Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

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

Wei-Neng Chen, Jun Zhang and Wen-Liang Zhong
Department of Computer Science
Sun Yat-sen University
Guangzhou, China P.R. 510275

***Abstract*—Particle swarm optimization (PSO) is predominately used to find solutions in continuous space. To explore the utility of PSO in the discrete space, this paper proposes a novel set-based PSO (S-PSO) method for combinatorial optimization problems (COPs). The proposed algorithm describes the discrete solution domains of COPs based on the concept of sets. A possible solution to the problem is mapped to a crisp set, and the velocity in S-PSO is defined as a set with possibilities. All arithmetic operators in the velocity and position updating rules in the original PSO are replaced by the operators defined on crisp sets and sets with possibilities in the proposed S-PSO. Based on S-PSO, 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 discrete versions. These discrete PSO variants based on S-PSO are tested on two famous COPs: the traveling salesman problem (TSP) and the 0-1 knapsack problem (0-1 KP). Experimental results show that the discrete version of the CLPSO algorithm based on the proposed S-PSO is promising.**

***Keywords*—particle swarm optimization, combinatorial optimization, discrete space**

I. INTRODUCTION

Particle swarm optimization (PSO) is a population-based stochastic optimization algorithm proposed by Kennedy and Eberhart in 1995 [1]. The algorithm is inspired by the social interaction behavior of bird flocking and fish schooling. To search for the optimal solution, each individual, which is 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 the problems in the continuous space [1]-[3].

The original PSO is simple and effective, but it is restricted to the continuous space. As many optimization problems are defined in a discrete space, the research on extending the PSO to solve discrete space COPs become attractive in recent years.

The first attempt to extend PSO to discrete domains is the binary version of PSO proposed by Kennedy and Eberhart [4]. As the binary PSO use binary coding schemes, the types of COPs that can be solved are limited. In order to define a more general discrete PSO (DPSO) algorithm, several approaches have been developed. These approaches can be classified into four types. The first type is the swap-operator based DPSO outlined by Clerc [5]. Another type of discrete PSO algorithms is characterized by a space transformation technique. The most common space transformation method is to view the position as a priority-based list [6]-[8]. The third type of discrete PSO algorithms defines the position and velocity as a fuzzy matrix [9][10]. Besides, there are also some hybrid PSO algorithms which integrate PSO with other meta-heuristics or problem-dependent local search methods to solve specific discrete optimization problems [11]-[13].

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 [5][8][9][12] are still far behind the ones obtained by the ant colony optimization (ACO) algorithm [14]. 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 explores the utility of PSO for the optimization problems in discrete space and proposes a set-based PSO (S-PSO) method based on the concept of sets and possibility theory. The S-PSO method has the following features. First, a set-based representation scheme is designed to describe the discrete search space as a universal set of elements. Based on this representation scheme, many different kinds of COPs can be mapped to the problems that can be solved by S-PSO. Second, in S-PSO, a candidate solution corresponds to a crisp subset 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

This work was supported in part by the National Science Foundation of China under Project 60573066, in part by the National Natural Science Foundation of China Joint Fund with Guangdong under Key Project U0835002, in part by the National High-Technology Research and Development Program of China, No. 2009AA01Z208.

Jun Zhang is the corresponding author. (email: junzhang@ieee.org)

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

replaced by the operators defined on crisp sets and sets with possibilities in S-PSO.

The above features enable the S-PSO method to follow the simple structure of the original PSO in the discrete set space. As a result, the search behavior of S-PSO is similar to that of the original PSO in continuous domains. Moreover, different improved variants of the original PSO, e.g., the PSO with different topologies [15][16] and the comprehensive learning PSO (CLPSO) [3], can also be extended to their discrete versions based on the S-PSO method. The proposed algorithm is tested in two famous COPs, the TSP and the 0-1 KP. Experimental results demonstrate the effectiveness of the proposed algorithm.

The rest of this paper is organized as follows. The next section gives a brief review of the original PSO in continuous domains and some of its representative successors. In Section III, the proposed S-PSO is described. Section IV further discusses the search behavior of S-PSO. Experimental studies are given in Section V, and the conclusions are finally summarized in Section VI.

II. TRADITIONAL PSO ALGORITHMS IN CONTINUOUS SPACE

In the original PSO [1][2], a swarm of M particles cooperate to search for the global optimum in the n-dimensional search space. Each particle i maintains a position $\boldsymbol{X}_i(x_i^1, x_i^2, \cdots, x_i^n)$ and a velocity $\boldsymbol{V}_i(v_i^1, v_i^2, \cdots, v_i^n)$. In each iteration, each particle updates the velocity and position as follows

$$v_i^j \leftarrow \omega v_i^j + c_1 r_1^j (pbest_i^j - x_i^j) + c_2 r_2^j (gbest^j - x_i^j) \quad (1)$$

$$x_i^j \leftarrow x_i^j + v_i^j \quad (2)$$

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

As the original PSO has the problem of premature convergence, developing different topologies has become an important strategy to improve the performance of PSO. While the original PSO uses the global best position ***GBest*** to update the velocity, Kennedy and Mendes [15][16] suggested to use the local best position $\boldsymbol{LBest}_i(lbest_i^1, lbest_i^2, \cdots, lbest_i^n)$ of a neighborhood and modified the velocity rule (1) into

$$v_i^j \leftarrow \omega v_i^j + c_1 r_1^j (pbest_i^j - x_i^j) + c_2 r_2^j (lbest_i^j - x_i^j) \quad (3)$$

The neighborhood of a particle can be defined by different topologies, such as Ring, URing, von Neumann, random, and so on [15][16]. In this paper, we term the global version GPSO, the local version with URing topology ULPSO, and the local version with von Neumann topology VPSO.

Some PSO variants modify the learning strategies of particles to prevent premature convergence. A representative PSO variant is the comprehensive learning PSO (CLPSO) proposed by Liang *et al.* [3]. The algorithm updates velocity using the following equation:

$$v_i^j \leftarrow \omega v_i^j + c \cdot r^j (pbest_{f_i(j)}^j - x_i^j) \quad (4)$$

where c is a parameter, r^j is a random number in [0,1], and $pbest_{f_i(j)}^j$ means the j^{th} dimension of the ***PBest*** position of the particle $f_i(j)$. $f_i(j)$ is given by first generating a random number *ran*. 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 to choose $f_i(j)$. CLPSO has been shown to be excellent for complex multimodal optimization problems [3].

III. THE PROPOSED SET-BASED PSO

A. Representation

According to [17], a COP can be defined by a triple (PS, f, Ω), where PS is a set of candidate solutions (search space), f is the objective function, and Ω is a set of constraints. The goal is to find a global optimal feasible solution $X^* \in PS$ that satisfies Ω and optimizes the objective function f.

According to [18], 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 COP (PS, f, Ω) is mapped to a problem that includes the following characteristics:

- A universal set E of elements is given. E has n dimensions, i.e., $E = E^1 \cup E^2 \cup \cdots \cup E^n$.
- A candidate solution to the problem $X \in PS$ corresponds to a subset of E, i.e., $X \subseteq E$. X also has n dimensions, i.e., $X = X^1 \cup X^2 \cup \cdots \cup X^n$, where $X^j \subseteq E^j$ (j=1,2,…,n).
- X is a feasible solution only if X satisfies the Ω.
- The objective of the COP problem is to find a feasible solution X^* that optimizes f.

With the above representation scheme, taking the symmetric TSP for example, each arc (j,k) is considered as an element. The universal set E corresponds to the set of arcs A. 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 is feasible only if the arcs in X form a Hamiltonian circuit of the graph. In the 0-1 KP, each dimension E^j can be defined as a set of two elements $E^j=\{(j,0),(j,1)\}$, where (j,1) indicates that item j is chosen in the bag and (j,0) means item j is not chosen. A candidate solution is $X = X^1 \cup X^2 \cup \cdots \cup X^n$, where $X^j=\{(j,0)\}$ or $X^j=\{(j,1)\}$. If X satisfies the capacity constraint, X is a feasible solution to the problem.

B. Velocity Updating

In PSO, a velocity gives the moving direction and tendency of a particle. In the original PSO, 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 the S-PSO method follows the same idea. For example, the velocity updating rule in the discrete version of GPSO based on S-PSO is given by

$$V_i^j \leftarrow \omega V_i^j + c_1 r_1^j (PBest_i^j - X_i^j) + c_2 r_2^j (GBest^j - X_i^j) \quad (5)$$

where V_i^j is the j^{th} dimension of the i^{th} particle's velocity and X_i^j is the j^{th} dimension of the i^{th} particle's position. $PBest_i^j$ is the j^{th}

dimension of the best-so-far solution found by particle i, and $GBest^j$ is the j^{th} dimension of the best-so-far solution found by all particles. $c_1>0$, $c_2>0$, and $\omega \in [0,1]$ are parameters. $r_1^j \in [0,1]$ and $r_2^j \in [0,1]$ are random numbers. Obviously, the velocity updating rule in equation (5) is in the same format as equation (1), but the positions (X_i^j, $PBest_i^j$ and $GBest^j$), velocity (V_i^j) and all related arithmetic operators in equation (5) are redefined in discrete domains as follows.

- **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 i^{th} particle as X_i ($X_i \subseteq E$). Similarly, $PBest_i \subseteq E$, $GBest \subseteq E$, and $LBest_i \subseteq E$ are the i^{th} particle's best-so-far position, global best-so-far position, and the i^{th} particle's local best-so-far position, respectively. At the beginning, X_i is initialized with a randomly-generated feasible solution.

- **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

$$V = \{e / p(e) \mid e \in E\} \quad (6)$$

that is, each element $e \in E$ has a possibility $p(e) \in [0,1]$ in the set with possibilities. In fact, if E' is a crisp subset of E, we can also regard E' as a set with possibilities defined on E by assigning $p(e)=1$ if $e \in E'$ and $p(e)=0$ if $e \notin E'$. In general, if $p(e)=0$, we omit the item $e/p(e)$ in the set for short.

Based on this definition, $V_i^j = \{e / p(e) \mid e \in E^j\}$ is a set with possibilities defined on E^j. The overall velocity for particle i is $V_i = \{e / p(e) \mid e \in E\}$, which is a set with possibilities defined on E. 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 a new position.

At the beginning of the algorithm, V_i is initialized by randomly selecting n elements from E and assigning each of these elements a random possibility $p(e) \in [0,1]$. The possibilities for the other unselected elements are set to 0.

- **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 $V = \{e / p(e) \mid e \in E\}$, their product is given by

$$cV = \{e / p'(e) \mid e \in E\}, \quad p'(e) = \begin{cases} 1, & \text{if } c \times p(e) > 1 \\ c \times p(e), & \text{otherwise} \end{cases} \quad (7)$$

- **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

$$A - B = \{e \mid e \in A \text{ and } e \notin B\} \quad (8)$$

Based on this definition, the effect of $GBest^j - X_i^j$ and $PBest_i^j - X_i^j$ is to find out the elements used by the $GBest$ ($PBest_i$) position but not used by the current position X_i. Such elements may have great potential to improve X_i.

- **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. According to definition 1, E' can be considered as a set with possibilities $E' = \{e / p(e) \mid e \in E\}$ where $p(e)=1$ if $e \in E'$ and $p(e)=0$ if $e \notin E'$. So the product of c and E' can be defined in the same way as definition 2, i.e.

$$cE' = \{e / p'(e) \mid e \in E\}, \quad p'(e) = \begin{cases} 1, & \text{if } e \in E' \text{ and } c > 1 \\ c, & \text{if } e \in E' \text{ and } 0 \leq c \leq 1 \\ 0 & \text{if } e \notin E' \end{cases} \quad (9)$$

- **Velocity + Velocity**

Finally, we define the union operator between two sets with possibilities.

Definition 4 (union operator between two sets with possibilities): Given two sets with possibilities $V_1 = \{e / p_1(e) \mid e \in E\}$ and $V_2 = \{e / p_2(e) \mid e \in E\}$ defined on E, $V_1 \cup V_2$ is defined as

$$V_1 \cup V_2 = V_1 + V_2 = \{e / \max(p_1(e), p_2(e)) \mid e \in E\} \quad (10)$$

that is, the possibility $p(e)$ for element e in $V_1 \cup V_2$ is set to the larger one between $p_1(e)$ and $p_2(e)$. In order to be consistent with the notation in the original PSO, $V_1 \cup V_2$ is also denoted as V_1+V_2 in this paper.

- **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 for particle i, different dimensions of velocity V_i^j 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 velocity updating can be kept consistent.

- **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 [equation (5)] is summarized in Fig. 1.

The velocity updating rule of the other PSO variants can also be extended to discrete versions based on S-PSO in a similar way. The velocity updating rules in the PSO with different topologies [equation (3)] and the CLPSO [equation (4)] are as follows

$$V_i^j \leftarrow \omega V_i^j + c_1 r_1^j (PBest_i^j - X_i^j) + c_2 r_2^j (LBest_i^j - X_i^j) \quad (11)$$

$$V_i^j \leftarrow \omega V_i^j + c r^j (PBest_{fi(j)}^j - X_i^j) \quad (12)$$

where $LBest_i$ is the local best position of a neighborhood and $PBest_{fi(j)}^j$ is the j^{th} 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 equation (11) and (12)

are the same as the ones in equation (3) and (4). In the rest of this paper, we name the discrete version of GPSO, VPSO, ULPSO [15][16], and CLPSO [3] based on S-PSO as S-GPSO, S-VPSO, S-ULPSO, and S-CLPSO, respectively.

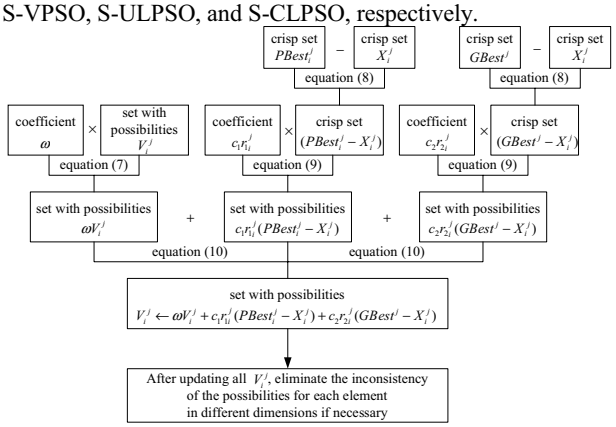


Fig. 1. The arithmetic operators in the velocity updating rule of S-PSO

```
procedure position_updating(X_i, V_i)
    generate a random number α ∈ (0,1);
    for each dimension j (j = 1,2,⋯,n)
        cut_α(V_i^j) = {e | e / p(e) ∈ V_i^j and p(e) ≥ α};
    end for
    NEW_X_i = Φ;
    for each dimension j (j = 1,2,⋯,n)
        Candidate_Set_i^j = {e | e ∈ cut_α(V_i^j) and e satisfies Ω};
        while the construction of NEW_X_i^j is not finished and Candidate_Set_i^j ≠ Φ
            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 | e ∈ X_i^j and e satisfies Ω};
            while the construction of NEW_X_i^j is not finished and Candidate_Set_i^j ≠ Φ
                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 | e ∈ E^j and e satisfies Ω};
            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. 2. pseudo code for the position updating procedure

C. Position Updating

After updating velocity, particle i uses the new velocity V_i 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 NEW_X_i, in S-PSO, the i^{th} particle applies the position updating procedure *position_updating*(X_i, V_i) given in Fig. 2 to build new positions. That is, in S-PSO, particle i updates its position as follows

$$X_i \leftarrow position\_updating(X_i, V_i) \qquad (13)$$

1) The set with possibilities V_i is converted to a crisp set.

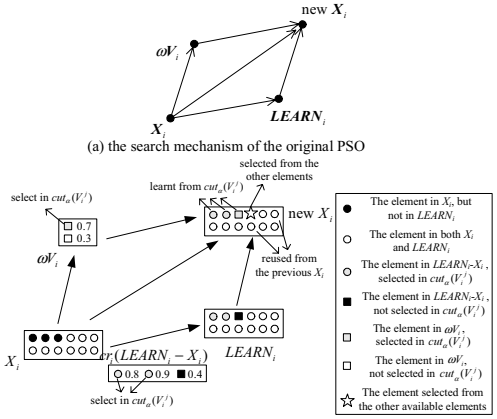
In each iteration, a random number $\alpha \in (0,1)$ is generated for each particle. For each element e in the j^{th} dimension, if its corresponding possibility $p(e)$ in the new velocity V_i^j is not smaller than α, element e is reserved in a crisp set , that is,

$$cut_\alpha(V_i^j) = \{e \mid e / p(e) \in V_i^j \text{ and } p(e) \ge \alpha\} \qquad (14)$$

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 $NEW\_X_i=\Phi$. We denote the j^{th} 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 $NEW\_X_i^j$. 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 $NEW\_X_i^j$. 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 $NEW\_X_i^j$. After all $NEW\_X_i^j$ have been completed, the construction of NEW_X_i is finished.

Note that there is a selection operator in the position updating rule. The operator can be either 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 to add to the new position. The performance of different types of selection operator will be discussed in the next section.



(a) the search mechanism of the original PSO

(b) the search mechanism of the S-PSO

Fig. 3. Comparison between the original PSO and the S-PSO

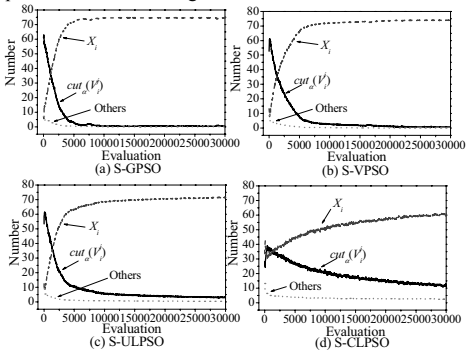


Fig. 4. The sources of the arcs that compose a new position

IV. BEHAVIOR OF THE PROPOSED SET-BASED PSO

A. Behaviors of Different DPSO Variants Based on S-PSO

We first give an insight into the essence of the velocity and the position updating rules in S-PSO. The comparison between the original PSO and the S-PSO is illustrated in Fig. 3. In the original PSO, the particle uses the velocity vector $\boldsymbol{V}_i$ and the vector that learned from the previous search experience $\boldsymbol{Learn}_i$-$\boldsymbol{X}_i$ to modify the position as shown in Fig. 3(a). Here, $\boldsymbol{Learn}_i$ represents the position from which the particle learns.

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 $cr(Learn_i\text{-}X_i)$. ($Learn_i$ represents $GBest$, $LBest_i$, $PBest_i$, etc.) In terms of the definition of "position-position", the effect of $Learn_i\text{-}X_i$ is 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 $Learn_i\text{-}X_i$ iteratively to improve their current positions, as illustrated in Fig. 3(b).

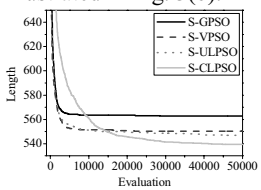


Fig. 5. Comparison of the convergence speeds among the four discrete PSO variants based on S-PSO in the TSP instance eil76. The results are averaged over 30 runs.

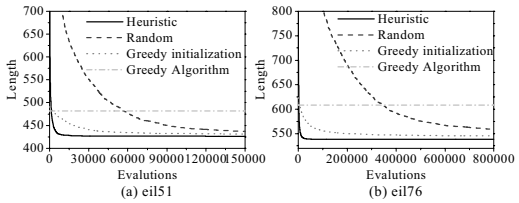


Fig. 6. Comparison between the random and the greedy selection operator: the results are averaged over 30 runs. In the plots, "Heuristic" and "Random" represents the S-CLPSO algorithm with heuristic-based selection operator and random selection operator, respectively. "Greedy initialization" represents 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.

According to the position updating procedure, the elements in a new position come from three sources: $cut_\alpha(V_i^j)$, X_i^j and the other available elements. The sources of the elements in a new position are tightly related to the convergence behavior of S-PSO. The fact that a large number of elements coming from $cut_\alpha(V_i^j)$ implies a very "fast" flying speed, as the particle can learn from a lot of elements from $cut_\alpha(V_i^j)$. Oppositely, if the number of elements from $cut_\alpha(V_i^j)$ is small, the particle only searches in a small neighborhood. We run the S-GPSO, S-VPSO, S-ULPSO, and S-CLPSO algorithms on a TSP instance eil76 from TSPLIB [19]. In Fig. 4, it can be seen that only a few arcs come from neither $cut_\alpha(V_i^j)$ nor the X_i. At the early stage, most of the arcs come from $cut_\alpha(V_i^j)$. In this case, a lot of new elements can be introduced to improve X_i, and the search shows a diverse behavior. As the procedure continues, the differences between $Learn_i$ and X_i become smaller, and thus the number of elements in $cut_\alpha(V_i^j)$ reduces. As a result, more arcs come from the previous X_i. In the S-GPSO, S-VPSO, and S-ULPSO, after 5000 evaluations, only less than 5 arcs come from $cut_\alpha(V_i^j)$. In this situation, the search procedure has converged to a small search area. In an extreme case, if all arcs come from the previous X_i, the search procedure stagnates. Similar to the CLPSO algorithm for continuous problems [3], compared with the other S-PSO variants, the S-CLPSO based on S-PSO is also able to keep the swarm's diversity (Fig. 4). Therefore it manages to achieve better performance compared with S-GPSO, S-VPSO, and S-ULPSO (Fig. 5).

B. The Selection Operator in Position Updating

In the position updating procedure (Fig. 2), there is a selection operator. The selection operator can be either random or heuristic-based. The performance of these two selection operators is tested. The results are plotted in Fig. 6. Obviously, the algorithm with the heuristic-based selection operator performs better, as some problem-dependent information is employed to guide the search direction. In fact, many successful algorithms for COPs, e.g., ACO [14][17], also employ problem-dependent heuristic information. Such heuristic information is able to accelerate the search process especially in large-scale instances.

Note that using the heuristic-based selection operator does not mean that the algorithm behaves in the same way as the greedy-search algorithm. The algorithm still follows the learning mechanism of S-PSO. In fact, according to the results in Fig. 6, even using the random selection operator instead of the heuristic-based selection operator, the algorithm can also achieve much better results than the ones obtained by the greedy-search algorithm, though the search speed is slow. This demonstrates that the learning mechanism in S-CLPSO is indeed contributing. The effect of the heuristic-based selection operator is to accelerate the search speed of the algorithm.

In some COPs, it may be difficult to design a suitable heuristic-based selection operator. In this situation, to accelerate the search speed, we can apply some deterministic search techniques to generate good initial solutions. For example, in the TSP, we use the greedy-search algorithm to generate initial solutions, but only use the random selection operator in the course of search. Fig. 6 reveals that this scheme also manages to achieve acceptable results quickly.

C. Inertia Weight and Acceleration Coefficient

The most important parameters in the PSO algorithms for continuous domains are inertia weight ω and acceleration coefficients (c_1 and c_2 in GPSO, and c in CLPSO). Interestingly, in the experiments, it is found that these parameters are able to play similar roles 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 $\alpha \in [0,1]$ can be reserved in $cut_\alpha(V_i^j)$. 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 $\omega^t p(e)$. Therefore, a small ω will make particles forget the elements in the previous V_i quickly, and the size of $cut_\alpha(V_i^j)$ becomes small. As most of the elements come from the previous X_i, the algorithm performs a local search behavior. In contrast, a large ω will reinforce the elements in the previous V_i. Typically when ω=1, the elements in previous V_i will never be forgotten. As there are too many elements in $cut_\alpha(V_i^j)$, particles only use the elements in $cut_\alpha(V_i^j)$, and the search procedure becomes too diverse to find good solutions. We also test the scheme proposed by [2] to decrease the value of ω from 0.9 to 0.4. The results in Fig. 7 show that this configuration for ω manages to balance the convergence and diversity and performs well.

The acceleration coefficient c weighs the importance of newly-learnt elements. Since each element e learnt from $LEARN_i$-X_i is assigned a possibility $c \times r$, a larger c will give a better chance for reserving e in $cut_\alpha(V_i^j)$. 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. 8, 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 the figure because its results are too bad and out of the scale of the figure.) When c>1, the diversity and convergence of the algorithm can be balanced, and the algorithm performs well.

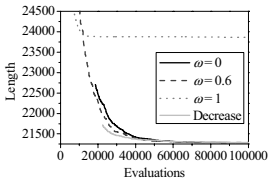


Fig. 7. Performance of the algorithm with different inertia weight values in the TSP instance kroA100. "Decrease" means the scheme to decrease the value of ω from 0.9 to 0.4. The results are averaged over 20 runs.

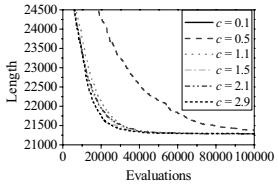


Fig. 8. Performance of the algorithm with different acceleration coefficient values in the TSP instance kroA100. The results are averaged over 20 runs.

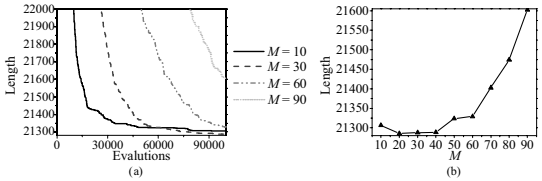


Fig. 9. Performance of the algorithm with different swarm sizes in the TSP instance kroA100. The results are averaged over 20 runs.

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

TSP instances (derived from the TSPLIB)			
instance name	maximum number of solutions	instance name	maximum number of solutions
eil51	25000	eil101	50000
Berlin52	25000	lin105	50000
st70	35000	kroA150	75000
eil76	35000	kroA200	100000
pr76	35000	pr299	150000
kroA100	50000	lin318	150000
0-1 KP instances (derived from Ref. [20])			
100	5000	750	37500
250	12500	1000	50000
500	25000	2000	100000

V. EXPERIMENTAL RESULTS AND COMPARISONS

In this section, we present the experimental results of S-PSO. The algorithm is tested on the TSP and the 0-1 KP. The TSP instances can be found in the TSPLIB [19], and the 0-1 KP instances are generated by the same method as in [20]. The information of instances is given in TABLE I. The experiments are performed on a machine with Pentium IV 2.80 GHz CPU, 256 MB of RAM, and MS Windows XP operation system. As has been mentioned before, S-CLPSO 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 with some meta-heuristics for COPs.

According to the analysis in Section IV, we apply the classical settings in [2] to decrease ω from 0.9 to 0.4 linearly and set c=2.0. The swarm size M is also an important parameter in PSO. From Fig. 9, 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 than that of M=20. In the experiment, as the population size of the ACS algorithm [14] is 10, to make a fair comparison, we also use M=10 in the comparison studies.

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 [7] and the discrete PSO algorithm proposed in [12]. In [7], the PSO-TS-CO-2opt algorithm is integrated with a 2-opt local search procedure and a chaotic operation (CO). 100000 solutions are generated in each single run and the results are averaged over ten runs. The mean results are recorded. In [12], the discrete PSO algorithm is integrated with a delete-crossover process. Each instance is run 100 times. The best and the worst solutions are recorded. The S-CLPSO algorithm with the aforementioned parameter configurations and the heuristic-based selection operator is run 50 times. The results are compared with the data reported in [7] and [12] in TABLE II. From the results, it is apparent that the S-CLPSO outperforms the other two algorithms in all cases.

TABLE II COMPARISON BETWEEN S-CLPSO AND THE OTHER EXISTING PSO-BASED APPROACHES

	S-CLPSO			PSO-TS-CO-2opt [7]			Discrete PSO [12]		
	best	worst	mean	best	worst	mean	best	worst	mean
eil51	**426**	**433**	**427.5**	N/A	N/A	440.9	427	452	N/A
Berlin52	**7542**	**7618**	**7544.3**	N/A	N/A	7704	**7542**	8362	N/A
st70	**675**	**687**	**687.7**	N/A	N/A	N/A	**675**	742	N/A
eil76	**538**	**547**	**540.3**	N/A	N/A	560.7	546	579	N/A
pr76	**108159**	**110213**	**108447**	N/A	N/A	N/A	108280	124365	N/A

The results of S-CLPSO are averaged over 50 runs. The results of PSO-TS-CO- 2opt and the discrete PSO algorithm are derived from [14] and [20].

TABLE III COMPARISON BETWEEN S-CLPSO AND ACS

instance	best known	algorithm	best	worst	mean	deviation	Time (ms)	t-test (ACS - S-CLPSO)
eil51	426	S-CLPSO	**426**	**433**	**427.5**	**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	108159	S-CLPSO	**108159**	**110213**	**108447**	**475.80**	**2581**	9.247 #
		ACS	**108159**	113096	110197.7	1251.31	6453	
kroA100	21282	S-CLPSO	**21282**	**21658**	**21352.5**	**84.51**	**5047**	4.823 #
		ACS	**21282**	22823	21584	328.66	15304	
eil1101	629	S-CLPSO	**629**	**646**	**637.1**	**4.45**	**5563**	7.575 #
		ACS	631	661	646.3	7.30	15581	
lin105	14379	S-CLPSO	**14379**	**14561**	**14462.7**	**55.29**	**5393**	3.960 #
		ACS	**14379**	15121	14539.5	125.48	17225	
kroA150	26524	S-CLPSO	**26537**	**27317**	**26892.1**	**165.36**	**14389**	6.898 #
		ACS	26734	28008	27202.6	272.03	48153	
kroA200	29368	S-CLPSO	**29399**	**30139**	**29722.4**	**153.58**	**37072**	4.800 #
		ACS	29506	31138	30001.5	381.33	111003	
pr299	48191	S-CLPSO	**48478**	**50427**	**49222.5**	**415.12**	**249369**	5.119 #
		ACS	48828	50936	49721.6	550.41	338787	
lin318	42029	S-CLPSO	**42719**	**44209**	**43518.4**	**335.92**	**314635**	1.513
		ACS	43050	44716	43624.4	364.02	383877	

The results are averaged over 50 runs. " # " means the value of t with 49 degrees of freedom is significant at α=0.05 by a two-tailed test.

We also compare the S-CLPSO algorithm with the ant colony system (ACS) [14]. In the comparison, the parameter settings in ACS are the same as [14], that is, β=2, q_0=0.9, α=ρ=0.1, and the number of ants is 10. In every single run, both algorithms generate the same number of solutions. The number is given in TABLE I. Each instance is run for 50 times. The best results, worst results, average results, deviations, and the average time to complete a single run (in millisecond) are shown in TABLE III. It can be seen that the S-CLPSO algorithm outperforms the ACS algorithm in all test instances. We also study the convergent speed of these two algorithms. The evolutionary curves of both algorithms are given in Fig. 10. The curves are based on the average results of 50 runs. Though S-CLPSO performs worse than ACS in the early stages, in the later stages, S-CLPSO is still diverse enough to avoid being trapped in local optima. Therefore, S-CLPSO manages to yield much better results.

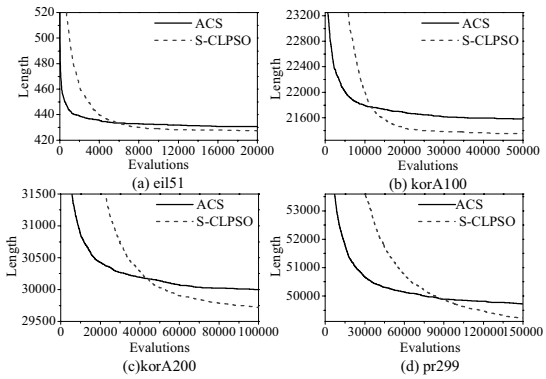


Fig. 10. Comparison between the search speed between S-CLPSO and ACS.

TABLE IV COMPARISON BETWEEN THE S-CLPSO, BPSO, AND QEA IN THE 0-1 KP

instance	algorithm	best	mean	deviation	time(ms)
100	S-CLPSO (h)	**610.02**	**609.05**	1.915	**133.1**
	S-CLPSO (r)	**610.02**	605.77	**1.779**	134.7
	BPSO	610.01	603.91	2.528	280.0
	QEA	610.00	604.20	2.528	394.7
250	S-CLPSO (h)	**1522.94**	**1521.00**	2.423	**773.7**
	S-CLPSO (r)	1522.92	1516.69	**2.364**	795.6
	BPSO	1512.93	1507.03	4.594	1724.0
	QEA	1517.93	1505.43	5.997	2435.6
500	S-CLPSO (h)	**3074.39**	**3073.68**	**1.751**	**3128.1**
	S-CLPSO (r)	3074.37	3064.74	3.609	3140.9
	BPSO	3059.39	3048.08	6.448	6986.2
	QEA	3054.38	3038.68	7.356	9750.9
750	S-CLPSO (h)	**4604.44**	**4599.23**	**2.842**	**7199.6**
	S-CLPSO (r)	4589.44	4581.40	5.783	7255.0
	BPSO	4579.44	4558.24	8.722	16074.0
	QEA	4554.44	4531.64	9.044	21931.6
1000	S-CLPSO (h)	**6128.75**	**6122.21**	**3.499**	12832.2
	S-CLPSO (r)	6108.56	6095.97	5.797	**12745.9**
	BPSO	6093.74	6071.14	11.579	29025.6
	QEA	6043.75	6019.43	16.560	39105.3
2000	S-CLPSO (h)	**12245.81**	**12239.77**	**4.613**	54117.2
	S-CLPSO (r)	12195.80	12174.38	9.410	**51353.8**
	BPSO	12175.81	12153.85	11.725	119600.3
	QEA	12050.81	12009.29	20.711	156880.9

The results are averaged over 50 runs.

B. Performance on the 0-1 KP

We also study the performance of S-CLPSO in the 0-1 KP and compare it with BPSO [4] and the quantum-inspired evolutionary algorithm (QEA) [20].

In the QEA [20], a local refinement strategy is applied to improve the performance of the algorithm. The strategy removes random items from the knapsack iteratively until the knapsack is not overfilled, and adds random items to the knapsack iteratively until the knapsack becomes overfilled. In order to make fair comparison, we apply this strategy to all of the three algorithms in the experiment. Both the S-CLPSO algorithms with the random selection operator and the heuristic-based selection operator are implemented. The heuristic-based selection operator for the 0-1 KP is as follows [21]. Before executing S-CLPSO, all items are sorted by v_j/w_j in ascending order. Suppose the j^{th} item's rank is $rank_j$. In the selection operator, if both (j,0) and (j,1) are in the candidate set, a random number $ran \in [0,1]$ is generated. If $ran<rank_j/n$, (j,1) is chosen. Otherwise, (j,0) is chosen.

Because each candidate solution to the 0-1 KP is actually a binary string, the discrete binary version of PSO [4] can be directly applied to the 0-1 KP. In BPSO, each dimension j has a bit x_j (x_j=0 or 1) and a velocity v_j. v_j determines the possibility that the bit x_j will take on the value one or zero. While building a new solution, if $S(v_j)$ is larger than a random number, then x_j=1, otherwise x_j=0 ($S(v)$ is a sigmoid function). In this case, the method to construction a new position in BPSO does not use the information of the previous position, which is quite different from the method in the proposed S-PSO method. In the experiment, the parameter configurations for BPSO suggested in [4] are used. That is, c_1=c_2=2, $V_{\max}$=6, and the population size is 20.

The parameter configurations for QEA are the same as the QEA(3) version given in [20]. These configurations are shown to be the best compared with the other QEA versions.

For each run, the algorithms generate the same number of solutions. The number is given in TABLE I. Each instance is

run for 50 times. The experiment performs on the same machine as the one used in previous experiments. Experimental results are given in TABLE IV. In the table, S-CLPSO(h) means the algorithm with the heuristic-based selection scheme, and S-CLPSO(r) represents the algorithm with the random selection scheme. Apparently, the S-CLPSO(h) algorithm has the best performance. Both S-CLPSO(h) and S-CLPSO(r) manage to yield better solutions than BPSO and QEA in all instances. Moreover, the S-CLPSO consumes less execution time to achieve these results. This is because QEA needs to evaluate a lot of trigonometric functions, and BPSO uses the sigmoid function, which includes an "exp" operator. These results reveal that the S-CLPSO based on S-PSO is promising.

VI. CONCLUSION

A set-based particle swarm optimization (S-PSO) method for discrete space COPs has been proposed. S-PSO follows the basic idea of the original PSO. In order to solve discrete space problems, S-PSO represents the search space of the discrete problem with the concept of set. The term "position", "velocity" and all related arithmetic operators in the original PSO are replaced by the operators defined on sets and sets with possibilities. Based on S-PSO, different PSO variants can be extended to their discrete versions. Experimental results demonstrate the effectiveness of the proposed algorithm.

REFERENCES

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942-1948, 1995.

[2] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 69-73, 1998.

[3] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions Evolutionary Computation*, vol.10, no.3, pp. 281-295, 2006.

[4] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of IEEE International Conference on System, Man, and Cybernetics*, pp 4104–4109, 1997.

[5] M. Clerc, "Discrete particle swarm optimization," *New Optimization Techniques in Engineering*, Springer-Verlag. 2004.

[6] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems*, vol. 26, pp. 363-371, 2002.

[7] D.Y. Sha and C. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers & Industrial Engineering*, vol. 51, pp. 791–808, 2006.

[8] W. Pang, *et al*., "Modified particle swarm optimization based on space transformation for solving traveling salesman problem," in *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, pp. 2342-2348, 2004.

[9] W. Pang, K.-P. Wang, C.-G Zhou, and L.-J. Dong, "Fuzzy discrete particle swarm optimization for solving traveling salesman problem," in *Proceedings of the 4th International Conference on Computer and Information Technology* (*CIT'04*), pp. 796-800, 2004.

[10] C.-J. Liao, C.-T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization flowshop scheduling problems," *Computers and Operations Research*, vol. 34, pp. 3099-3111, 2007.

[11] Y. Wang, *et al*., "A novel quantum swarm evolutionary algorithm and its applications," *Neurocomputing*, vol. 70, pp. 633-640, 2007.

[12] 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," *Information Processing Letters*, vol. 103, pp. 169-176, 2007.

[13] C.-T. Tseng and C.-J. Liao, "A discrete particle swarm optimization for lot-streaming flowshop scheduling problem," *European Journal of Operational Research*, in press.

[14] Marco Dorigo and Luca Maria Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp.53-66, 1997.

[15] J. Kennedy, "Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance," in *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, vol. 3, 1999.

[16] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC02)*, vol. 2, pp. 1671-1676, 2002.

[17] M. Dorigo, T. Stützle, *Ant colony optimization*, MIT Press, 2004.

[18] S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," Operations Research, pp. 498-516, 1972.

[19] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376-384, 1991.

[20] K. Han, J. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization", *IEEE Transactions on Evolutionary Computation,* vol. 6, no. 6, pp. 580-593, 2002.

[21] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Sterin, *Introduction to algorithms (Second Edition)*, MIT Press, 2001