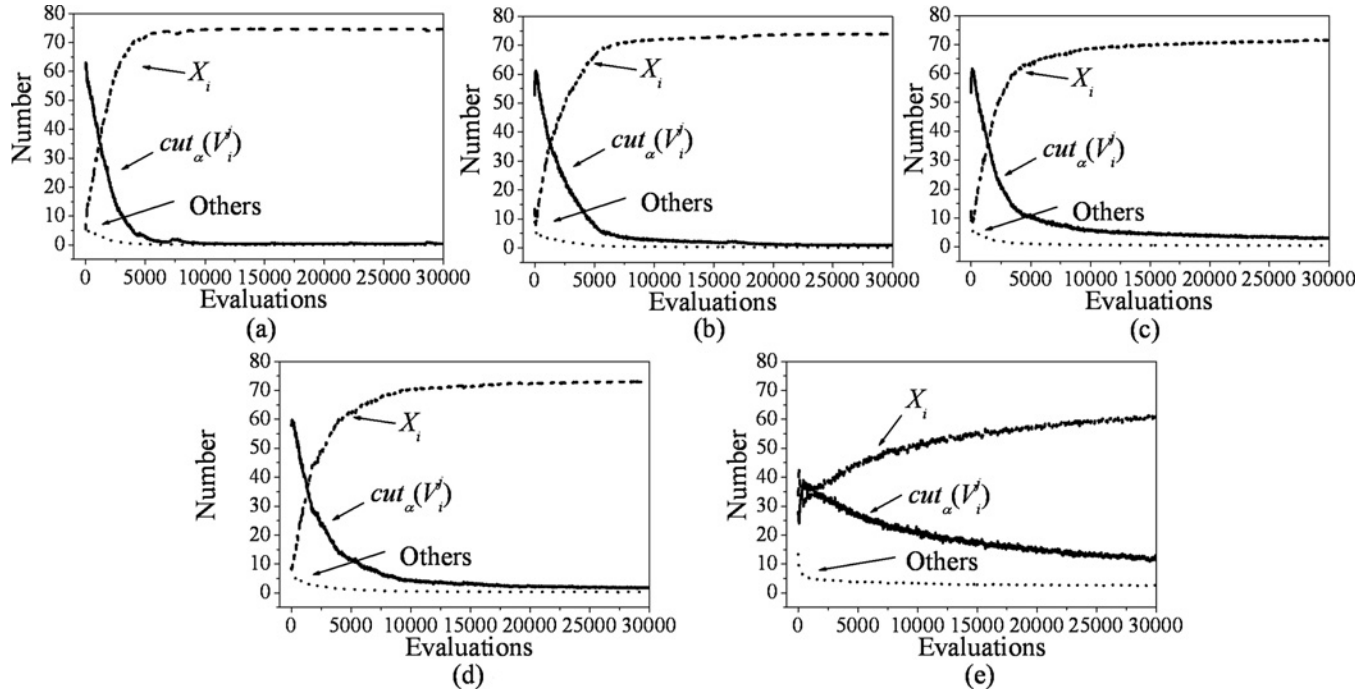


Fig. 10. Sources of the arcs that compose a new position in the TSP instance eil76. (a) S-GPSO. (b) S-VPSO. (c) S-ULPSO. (d) S-RPSO. (e) S-CLPSO.

is decreasing linearly from 0.9 to 0.4 [2], and the swarm size $M = 20$. In S-CLPSO, there are two additional parameters, i.e., the learning probability Pc and the refreshing gap rg. We follow the original CLPSO [6] to set $rg = 7$ and

$$Pc_i = 0.05 + 0.45 \times \frac{(\exp(\frac{10(i-1)}{M-1}) - 1)}{(\exp(10) - 1)} \quad (16)$$

where $i = 1, 2, \ldots, M$ is the ID of the particle. In general, the heuristic-based selection operator is applied in the position updating procedure. The effect of the above configurations will also be discussed in this section.

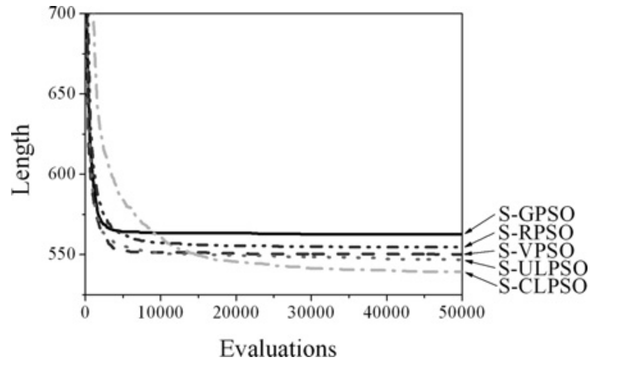


Fig. 11. Comparison of the convergence speeds among the five discrete PSO variants based on S-PSO in the TSP instance eil76. The results are averaged over 30 runs.

A. Behaviors of Different Discrete PSO Variants Based on the S-PSO

To study the search behaviors of S-PSO, we first give an insight into the essence of the velocity and the position updating rules in the S-PSO.

The comparison of the search mechanisms between the original PSO and the S-PSO is illustrated in Fig. 9. In the original PSO, velocity is composed of the inertia vector $\omega \boldsymbol{V}_i$ and the vector that learned from the previous search experience $cr(\boldsymbol{LEARN}_i - \boldsymbol{X}_i)$. Here, $\boldsymbol{LEARN}_i$ represents the position from which the particle learns, i.e., $\boldsymbol{LEARN}_i$ represents the $\boldsymbol{GBest}$, $\boldsymbol{LBest}_i$, or $\boldsymbol{PBest}_i$ position in different PSO variants. The particle uses the velocity vector to update the position as shown in Fig. 9(a).

The velocity and position updating rules in S-PSO work in a similar way. The velocity in S-PSO also includes the inertia ωV_i and the elements learnt from previous search experience $cr(LEARN_i - X_i)$. Here $LEARN_i$ represents $GBest$, $LBest_i$, or $PBest_i$. In terms of the definition of "position − position," the effect of $LEARN_i - X_i$ is actually to find out the elements that are used by the promising solution $LEARN_i$, but not used by X_i. Such elements may have great potential to improve X_i. The essence of S-PSO is to let particles learn from some of these promising elements from the previous V_i and $LEARN_i - X_i$ iteratively to improve their current positions, as illustrated in Fig. 9(b).

According to the position updating procedure in Fig. 7, the elements in a new position come from three sources. A particle first learns from the elements in $cut_\alpha(V_i^j)$. Then it reuses the elements in the previous position X_i^j. If there is no available element in $cut_\alpha(V_i^j)$ and X_i^j, the particle chooses the other available elements to complete the new positions.

The sources of the elements in a new position are tightly related to the convergence behavior of S-PSO. Basically, the fact that a large number of elements come from $cut_\alpha(V_i^j)$ implies a very "fast" flying speed, as the particle can learn from a lot of elements in $cut_\alpha(V_i^j)$. On the other hand, if the number of

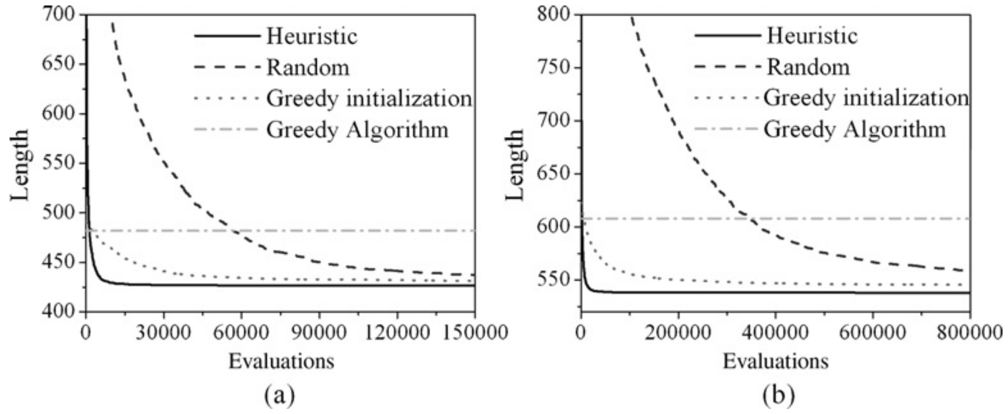


Fig. 12. Comparison between the random and the greedy selection operator in the TSP. The results are averaged over 30 runs. In the plots, "Heuristic" represents the algorithm with heuristic-based selection operator. "Random" represents the algorithm with random selection operator. "Greedy initialization" represents the algorithm with random selection operator, but the positions are initialized using the greedy algorithm. "Greedy algorithm" represents the best results obtained by the greedy algorithm. (a) eil51. (b) eil76.

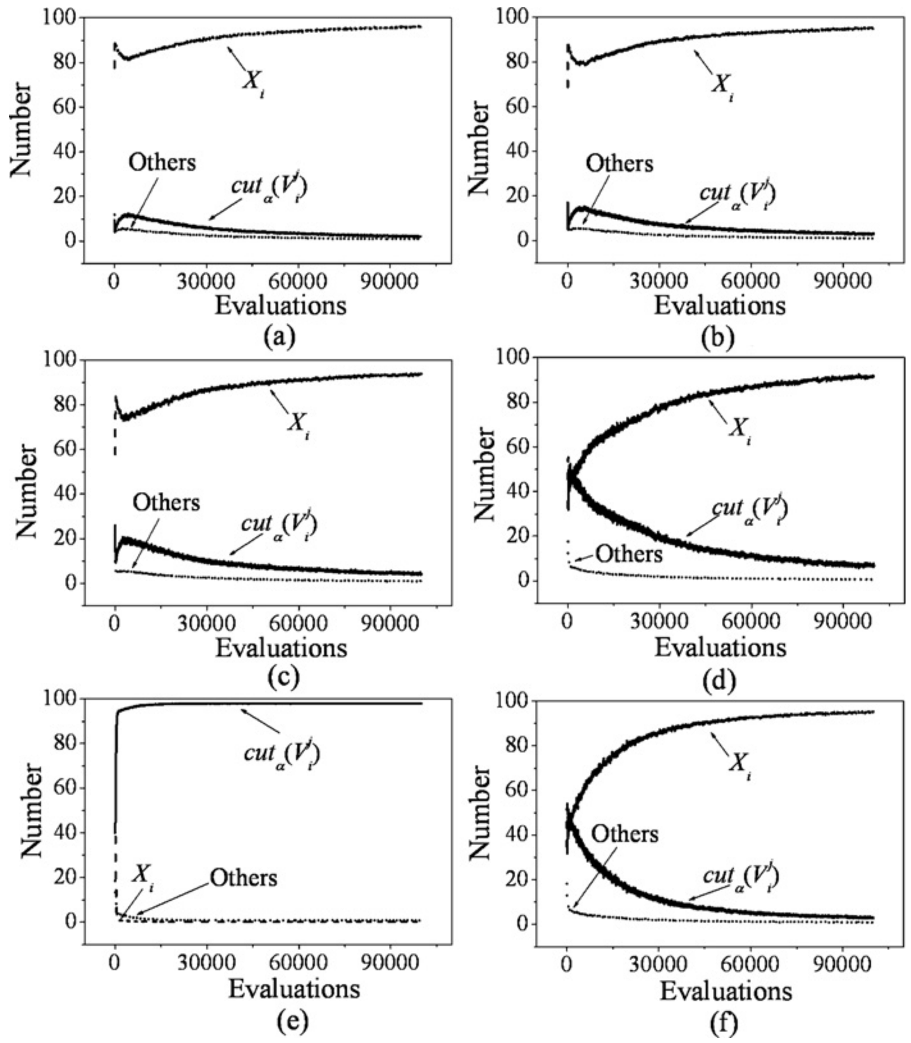


Fig. 13. Search behaviors of the algorithm with different inertia weight values in the TSP instance kroA100. (a) $\omega = 0.0$. (b) $\omega = 0.3$. (c) $\omega = 0.6$. (d) $\omega = 0.9$. (e) $\omega = 1$. (f) decreasing ω.

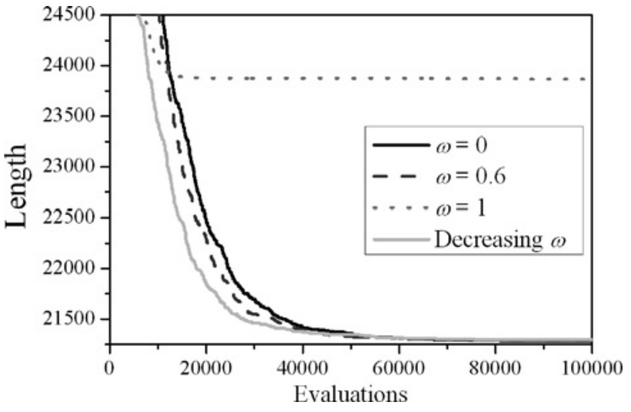


Fig. 14. Performance of the algorithm with different inertia weight values in the TSP instance kroA100. The results are averaged over 20 runs.

elements that come from $cut_\alpha(V_i^j)$ is small, the particle moves very slowly and only searches in a small neighborhood. We run the S-GPSO, S-VPSO, S-ULPSO, S-RPSO, and S-CLPSO algorithms on a TSP instance eil76 from traveling salesman problem library (TSPLIB) [37] (http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html). The sources of elements in new positions in every iteration are recorded. The typical results are plotted in Fig. 10. In general, the number of the arcs that come from neither $cut_\alpha(V_i^j)$ nor the previous *Xi* is very small. In all DPSO variants based on S-PSO, at the early stage, most of the arcs in the new positions come from $cut_\alpha(V_i^j)$. Therefore the search procedure shows a diverse behavior. As the procedure continues, the differences between $LEARN_i$ and X_i become small, and thus the number of elements in $cut_\alpha(V_i^j)$ reduces. As a result, more and more arcs come from the previous X_i, and fewer and fewer arcs come from $cut_\alpha(V_i^j)$. From Fig. 10, it can be seen that in the S-GPSO, S-VPSO, S-ULPSO and S-RPSO algorithms, after 5000 evaluations, more than 70 arcs come from the previous X_i. In this situation, the search has converged to a small area. In an extreme case, if all arcs come from the previous X_i, the search procedure stagnates and no better solutions can be further found.

Fig. 11 illustrates different convergence characteristics between the five DPSO variants based on S-PSO. Similar to the case in the continuous space, premature convergence is still the main deficiency of S-GPSO. In contrast, as has been pointed out by Liang *et al.* [6], the CLPSO algorithm manages to increase the swarm's diversity when solving complex multimodal problems. As COPs are usually complex problems with many local optima. The discrete version of CLPSO based on S-PSO (S-CLPSO) manages to increase the swarm's diversity, and outperforms the other discrete PSO versions based on S-PSO.

As the S-CLPSO algorithm is the most suitable variant, in the following discussions, we focus on the S-CLPSO algorithm.

B. The Select Operator in the Position Updating Procedure

In the position updating procedure (Fig. 7), there is a selection operator. The selection operator can be either random or heuristic-based. In the heuristic-based selection operator in the TSP, the shortest arc in the candidate set is selected. The performance of the S-CLPSO algorithm with the random and the heuristic-based selection operators is compared. The results are plotted in Fig. 12. Obviously, the algorithm with the heuristic-based selection operator manages to obtain acceptable solutions much faster. The reason is that the heuristic-based selection scheme employs some problem-dependent information to guide the search. In fact, many successful algorithms for COPs, for example, the ACO meta-heuristic [28], [39], also employ problem-dependent heuristic information. Such heuristic information is able to accelerate the search, especially in large-scale instances.

Note that selecting the shortest arc from the candidate set does not mean that the algorithm behaves in the same way as the greedy-search algorithm. The algorithm still follows the mechanism that particles learn from the elements in $cut_\alpha(V_i^j)$ to improve the previous X_i. Only if there are some elements in $cut_\alpha(V_i^j)$ that cannot be added to the new position $NEW\_X_i^j$ at the same time, the heuristic-based selection operator is applied to choose the best elements for particle i to learn. In Fig. 12, we also plot the best results that can be obtained by the greedy search algorithm. Compared with the S-CLPSO algorithm, although the search speed of the S-CLPSO algorithm with the random selection operator is rather slow, given a long enough time, the algorithm can also achieve much better results than the ones obtained by the greedy-search algorithm. This demonstrates that the learning mechanism in S-CLPSO is indeed contributing.

In some problems, it may be difficult to define an effective heuristic-based selection operator. In this situation, to accelerate the search speed, we can apply some deterministic techniques to generate good initial solutions at the beginning. For example, in the TSP, we use the greedy-search algorithm to generate initial solutions for all particles (the greedy-search algorithm begins at random cities to build different tours for different particles), but only use the random selection operator in the course of search. This method is labeled "greedy initialization" and the results are also plotted in Fig. 12. The figure reveals that this scheme also manages to achieve acceptable results quickly.

C. The Inertia Weight and Acceleration Coefficient in S-PSO

The most important parameters in the original PSO are the inertia weight ω and the acceleration coefficients (c_1 and c_2 in GPSO, and c in CLPSO). By defining the velocity as a set with possibilities, these parameters are able to play a similar role in S-PSO.

In S-PSO, each element e in a velocity is assigned with a possibility $p(e)$. Only the elements whose possibilities are not smaller than a random number $\alpha \in [0, 1]$ will be learnt by particles. According to the velocity updating rule, in each iteration, the possibility $p(e)$ for each element e that inherited from the previous velocity is multiplied by an inertia weight $\omega \in [0, 1]$. In this situation, after t iterations, $p(e)$ is reduced to $\omega^t p(e)$. In other words, the possibility for reserving e in $cut_\alpha(V_i^j)$ decreases exponentially in the number of iterations. A small value for ω will make particles forget the elements in the previous V_i quickly. In contrast, a large value for ω

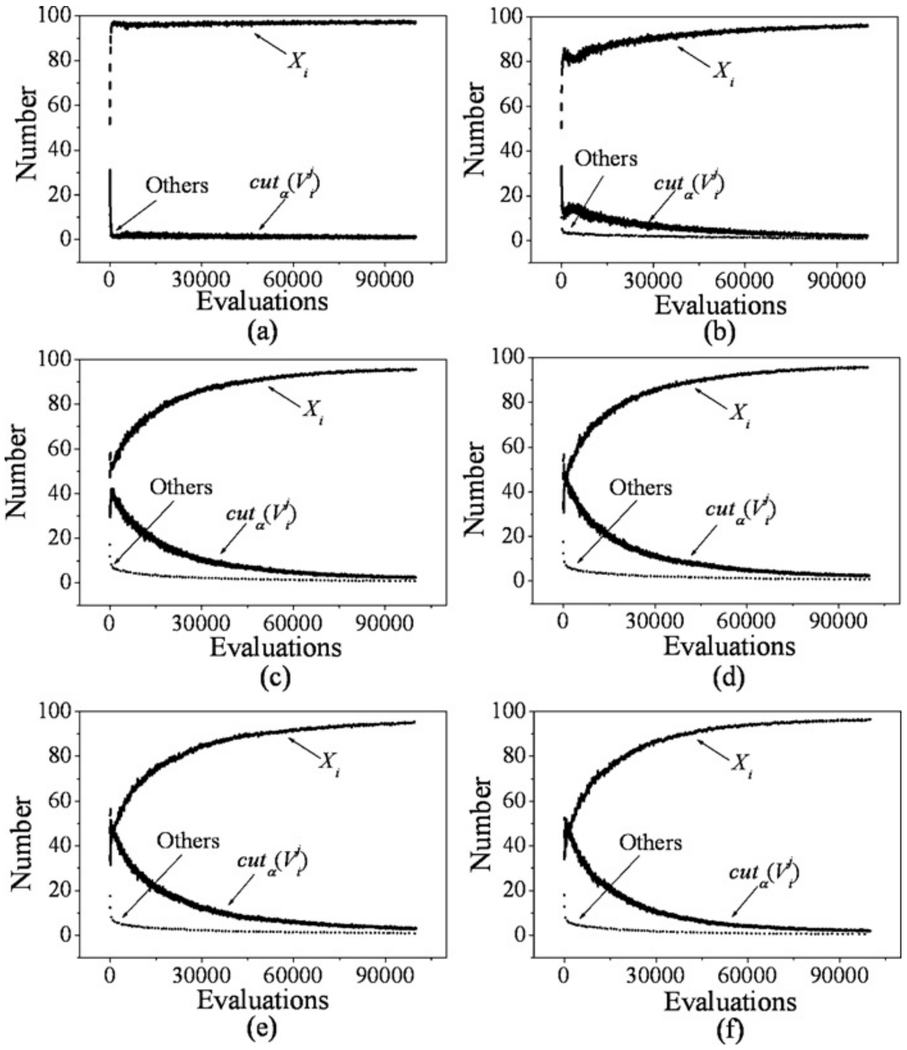


Fig. 15. Search behaviors of the algorithm with different values of acceleration coefficient in the TSP instance kroA100. (a) $c = 0.1$. (b) $c = 0.5$. (c) $c = 1.1$. (d) $c = 1.5$. (e) $c = 2.1$. (f) $c = 2.9$.

will reinforce the elements in the previous V_i. Figs. 13 and 14 report the search behaviors and performance of the algorithm with different ω values. It can be seen that if ω is small, there will be fewer elements in $cut_\alpha(V_i^j)$ for the particle to learn from. Oppositely, if $\omega = 1$, the elements in the previous V_i will never be forgotten, and thus there will be too many elements in $cut_\alpha(V_i^j)$ for the particle to learn from. In this case, the search is mainly determined by the selection operator in line 9 in Fig. 7, which will lead to very poor performance. We also test the scheme proposed by [2] to decrease the value of ω from 0.9 to 0.4. The results in Figs. 13 and 14 show that this setting manages to balance the convergence and diversity of the algorithm and performs well. On the other hand, as the influence of the previous velocity decreases exponentially in the number of iterations, the initialization of the velocities has little influence on the performance of the algorithm.

The acceleration coefficient c weighs the importance of the newly-learnt elements. Since each element e learnt from $LEARN_i - X_i$ is assigned with a possibility $c \times r$, a larger c will give a better chance for reserving e in $cut_\alpha(V_i^j)$, and vice versa. Fig. 15 shows that when c is small, particles learn from only a few elements in $cut_\alpha(V_i^j)$, and reuse most of the elements in the previous X_i. Consequently, as shown in Fig. 16, the performance of the algorithm with $c = 0.1$ and $c = 0.5$ is very poor. (The curve for $c = 0.1$ does not appear in Fig. 16 because its results are so poor that they are out of the scale of the figure.) When $c > 1$, the diversity and convergence of the algorithm can be balanced, and the algorithms perform quite well. Note that because the possibilities are limited in the interval [0, 1], when $c > 1$, the differences between different values of c are not significant.

According to the above discussions, the classical configurations for the inertia weight and acceleration coefficients in the original PSO are still effective in the proposed S-PSO.

Note that the value of $p(e)$ may become very small as it decreases exponentially with the number of iterations. In our implementation, if $p(e) < \varepsilon = 0.001$, we delete the item $e/p(e)$ from V_i to save storage space. Given any small enough positive value ε, if $p(e) < \varepsilon$, element e can hardly be learnt by the particle. Thus the implementation scheme to delete $e/p(e)$ if $p(e) < \varepsilon$ can hardly influence the search behavior of S-PSO, but can save storage space in the implementation.

D. The Swarm Size in S-PSO

The swarm size M can also influence the performance of the S-PSO algorithm. In general, a small swarm size is able to accelerate the search process, but the diversity of the population is reduced. On the other hand, a large swarm size brings higher diversity to the population but the computational effort in each single iteration significantly increases, and thus the search process becomes slow. Fig. 17(a) and (b) compares the performance of the algorithm with different swarm sizes for the TSP and the MKP. In the comparison, 100 000 solutions are generated, and the results are averaged over 50 runs. For the TSP, it can be seen that M = 20, 30, and 40, are able to achieve the best results. The performance of M = 10 is slightly worse. Because the maximum number of evaluations is limited in each run, fewer iterations can be executed by the algorithm with a large M. Therefore, the performance of M = 80 and 90 is poor. For the MKP, it is found that good performance is achieved when M is larger than 50. If M is small, the algorithm is more likely to be trapped in local optima. The different configurations of M for the TSP and the MKP may be caused by the different fitness landscapes of these two problems. In TSP, a large number of short edges are actually shared by different local optimal (including the global optimal) tours. By using the heuristic-based selection operator, S-CLPSO is still able to focus on searching the tours that are composed by those short edges even if the swarm size is small. On the other hand, in MKP, as there are multidimensional resource constraints, the local optimal solutions may be scattered in different places in the search space. In this case, the population has to maintain higher diversity to prevent from trapping in local optima. Thus a large M is more suitable for MKP.

V. Experimental Results and Comparison Studies

In this section, we present the numerical experimental results of the discrete PSO algorithm based on the proposed S-PSO. The experiments are done on two famous COPs, the TSP and the MKP. The TSP instances are derived from the TSPLIB [37]. The information of the instance is given in Table I. The MKP instances are extracted from the operations research library (ORLIB) (http://people.brunel.ac.uk/∼mastjjb/jeb/orlib). The experiments on the TSP without using the 3-opt local search operator are performed on a machine with Pentium IV 2.80 GHz CPU and 256 MB of RAM. The experiments on the TSP with the use of the 3-opt operator and the MKP are performed on a machine with Intel(R) Core(TM) 2 Quad CPU Q6600 at 2.40 GHz and 1.96 GB of RAM (but only a single processor is used due to the sequential implementation of the algorithm). The operating system is MS Windows XP and the compiler is VC++ 6.0. As has been mentioned before, the S-CLPSO algorithm is the best discrete PSO variant based on S-PSO. Here, we compare the S-CLPSO algorithm with the other existing PSO-based approaches and some other meta-heuristics.

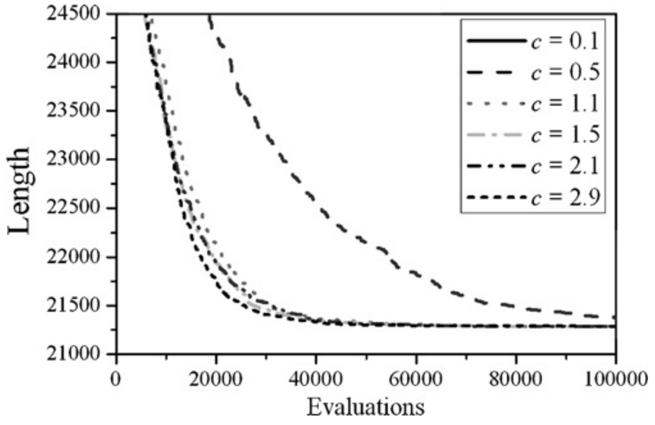


Fig. 16. Performance of the algorithm with different acceleration coefficient values in the TSP instance kroA100. The results are averaged over 20 runs.

In the experiments in this section, according to the analyses in Section IV, the configurations of decreasing ω from 0.9 to 0.4 linearly and setting c = 2.0 are used. As population sizes of the ant colony system (ACS) algorithm for TSP [28] and the binary PSO algorithm for MKP [47] are set to M = 10 and M = n (n is the number of items) respectively, to make fair comparisons, the population size of the proposed algorithm is also set M = 10 for TSP and M = n for MKP. Besides, there are two other parameters in the CLPSO algorithm proposed in [6], namely the learning probability Pc and the refreshing gap rg. We find that the classical settings given in [6] [i.e., Pc is set according to (16) and rg = 7] are able to yield good results for both TSP and MKP. For the experiments on TSP, to accelerate the search process, we use the heuristic-based selection operator to select short edges. For the experiments on MKP, only the random selection operator is used.

A. Performance on the TSP

We first compare the S-CLPSO algorithm with the other existing PSO-based algorithms for the TSP, i.e., the PSO-TS-CO-2opt algorithm [14], the discrete PSO algorithm [20], and the C3PSO algorithm proposed in [38]. In [14], the PSO-TS-CO-2opt algorithm is integrated with a 2-opt local search procedure and a chaotic operation. 100 000 solutions are generated in each single run and the results are averaged over ten runs. In [20], each instance is run 100 times. The best, the worst, and the mean results are recorded. In [38], 300n solutions are generated in each single run, where n is the size of the instance. The best and the average results over 50 runs are recorded. The S-CLPSO algorithm with the aforementioned parameter configurations and the heuristic-based selection operator is run 50 times. To make a fair comparison, in each run, the S-CLPSO algorithm also generates 300n solutions. From the results given in Table II, even the worst results obtained by S-CLPSO in 50 runs are better than the mean results obtained by the PSO-TS-CO-2opt algorithm. On the other hand, the results yielded by S-CLPSO after 300n evaluations are better than the results reported in [20] and [38]. The results reveal that the proposed algorithm is as competitive as these existing pure discrete PSO approaches for the TSP.

TABLE I
NUMBER OF SOLUTIONS GENERATED IN EACH RUN FOR EACH INSTANCE

TSP instances (TSPLIB)			
Name	Maximum Number of Solutions	Name	Maximum Number of Solutions
(Instances for the Experiments in Table III)			
eil51	25 000	eil101	50 000
Berlin52	25 000	lin105	50 000
st70	35 000	kroA150	75 000
eil76	35 000	kroA200	100 000
pr76	35 000	pr299	150 000
kroA100	50 000	lin318	150 000
(Large-Scale Instances for the Experiments in Table VII)			
d493	246 500	d657	328 500
u724	362 000	d1291	645 500
fl1400	700 000	fl1577	788 500
(Other Instances Used in the Experiments)			
kroC100	50 000	kroD100	50 000
Pcb442	2 000 000		

TABLE II
COMPARING S-CLPSO WITH THE OTHER EXISTING PSO-BASED APPROACHES IN THE SYMMETRIC TSP

	S-CLPSO			PSO-TS-2opt [14]	Discrete PSO [20]			C3PSO [38]	
	Best	Worst	Mean	Mean	Best	Worst	Mean	Best	Mean
eil51	**426**	**433**	**427.9**	440.9	427	452	436.9	427	433.64
Berlin52	**7542**	**7662**	**7548.4**	7704	**7542**	8362	7756	**7542**	7598.8
st70	**675**	**690**	**680.1**	N/A	**675**	742	697.5	N/A	N/A
eil76	**538**	**549**	**541.7**	560.7	546	579	560.4	540	551.7
pr76	**108 159**	**110 255**	**108 690**	N/A	108 280	124 365	112 288	N/A	N/A
kroA100	**21 282**	**21 591**	**21 368.2**	N/A	N/A	N/A	N/A	21 296	21 689.3
kroA200	**29 544**	**30 748**	**29 981.7**	N/A	N/A	N/A	N/A	29 563	30 374.3

Results of S-CLPSO are averaged over 50 runs.

We also compare the S-CLPSO algorithm with the ACS algorithm [28]. ACS is one of the best-performed ACO algorithms [39]. In the experiment, we follow the source code available at http://iridia.ulb.ac.be/∼mdorigo/ACO/aco-code/public-software.html and modify it for the VC++ 6.0 compiler environment. Both S-CLPSO and ACS are tested on the first 12 TSP instances in Table I on the same machine. The parameter settings in ACS are the same as [28], i.e., $\beta = 2$, $q_0 = 0.9$, $\alpha = \rho = 0.1$, and the number of ants is ten. In every single run, both algorithms generate the same number of solutions as given in Table I. Each instance is run for 50 times. The best results, worst results, average results, standard deviations, and the average time to complete a single run (in milliseconds) for each instance are listed in Table III. It can be seen that the S-CLPSO algorithm outperforms the ACS algorithm in all test instances. According to the standard deviations, the S-CLPSO is more stable. Moreover, the results of two-tailed tests reveal that except for the instance lin318, the results obtained by the S-CLPSO algorithm are significantly better than the ones obtained by the ACS algorithm. The execution time of the S-CLPSO algorithm is also shorter than ACS. The worst-case computational complexity of generating a solution in both S-CLPSO and ACS is $O(n^2)$. But in S-CLPSO, particles first tend to select the elements from $cut_\alpha(V_i^j)$, and then reuse the elements from X_i^j. Because $cut_\alpha(V_i^j)$ and X_i^j are smaller sets than the universal set E, the selection operator in S-CLPSO always applies to a much smaller set during the search process. Therefore the execution time is shortened. The convergence curves of both algorithms are plotted in Fig. 18. The curves are based on the mean results of 50 runs. It can be seen that S-CLPSO performs worse than ACS in the early stages. This is because the ACS algorithm employs some problem-dependent heuristic information and uses the aggressive pseudorandom proportional action choice rule [28], [39] to bias short arcs. However, in the later stages, while it is difficult for ACS to find better solutions, S-CLPSO is still diverse enough to avoid being trapped in local optima. Therefore, S-CLPSO manages to yield better results.

In Tables IV–VI, the S-CLPSO algorithm is compared with the Max–Min Ant System (MMAS) algorithm [40], the Lin–Kernighan algorithm [36], and some genetic algorithms (GAs) with different crossover operators such as edge assembly crossover (EAX) [41], heuristic crossover (HX) [42], and distance preserving crossover (DPX) [43]. The data of MMAS, the Lin–Kernighan algorithm, and the genetic algorithms with different crossover operators are extracted from [40], [44], and [45], respectively. In Table IV, the S-CLPSO algorithm is run for the same number of evaluations as the experiments in [40]. Better mean results are yielded by the S-CLPSO algorithm in all of the three instances compared with MMAS. In Table V, S-CLPSO is run ten times for each instance and each run is terminated when the

TABLE III
COMPARING S-CLPSO WITH ACS IN THE SYMMETRIC TSP. THE RESULTS ARE AVERAGED OVER 50 RUNS

Instance	Best Known	Algorithm	Best	Worst	Mean	Deviation	Time (ms)	t-test (ACS-S-CLPSO)
eil51	426	S-CLPSO	**426**	**433**	**427.3**	**1.23**	**1251**	6.220#
		ACS	**426**	438	430.3	2.93	2387	
Berlin52	7542	S-CLPSO	**7542**	**7618**	**7544.3**	**11.34**	**1200**	5.506#
		ACS	**7542**	7986	7657.4	144.84	2448	
st70	675	S-CLPSO	**675**	**687**	**677.7**	**3.04**	**2382**	7.558#
		ACS	**675**	716	685.8	6.94	5759	
eil76	538	S-CLPSO	**538**	**547**	**540.3**	**2.48**	**2699**	7.968#
		ACS	**538**	558	547.2	5.60	6633	
pr76	108 159	S-CLPSO	**108 159**	**110 213**	**108 447**	**475.80**	**2581**	9.247#
		ACS	**108 159**	113 096	110 197.7	1251.31	6453	
kroA100	21 282	S-CLPSO	**21 282**	**21 658**	**21 352.5**	**84.51**	**5047**	4.823#
		ACS	**21 282**	22 823	21 584	328.66	15 304	
eil101	629	S-CLPSO	**629**	**646**	**637.1**	**4.45**	**5563**	7.575#
		ACS	631	661	646.3	7.30	15 581	
lin105	14 379	S-CLPSO	**14 379**	**14 561**	**14 462.7**	**55.29**	**5393**	3.960#
		ACS	**14 379**	15 121	14 539.5	125.48	17 225	
kroA150	26 524	S-CLPSO	**26 537**	**27 317**	**26 892.1**	**165.36**	**14 389**	6.898#
		ACS	26 734	28 008	27 202.6	272.03	48 153	
kroA200	29 368	S-CLPSO	**29 399**	**30 139**	**29 722.4**	**153.58**	**37 072**	4.800#
		ACS	29 506	31 138	30 001.5	381.33	111 003	
pr299	48 191	S-CLPSO	**48 478**	**50 427**	**49 222.5**	**415.12**	**249 369**	5.119#
		ACS	48 828	50 936	49 721.6	550.41	338 787	
lin318	42 029	S-CLPSO	**42 719**	**44 209**	**43 518.4**	**335.92**	**314 635**	1.513
		ACS	43 050	44 716	43 624.4	364.02	383 877	

#Value of t with 49 degrees of freedom is significant at $\alpha = 0.05$ by a two-tailed test.

TABLE IV
COMPARING S-CLPSO WITH MMAS FOR THE SYMMETRIC TSP

Instance	Best Known	Max Evaluations	S-CLPSO	MMAS
eil51	426	510 000	**426.4**	427.2
kroA100	21 282	1 000 000	**21 342.5**	21 352.05
d198	15 780	1 980 000	**15 809.0**	16 065.95

Results of S-CLPSO are averaged over 30 runs. The results of MMAS are derived from [40].

TABLE V
COMPARING S-CLPSO WITH THE LIN–KERNIGHAN ALGORITHM FOR THE SYMMETRIC TSP

Instance	Best Known	Average Evaluations of S-CLPSO	S-CLPSO	Number of Runs that S-CLPSO Obtains the Optimal Solution	Lin–Kernighan
eil51	426	12 557	**427.2**	2	427.4
eil76	538	28 126	**540.3**	3	549.7
eil101	629	42 751	**637.1**	1	640
st70	675	22 004	**677.5**	5	684.6
kroA100	21 282	33 528	**21 352.5**	4	21 380.9
kroC100	20 749	25 219	**20 824.6**	2	20 961
kroD100	21 294	37 516	**21 405.6**	0	21 417.3
lin105	14 379	31 006	**14 462.7**	3	14 566.5
pcb442	50 778	1 120 741	**51 577.7**	0	51 776.5

Results of S-CLPSO are averaged over 10 runs. The results of Lin–Kernighan are extracted from [44].

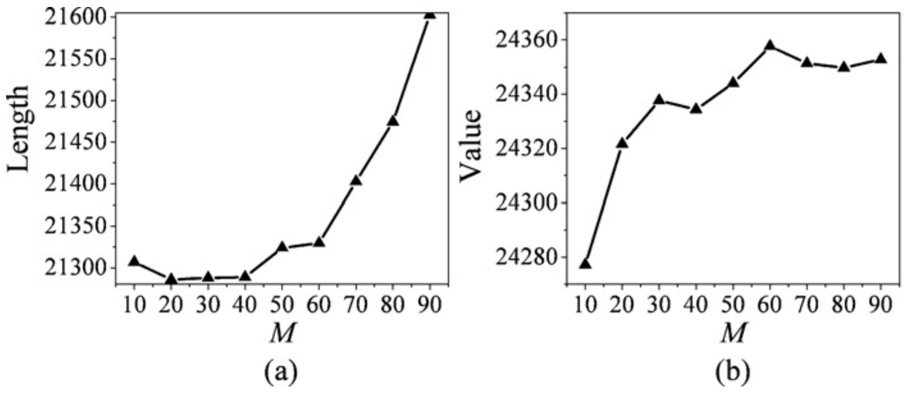


Fig. 17. Performance of the algorithm with different swarm sizes. The results are averaged over 50 runs. (a) TSP instance: kroA 100, (b) MKP instance: 5.100.00.

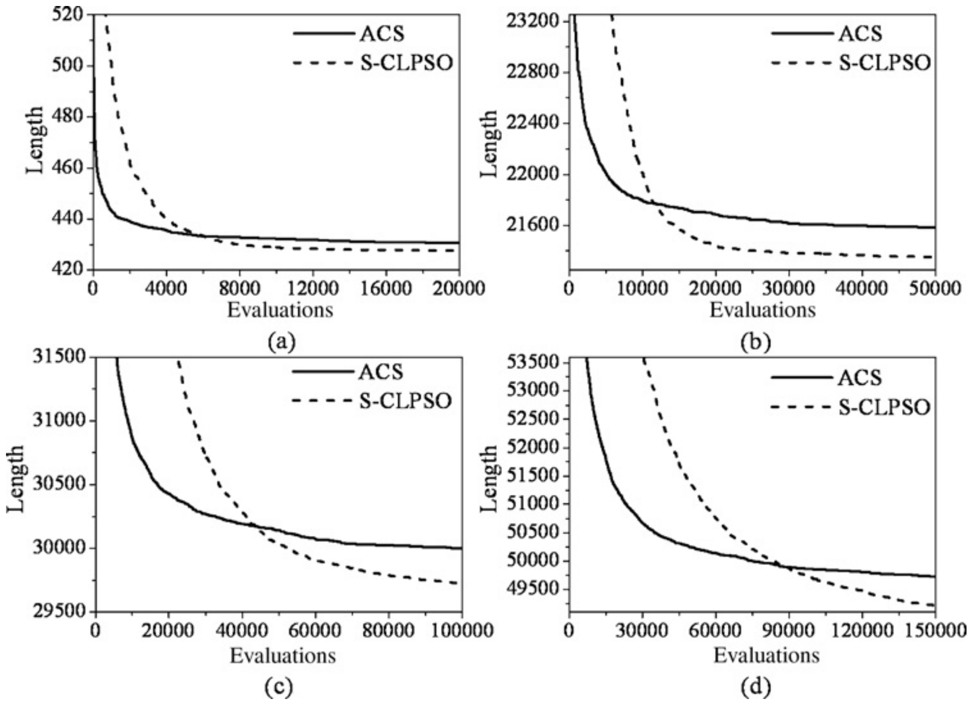


Fig. 18. Comparison between the convergent speed of S-CLPSO and ACS. (a) ei151, (b) korA100, (c) kor A200, and (d) pr299.

TABLE VI

COMPARING S-CLPSO WITH THE GAS WITH DIFFERENT EDGE-BASED CROSSOVER OPERATORS FOR THE SYMMETRIC TSP

Instance	Evaluations	S-CLPSO	EAX	HX	DPX
att48	480 000	0.0897	**0.0000**	2.3758	4.0581
eil76	760 000	0.0126	**0.0000**	2.5465	2.4535
eil101	1 010 000	0.1643	**0.0000**	3.8474	8.6963
kroA150	1 500 000	0.4728	**0.0000**	6.0153	5.8355
kroA200	2 000 000	0.1377	**0.0051**	7.1700	3.7602

Results of S-CLPSO are averaged over 30 runs. The results of the GA approaches are derived from [45]. In the table, the results are given by the average error defined as $(average/best\text{-}known - 1) \times 100$ where *average* is the average values obtained by the algorithm and *best-known* is the best known result of the instance.

TABLE VII
COMPARING THE S-CLPSO ALGORITHM WITH CANDIDATE LIST, WITH THE ACS ALGORITHM WITH CANDIDATE LIST, FOR LARGE-SCALE SYMMETRIC TSP INSTANCES

Instance	Best Known	Algorithm	Mean	Deviation	Time (ms)
d493	35 002	S-CLPSO	**37 028.83**	**323.53**	**78 751**
		ACS	37 120.97	785.79	156 588
d657	48 912	S-CLPSO	**51 799.53**	**538.94**	**167 444**
		ACS	52 098.87	1429.26	299 591
u724	41 910	S-CLPSO	**43 373.3**	**185.35**	**188 186**
		ACS	43 405.6	266.86	342 648
d1291	50 801	S-CLPSO	**52 353.1**	**432.69**	**902 063**
		ACS	53 122.3	861.56	1 458 964
fl1400	20 127	S-CLPSO	22 025.2	**185.94**	**1 419 804**
		ACS	**21 727.3**	506.53	2 201 702
fl1577	22 249	S-CLPSO	**23 038.4**	**129.08**	**1 562 957**
		ACS	23 107.5	357.45	2 262 754

Results are averaged over 30 runs.

TABLE VIII
PERFORMANCE OF THE S-CLPSO ALGORITHM WITH THE 3-OPT LOCAL SEARCH OPERATOR

Instance	Best Known	gbest		Iteration Best			All Positions	
		Best (Error)	Mean (Error)	Best (Error)	Mean (Error)	Time (ms)	Best (Error)	Mean (Error)
d198	15 780	15 800 (0.13%)	16 014 (1.48%)	15 780 (0.00%)	15 780 (0.00%)	9141	15 780 (0.00%)	15 780 (0.00%)
lin318	42 029	42 349 (0.76%)	42 861 (1.98%)	42 029 (0.00%)	42 130 (0.24%)	24 558	42 029 (0.00%)	42 029 (0.00%)
d493	35 002	35 367 (1.04%)	36 087 (3.10%)	35 002 (0.00%)	35 075 (0.21%)	63 823	35 002 (0.00%)	35 003 (0.00%)
d657	48 912	50 139 (2.51%)	50 729 (3.71%)	48 913 (0.00%)	48 956 (0.09%)	117 928	48 913 (0.00%)	48 919 (0.01%)
u724	41 910	42 619 (1.69%)	42 972 (2.53%)	41 910 (0.00%)	41 950 (0.10%)	133 549	41 910 (0.00%)	41 916 (0.01%)
rat783	8806	9034 (2.59%)	9095 (3.28%)	8808 (0.02%)	8826 (0.23%)	155 404	8806 (0.00%)	8806 (0.00%)
d1291	50 801	51 183 (0.75%)	51 756 (1.88%)	50 801 (0.00%)	50 859 (0.11%)	501 289	50 801 (0.00%)	50 807 (0.01%)
fl1400	20 127	20 349 (1.10%)	20 937 (4.02%)	20 127 (0.00%)	20 138 (0.05%)	757 030	20 127 (0.00%)	20 132 (0.02%)
fl1577	22 249	22 455 (0.93%)	22 821 (2.57%)	22 254 (0.02%)	22 255 (0.03)	693 490	22 254 (0.02%)	22 254 (0.02%)

"Gbest" is the version that 3-opt is performed on the Gbest position, "iteration best" is the version that 3-opt is performed on the iteration best position, and "all positions" is the version that 3-opt is performed on all the positions found by particles. 30 independent runs are executed for each instance.

maximum number of evaluations given in Table I has been reached. Compared with the average results of ten runs of the Lin–Kernighan algorithm reported in [44], S-CLPSO is able to achieve better averages in all instances. Moreover, while the Lin–Kernighan algorithm fails to obtain optimal results in all ten runs for all instances [44], S-CLPSO manages to find the optimal results of all instances in at least one out of ten runs except for kroD100 and pcb442. On the other hand, as an effective heuristic procedure originally designed for TSP, the Lin–Kernighan consumes much less execution time than S-CLPSO. If both algorithms are run for the same execution time (as opposed to the same number of runs), the Lin–Kernighan algorithm is likely to win the competition easily. According to Table VI, S-CLPSO is not as good as the GA approach with the EAX operator, which is one of the most powerful GA operators for the TSP. However, compared with the GAs with other crossover operators, S-CLPSO is able to achieve significantly better average results, and the errors of S-CLPSO in all instances are less than 0.5%. These results also reveal that S-CLPSO is promising.

To test the performance of the algorithm on bigger TSP instances, we introduce the candidate list strategy to accelerate the S-CLPSO algorithm. The candidate list strategy is a simple domain-based technique for TSP [28], [46]. A candidate list is a static data structure. For each node j, the candidate list records the cl shortest arcs connected with j, ordered by increasing lengths. We set cl = 15 according to the ACS algorithm with candidate list [28]. The S-CLPSO algorithm with candidate list works as follows. In the position updating procedure (Fig. 7), suppose a particle i located on dimension (node) j is trying to select an arc to add to the new position; it first considers the arcs in the candidate list. In other

TABLE IX
COMPARING S-CLPSO WITH THE BINARY PSO ALGORITHM PROPOSED IN [47] FOR THE MKP

Instance	Optimum	PSO-P		S-CLPSO-V1		PSO-R		S-CLPSO-V2	
		Best	Mean	Best	Mean	Best	Mean	Best	Mean
5.100.00	24 381	22 525	22 013	24 312	24 187	**24 381**	**24 356**	**24 381**	**24 356**
5.100.01	24 274	22 244	21 719	**24 274**	24 180	24 258	24 036	**24 274**	**24 213**
5.100.02	23 551	21 822	21 050	23 538	23 492	**23 551**	23 523	**23 551**	**23 530**
5.100.03	23 534	22 057	21 413	23 396	23 283	**23 527**	**23 481**	**23 527**	23 478
5.100.04	23 991	22 167	21 677	23 959	23 892	23 966	**23 966**	**23 991**	23 963
10.100.00	23 064	20 895	20 458	23 055	22 917	**23 057**	23 050	**23 057**	**23 051**
10.100.01	22 801	20 663	20 089	22 706	22 472	22 781	22 668	**22 801**	**22 725**
10.100.02	22 131	20 058	18 582	22 065	21 893	**22 131**	22 029	**22 131**	**22 073**
10.100.03	22 772	20 908	20 446	22 604	22 418	**22 772**	22 733	**22 772**	**22 741**
10.100.04	22 751	20 488	20 025	22 624	22 467	**22 751**	**22 632**	22 697	22 605

Results of S-CLPSO are averaged over 30 runs.

words, the particle preferentially exploits the arcs belonging to both the candidate list and $cut_\alpha(V_i^j)$, and then it prefers the ones belonging to both the candidate list and the previous X_i^j, and finally it considers the other arcs in the candidate list. Only if none of the arcs in the candidate list can be chosen, the particle considers the rest of the arcs following the procedure in Fig. 7. In this case, the particle always chooses the arcs from the small-size candidate list, resulting in significant reduction of computational time. The use of candidate list leads to a very aggressive search behavior of the algorithm. In order to balance the exploration and exploitation behavior of the particles, we empirically set Pc to a random number belonging to [0.005, 0.01]. The configuration for the refreshing gap rg remains rg = 7. The S-CLPSO algorithm with candidate list is compared with the ACS algorithm with candidate list [28] on six bigger TSP instances with 493 to 1577 cities. The results averaged over 30 trials are given in Table VII. The results reveal that the proposed S-CLPSO algorithm with candidate list is also competitive on bigger TSP instances, with slightly shorter mean tours on five out of six instances and shorter execution time in all cases.

The S-CLPSO algorithm with candidate list is further hybridized with the 3-opt local search operator, and the results are shown in Table VIII. The 3-opt operator is a well-known local search operator for tour improvement [28], [51]. In the experiment, we follow the source code available at http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html to implement the 3-opt operator. All parameter settings are the same as the ones used in the S-CLPSO with candidate list, except that the size of the candidate list is set to cl = 20. Three versions of the hybrid S-CLPSO algorithm are considered: 1) in the "gbest" version, local search is performed only on the *gbest* position in each iteration; 2) in the "iteration best" version, local search is performed on the best position found in each iteration; and 3) in the "all positions" version, local search is performed on all the positions found by particles. The algorithm is run for $50n$ iterations, where n is the size of the instance. According to the results in Table VIII, performing a local search on the iteration best position obtains better results than performing local search on the *gbest* position. The "iteration best" version achieves optimum solutions in six out of the nine instances and the average errors are at most 0.24%. With the local search operator performing on more positions, the "all positions" version yields optimum solutions in seven out of the nine instances and the average errors are at most 0.02%. Overall, the performance of both the "iteration best" version and the "all positions" version of the S-CLPSO algorithm hybridized with 3-opt is promising.

B. Performance on the MKP

In order to study the performance of S-CLPSO on the MKP, we first compare the algorithm with the PSO algorithm implemented by M. Kong and P. Tian [47]. The algorithm in [47] is based on Kennedy and Eberhart's binary PSO [7]. It uses the ring topology as the neighborhood structure with the number of neighborhoods set to 2. Two versions of the PSO algorithm were proposed in [47], including PSO-P, which uses the penalty function technique to deal with the resource constraints, and PSO-R, which applies the repair procedure proposed in [32] to convert infeasible solutions to feasible ones. Similarly, we also design two versions of the S-CLPSO algorithm for MKP. The first version only selects feasible elements in the position updating procedure (Fig. 7) so that the feasibility of the solutions can be always guaranteed. The second version also uses the repair procedure proposed in [32], which works by dropping and adding items according to the surrogate constraint coefficient of each item. The surrogate constraint coefficient is calculated by transforming multiple constraints into a single constraint using surrogate weights. The weights are derived by solving the linear programming relaxation of the original MKP. (In implantation, we use MATLAB as the linear programming solver and the MATLAB program is linked with the C program of the algorithm in the execution. Note that the surrogate weights only need to be computed once and remain unchanged during the whole process of the algorithm.) We denote these two versions as S-CLPSO-V1 and S-CLPSO-V2 in Table IX. All of these four algorithm versions are tested on the first five instances of the instance sets "mknapcb1" and "mknapcb4" from the ORLIB. The results of PSO-P and PSO-R are extracted from [47]. S-CLPSO-V1 and S-CLPSO-V2 are run 30 times for each instance and the number of

TABLE X
COMPARING S-CLPSO WITH CLERC'S SIMPLIFIED BPSO ALGORITHM FOR THE MKP

Instance	Optimum	S-CLPSO		S _BPSO(2)		S-BPSO(3)		S-BPSO(03)		S-BPSO(23)	
		Mean	Time (ms)	Mean	Time (ms)	Mean	Time (ms)	Mean	Time (ms)	Mean	Time (ms)
5.100.00	24 381	**24 372**	**21 166**	24 362	46 450	24 329	33 086	24 336	33 206	24 365	30 556
5.100.01	24 274	**24 265**	**22 141**	24 135	48 239	24 211	34 982	24 178	34 891	24 194	32 992
5.100.02	23 551	**23 532**	**20 998**	23 525	47 982	23 515	33 723	23 521	33 447	23 523	32 733
5.100.03	23 534	23 505	**22 147**	**23 509**	46 380	23 445	34 992	23 486	33 005	23 500	32 282
5.100.04	23 991	**23 964**	**20 989**	23 946	46 574	23 939	33 119	23 933	33 477	23 941	32 721
5.100.05	24 613	**24 603**	**21 282**	24 588	46 987	24 588	33 295	24 580	32 521	24 588	31 980
5.100.06	25 591	**25 591**	**20 441**	25 584	46 817	25 546	32 582	25 537	33 248	25 538	32 120
5.100.07	23 410	**23 410**	**20 976**	**23 410**	47 123	23 391	33 555	23 397	33 518	23 398	32 091
5.100.08	24 216	24 182	**21 250**	**24 205**	46 864	24 151	33 265	24 184	33 120	24 196	32 591
5.100.09	24 411	**24 410**	**22 328**	24 360	47 428	24 352	33 762	24 339	33 319	24 341	33 271
10.100.00	23 064	**23 052**	**24 944**	**23 052**	54 170	23 042	44 880	23 040	44 833	23 041	37 284
10.100.01	22 801	**22 762**	**26 356**	22 687	54 735	22 688	44 913	22 662	44 872	22 651	37 819
10.100.02	22 131	**22 107**	**25 835**	22 034	54 367	22 020	44 872	22 010	43 219	22 022	38 188
10.100.03	22 772	**22 758**	**26 630**	22 750	54 128	22 745	45 331	22 737	44 102	22 723	37 956
10.100.04	22 751	22 611	**24 933**	**22 633**	53 741	22 573	45 355	**22 633**	44 912	22 613	37 134
10.100.05	22 777	**22 698**	**26 974**	22 682	54 661	22 630	45 027	22 641	44 627	22 664	37 858
10.100.06	21 875	**21 824**	**26 069**	21 790	54 127	21 740	45 839	21 774	44 552	21 783	37 894
10.100.07	22 635	**22 568**	**25 803**	22 511	54 441	22 499	45 022	22 454	43 987	22 481	39 003
10.100.08	22 511	**22 414**	**25 552**	22 400	54 923	22 374	45 129	22 329	44 471	22 372	37 436
10.100.09	22 702	**22 681**	**24 718**	22 643	54 512	22 614	45 520	22 642	44 098	22 640	38 234

Results are averaged over 30 runs.

TABLE XI
COMPARING S-CLPSO WITH THE PUBLISHED RESULTS OF THE ANT ALGORITHMS IN [48] AND [49] AND THE EVOLUTIONARY ALGORITHM IN [50]

Instance	Optimum	S-CLPSO			Ant System [48]		Ant Algorithm [49]		EA [50]
		Best	Mean	Avg Iter	Best	Mean	Best	Mean	Best
5.100.00	24 381	**24 381**	**24 356**	493	**24 381**	24 331	**24 381**	24 342	23 984
5.100.01	24 274	**24 274**	24 213	1067	**24 274**	24 245	**24 274**	**24 247**	24 145
5.100.02	23 551	**23 551**	**23 530**	348	**23 551**	23 527	**23 551**	23 529	23 523
5.100.03	23 534	23 527	**23 478**	912	23 527	23 463	**23 534**	23 462	22 874
5.100.04	23 991	**23 991**	**23 963**	445	**23 991**	23 949	**23 991**	23 946	23 751
5.100.05	24 613	**24 613**	**24 589**	435	**24 613**	24 563	**24 613**	24 587	24 601
5.100.06	25 591	**25 591**	**25 591**	71	**25 591**	25 504	**25 591**	25 512	25 293
5.100.07	23 410	**23 410**	**23 403**	648	**23 410**	23 361	**23 410**	23 371	23 204
5.100.08	24 216	24 195	24 170	772	24 204	**24 173**	**24 216**	24 172	23 762
5.100.09	24 411	**24 411**	**24 375**	930	**24 411**	24 326	**24 411**	24 356	24 255
10.100.00	23 064	23 057	**23 051**	230	23 057	22 996	**23 064**	23 016	N/A
10.100.01	22 801	**22 801**	**22 725**	995	**22 801**	22 672	**22 801**	22 714	N/A
10.100.02	22 131	**22 131**	**22 073**	769	**22 131**	21 980	**22 131**	22 034	N/A
10.100.03	22 772	**22 772**	**22 741**	649	**22 772**	22 631	22 717	22 634	N/A
10.100.04	22 751	**22 697**	**22 605**	360	22 654	22 578	22 654	22 547	N/A
10.100.05	22 777	22 703	**22 643**	813	22 652	22 565	**22 716**	22 602	N/A
10.100.06	21 875	21 821	**21 815**	643	**21 875**	21 758	**21 875**	21 777	N/A
10.100.07	22 635	**22 635**	**22 573**	720	22 551	22 519	22 551	22 453	N/A
10.100.08	22 511	22 422	**22 379**	903	22 418	22 292	**22 511**	22 351	N/A
10.100.09	22 702	**22 702**	**22 687**	619	**22 702**	22 588	**22 702**	22 591	N/A

Results of S-CLPSO are averaged over 30 runs.

evaluations in each run is the same as that of PSO-P and PSO-R. According to the results in Table IX, for the algorithm versions without repair procedures, it is apparent that S-CLPSO-V1 outperforms PSO-P in all cases. The position updating rule used in S-CLPSO-V1 to only build feasible solutions provides a better way than the penalty function technique to deal with the constraints when no repair procedures are used. For the versions with repair procedures, S-CLPSO-V2 yields better average results on six out of the ten instances, while PSO-R obtains better results on three instances. In this sense, S-CLPSO-V2 performs slightly better than PSO-R.

In Table X, the CLPSO algorithm is further compared with Clerc's simplified BPSO (S_BPSO) algorithm with various strategies (http://clerc.maurice.free.fr/pso/). In S_BPSO, four strategies are provided. Strategy zero works by converting the binary coding into integer coding and following the updating rule in the Standard PSO 2007. Strategy 1 builds two positions respectively around the *PBest* and *LBest* positions, and merges them by the majority rule. Strategy two is the updating rule in Kennedy and Eberhart's BPSO, but instead of using the global topology, strategy 2 in the simplified BPSO uses a random topology. Strategy 3 simply builds a random position around the *LBest* position. These strategies can be combined together. We denote the algorithm with strategy zero as S_BPSO(0), the algorithm with strategies zero and 1 as S_BPSO(01), and so on. The source code of S_BPSO is available in http://clerc.maurice.free.fr/pso/binary_pso/simpleBinaryPSO_C.zip. We modify the code to solve MKP in the experiment. All the parameter configurations in the source code remain unchanged in the experiment expect for the swarm size M. We set $M = n$ in both S_BPSO and S_CLPSO. (For the strategy zero of S_BPSO, the additional parameter *numSize* is set to $n/25$, where n is the size of the MKP instance.) Four relatively good (combinations of) strategies for the MKP are selected in the experiment, i.e., S_BPSO(03), S_BPSO(02), S_BPSO(3), and S_BPSO(23). (Due to the characteristics of the MKP, strategies zero and 1 seem to be not suitable and the performance is not satisfying.) In the experiment, the repair procedure proposed in [32] is applied to all the algorithms to convert the infeasible solutions to feasible ones. In each run, all algorithms generate 1 000 000 solutions except for S_BPSO(3). As the strategy three is very simple, we allow S_BPSO(3) to generate 2 000 000 solutions in each run. Each instance is run 30 times. According to Table X, from the point of view of the solution quality, it can be seen that S-CLPSO manages to obtain the best results among the five algorithms in 17 out of 20 instances. From the point of view of the execution time, compared with the other algorithms, S-CLPSO consumes less time. This is because strategy zero of S_BPSO needs to convert binary numbers into integers and strategy 2 needs to evaluate the sigmoid function with an "exp" operator iteratively.

Finally in Table XI, the S-CLPSO algorithm with the repair procedure proposed in [32] is compared with Leguizamon and Michalewicz's ant system algorithm [48], the ant algorithm proposed by Alaya *et al*. [49], and Findanova's evolutionary algorithm (EA) [50]. In the experiment, the maximum number of iterations run by the S-CLPSO algorithm is 2000. The average numbers of iteration for S-CLPSO to achieve the best solution are also reported in Table XI. According to the best results, both S-CLPSO and the ant system algorithm [48] find the optimal solutions in 13 out of 20 instances, while the ant algorithm [49] is able to find the optimal solutions in 16 instances. The results of these three algorithms are all significantly better than those obtained by the evolutionary algorithm [50]. According to the mean results, S-CLPSO obtains the best average results in 18 out of 20 instances. These results reveal that S-CLPSO has better consistency to obtain relatively good solutions in different runs. On the other hand, according to the average iteration numbers, as the S-CLPSO algorithm here only uses the random selection operator (instead of the heuristic-based selection operator), the convergence process of S-CLPSO is slower than the two ACO algorithms.

VI. Conclusion

A set-based particle swarm optimization (S-PSO) method has been proposed. In order to solve discrete space problems, S-PSO defines the position and velocity using the concept of set and possibility. Based on the proposed S-PSO, different PSO variants, such as the global version of PSO, the local versions of PSO with different topologies, and the CLPSO can be extended to their discrete versions. Since the CLPSO algorithm [6] is proposed to solve complex multimodal problems and COPs are always complex problems, we have found that the discrete version of CLPSO based on S-PSO (S-CLPSO) performs the best. The most important parameters in the original PSO, i.e., the inertia weight and the acceleration coefficients, play a similar role in S-PSO, and the classical configurations are still suitable in S-PSO. We have implemented experiments to compare the S-CLPSO algorithm with both existing PSO-based approaches and some meta-heuristic algorithms. Experimental results have shown that the proposed algorithm is promising.

References

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[2] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1998, pp. 69–73.

[3] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," in *Proc. IEEE Int. Conf. Evol. Comput.*, vol. 1. 2001, pp. 101–106.

[4] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 58–73, Feb. 2002.

[5] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 225–239, Jun. 2004.

[6] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, Jun. 2006.

[7] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, 1997, pp. 4104–4109.

[8] B. Al-Kazemi and C. K. Mohan, "Discrete multi-phase particle swarm optimization," in *Information Processing with Evolutionary Algorithms*. Berlin, Germany: Springer, 2006, pp. 306–326.

[9] G. Pampara, N. Franken, and A. P. Engelbrecht, "Combining particle swarm optimisation with angle modulation to solve binary problems," in *Proc. 2005 IEEE Congr. Evol. Comput.*, vol. 1. pp. 89–96.
[10] M. Clerc, "Discrete particle swarm optimization," in *New Optimization Techniques in Engineering*. New York: Springer-Verlag, 2004.
[11] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proc. 2nd Int. Conf. Mach. Learning Cybern.*, 2003, pp. 1583–1585.
[12] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors Microsyst.*, vol. 26, no. 8, pp. 363–371, Aug. 2002.
[13] D. Y. Sha and C. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Comput. Ind. Eng.*, vol. 51, no. 4, pp. 791–808, Dec. 2006.
[14] W. Pang, K.-P. Wang, C.-G. Zhou, L.-J. Dong, M. Liu, H.-Y. Zhang, and J.-Y. Wang, "Modified particle swarm optimization based on space and transformation for solving traveling salesman problem," in *Proc. 3rd Int. Conf. Mach. Learning Cybern.*, 2004, pp. 2342–2348.
[15] W. Pang, K.-P. Wang, C.-G. Zhou, and L.-J. Dong, "Fuzzy discrete particle swarm optimization for solving traveling salesman problem," in *Proc. 4th Int. Conf. Comput. Information Technol. (CIT)*, 2004, pp. 796–800.
[16] C.-J. Liao, C.-T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization flowshop scheduling problems," *Comput. Operations Res.*, vol. 34, no. 10, pp. 3099–3111, Oct. 2007.
[17] B. Shen, M. Yao, and W. Yi, "Heuristic information based improved fuzzy discrete PSO method for solving TSP," in *Proc. 9th Pacific Rim Int. Conf. Artif. Intell. (PRICAI)*, 2006, pp. 859–863.
[18] Y. Wang, Y. Wanga, X.-Y. Fenga, Y.-X. Huanga, D.-B. Pub, W.-G. Zhoua, Y.-C. Lianga, and C.-G. Zhoua, "A novel quantum swarm evolutionary algorithm and its applications," *Neurocomputing*, vol. 70, nos. 4–6, pp. 633–640, Jan. 2007.
[19] F. Afshinmanesh, A. Marandi, and A. Rahimi-Kian, "A novel binary particle swarm optimization method using artificial immune system," in *Proc. IEEE Int. Conf. Comput. Tool (EUROCON)*, 2005, pp. 217–220.
[20] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. X. Wang, "Particle swarm optimization-based algorithms for TSP and generalized TSP," *Inform. Process. Lett.*, vol. 103, no. 5, pp. 169–176, Aug. 2007.
[21] Y.-X. Jin, H.-Z. Cheng, J.-Y. Yan, and L. Zhang, "New discrete method for particle swarm optimization and its application in transmission network expansion planning," *Electr. Power Syst. Res.*, vol. 77, nos. 3–4, pp. 227–233, Mar. 2007.
[22] S. Chandrasekaran, S. G. Ponnambalam, R. K. Suresh, and N. Vijayakumar, "A hybrid discrete particle swarm optimization algorithm to solve flow shop scheduling problems," in *Proc. Int. Conf. Cybern. Intell. Syst.*, 2006, pp. 1–6.
[23] Q.-K. Pan, M. F. Tasgentiren, and Y.-C. Liang, "A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem," *Comput. Operations Res.*, vol. 35, no. 9, pp. 2807–2839, Sep. 2008.
[24] C.-T. Tseng and C.-J. Liao, "A discrete particle swarm optimization for lot-streaming flowshop scheduling problem," *Eur. J. Oper. Res.*, vol. 191, no. 2, pp. 360–373, Dec. 2008.
[25] Z. Lian, X. Gu, and B. Jiao, "A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan," *Chaos Solitons Fractals*, vol. 35, no. 5, pp. 851–861, Mar. 2008.
[26] H. S. Lopes and L. S. Coelho, "Particle swarm optimization with fast local search for the blind traveling salesman problem," in *Proc. 5th Int. Conf. Hybrid Intell. Syst. (HIS)*, Nov. 2005, pp. 245–250.
[27] E. F. G. Goldbarg, G. R. de Souza, and M. C. Goldbarg, "Particle swarm for the traveling salesman problem," in *Proc. Conf. Evol. Comput. Combinatorial Optimisation*, 2006, pp. 99–110.
[28] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.
[29] R. C. Eberhart and Y. Shi, "Particle swarm optimization: Developments, applications and resources," in *Proc. IEEE Int. Congr. Evol. Comput.*, vol. 1. 2001, pp. 81–86.
[30] J. Kennedy, "Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance," in *Proc. 1999 Congr. Evol. Comput. (CEC)*, vol. 3. Washington D.C., 1999, pp. 1931–1938.
[31] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proc. 2002 Congr. Evol. Comput. (CEC)*, vol. 2. 2002, pp. 1671–1676.
[32] P. C. Chu and J. E. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *J. Heuristics*, vol. 4, no. 1, pp. 63–86, Jun. 1998.
[33] T. M. Blackwell and P. J. Bentley, "Do not push me! Collision-avoiding swarms," in *Proc. Congr. Evol. Comput. (CEC)*, 2002, pp. 1691–1696.
[34] A. Ratnaweera, S. K. Halgamuge, and H.C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 240–255, Jun. 2004.
[35] M. Lovbjerg, T. K. Rasmussen, and T. Krink, "Hybrid particle swarm optimizer with breeding and subpopulations," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 469–476.
[36] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Res.*, vol. 21, no. 2, pp. 498–516, Mar.–Apr. 1973.
[37] G. Reinelt, "TSPLIB: A traveling salesman problem library," *Orbit Reconst. Simulation Anal. J. Comput.*, vol. 3, no. 4, pp. 376–384, Jan. 1991.
[38] W.-L. Zhong, J. Zhang, and W.-N. Chen, "A novel discrete particle swarm optimization to solve traveling salesman problem," in *Proc. IEEE Int. Conf. Evol. Comput. (CEC)*, Singapore, Sep. 2007, pp. 3283–3287.
[39] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.
[40] T. Stützle and H. Hoos, "MAX-MIN ant system and local search for the traveling salesman problem," in *Proc. IEEE Int. Conf. Evol. Comput. (ICEC)*, 1997, pp. 309–314.
[41] Y. Nagata and S. Kobayashi, "Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem," in *Proc. 7th Int. Conf. Genetic Algorithm (ICGA)*, 1997, pp. 450–457.
[42] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," in *Genetic Algorithms Simulated Annealing*, L. Davis, Ed. London, U.K.: Pittman, 1987, ch. 4, pp. 42–60.
[43] B. Freisleben and P. Merz, "New genetic local search operators for the traveling salesman problem," in *Proc. 4th Parallel Problem Solving Nature (PPSN 4)*, 1996, pp. 890–899.
[44] G. Tao and Z. Michalewicz, "Inver-over operator for the TSP," in *Proc. Parallel Problem Solving Nature V (PPSN V)*, 1998, pp. 803–812.
[45] H.-K. Tsai, J.-M. Yang, Y.-F. Tsai, and C.-Y. Kao, "Some issues of designing genetic algorithms for traveling salesman problems," *Soft Comput.*, vol. 8, no. 10, pp. 689–697, Nov. 2004.
[46] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications*. New York: Springer-Verlag, 1994.
[47] M. Kong and P. Tian, "Apply the particle swarm optimization to the multidimensional knapsack problem," in *Proc. 8th Int. Conf. Artif. Intell. Soft Comput. (ICAISC)*, LNAI 4029. 2006, pp. 1140–1149.
[48] G. Leguizamon and Z. Michalewicz, "A new version of ant system for subset problems," in *Proc. Congr. Evol. Comput.*, vol. 2. 1999, pp. 1459–1464.
[49] I. Alaya, C. Solnon, and K. Ghéira, "Ant algorithm for the multidimensional knapsack problem," in *Proc. Int. Conf. Bio-Inspired Optimization Methods Their Appl. (BIOMA)*, Oct. 2004, pp. 63–72.
[50] S. Findanova, "Evolutionary algorithm for multidimensional knapsack problem," in *Proc. 7th Int. Conf. Parallel Problem Solving Nature Workshop*, 2002.
[51] S. Lin., "Computer solutions of the traveling salesman problem," *Bell Syst. J.*, vol. 44, pp. 2245–2269, Dec. 1965.

Wei-Neng Chen (S'07) received the Bachelor's degree in network engineering, in 2006, from Sun Yat-sen University, Guangzhou, China, where he is currently working toward the Ph.D degree in computer application techniques from the Department of Computer Science.

His current research interests include ant colony optimization, genetic algorithms, and other computational intelligence techniques, as well as the applications of project management and financial management.

Jun Zhang (M'02–SM'08) received the Ph.D degree in electronic engineering from the City University of Hong Kong, Kowloon Tong, Hong Kong, in 2002.

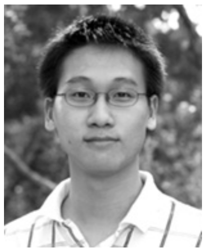
From 2003 to 2004, he was a Brain Korean 21 Postdoctoral Fellow at the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Yuseong-Gu, Korea. Since 2004, he has been with Sun Yat-sen University, Guangzhou, China, where he is currently a Professor in the Department of Computer Science. His research interests include computational intelligence, data mining, wireless sensor networks, operations research, and power electronic circuits. He is the author of four research book chapters, and over 80 technical papers in his research areas.

He is currently a Chair of the IEEE Beijing (Guangzhou) Section of the Computational Intelligence Society Chapter. He is currently an Associate Editor of the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.

Henry S. H. Chung (M'95–SM'03) received the B.E. and Ph.D degrees in electrical engineering from Hong Kong Polytechnic University, Hung Hom, Hong Kong, in 1991 and 1994, respectively.

Since 1995, he has been with the City University of Hong Kong, Kowloon Tong, Hong Kong. He is currently a Professor with the Department of Electronic Engineering, and the Chief Technical Officer of e.Energy Technology Ltd., Kowloon, Hong Kong—an associated company of the City University of Hong Kong. His research interests include time and frequency-domain analysis of power electronic circuits, switched-capacitor-based converters, random-switching techniques, control methods, digital audio amplifiers, soft-switching converters, and electronic ballast design. He holds ten patents, and he is the author of four research book chapters and over 250 technical papers, including 110 refereed journal papers in his research areas.

Dr. Chung was an IEEE Student Branch Counselor and a Track Chair of the Technical Committee on Power Systems and Power Electronic Circuits of IEEE Circuits and Systems Society during 1997–1998. He was an Associate Editor and a Guest Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, PART I: FUNDAMENTAL THEORY AND APPLICATIONS during 1999–2003. He is currently an Associate Editor of the IEEE TRANSACTIONS ON POWER ELECTRONICS and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS, PART I: FUNDAMENTAL THEORY AND APPLICATIONS. He was awarded the Grand Applied Research Excellence Award in 2001 from the City University of Hong Kong.



Wen-Liang Zhong received the Bachelor's degree in computer science and technology and the Master's degree in computer applicational technology from Sun Yat-sen University, Guangzhou, China, in 2007 and 2009, respectively. Currently, he is working toward the Ph.D. degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong.

His current research interests include machine learning and computational intelligence techniques and their applications.

Wei-Gang Wu (M'07) received the B.S. degree in material science and the M.S. degree in computer science, both from Xi'an Jiaotong University, Xi'an, China, in 1998 and 2003, respectively, and the Ph.D. degree in computer science from Hong Kong Polytechnic University, Hung Hom, Hong Kong, in 2007.

He has been a Postdoctoral Fellow with Hong Kong Polytechnic University from 2007 to 2009. He joined the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, as a lecturer in 2008. His research interests include intelligent computation, distributed algorithms, fault tolerance, mobile computing, and wireless networks. His recent research has focused on cooperative caching in mobile environments, and mobility management in wireless mesh networks. He is the author of more than 20 papers in conferences and journals, and he has served as a Transaction Processing Performance Council member for several conferences.

Yu-hui Shi (SM'98) received the Ph.D degree in electronic engineering from South-East University, Nanjing, China, in 1992.

He is a Professor in the Department of Electrical and Electronic Engineering, Xi'an Jiaotong-Liverpool University, Suzhou, China. He also holds the position of the Director of the Research and Postgraduate Office in Xi'an Jiaotong-Liverpool University. Before joining Xi'an Jiaotong-Liverpool University, he was with Electronic Data Systems Corporation, Indianapolis, IN. His main research interests include the areas of computational intelligence techniques (including swarm intelligence) and their applications.

Dr. Shi is the Editor-in-Chief of the *International Journal of Swarm Intelligence Research*, and an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION. He is the Chair of the IEEE Chief Information Officer Task Force on Swarm Intelligence.