Expert Systems With Applications 44 (2016) 413–431

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Particle Swarm Algorithm variants for the Quadratic Assignment Problems - A probabilistic learning approach

Faizal Hafiz[a,*], Adel Abdennour[b]

[a] Sustainable Energy Technologies Center, College of Engineering, King Saud University, P.O. Box 800, Riyadh 11421, KSA
[b] Department of Electrical Engineering, College of Engineering, King Saud University, P.O. Box 800, Riyadh 11421, KSA

ARTICLE INFO

Keywords:
Combinatorial optimization
Fitness landscapes
Meta-heuristics
Particle Swarm Optimization (PSO)
Quadratic Assignment Problem (QAP)
Search space analysis
Swarm intelligence

ABSTRACT

The Quadratic Assignment Problem (QAP) has attracted considerable research efforts due to its importance for a number of real life problems, in addition to its acknowledged difficulty. Almost all of the well-known nature-mimicking algorithms have been applied to solve the QAP. However, the Particle Swarm Optimization (PSO), which has proven to be very effective in various applications, has received little attention at this front. The reason can be ascribed to the Euclidian-distance based learning concept (at the core of the algorithm) which makes PSO, in its present form, unsuitable for combinatorial optimization problems. In this article, a new probability-based approach is proposed for the learning in PSO. Based on this learning concept, a generic framework is developed to discretize PSO and its variants, to make them suitable for combinatorial optimization. Five well-known PSO variants are discretized based on this proposed framework. A comparative study of all discretized PSO variants is also included. Moreover, the proposed framework is compared to other attempts to discretize PSO, in addition to three other meta-heuristic approaches. The comparison revealed that the proposed technique is more effective.

1. Introduction

The Quadratic Assignment Problem (QAP) was first suggested by Koopmans and Beckmann (1957). The problem requires the assignments of n facilities to n locations given the distance matrix (a) between the locations and the cost of material flow matrix (b) between them. The objective is to find the assignment (π) which results in a minimum cost (C). The problem can be described mathematically as follows (Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007):

$$C(\pi) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} b_{\pi(i)\,\pi(j)} \tag{1}$$

where n is the total number of facilities/locations, a_{ij} is the distance between location i and location j, b_{ij} is the flow of materials between location i and location j and π is the permutation of the set $\{1, 2, \ldots, n\}$.The problem is NP hard and is considered one of the most difficult optimization problems; especially for larger instances ($n > 30$) (Loiola et al., 2007).

The QAP has attracted many researchers due to its complexity and its importance for a number of real-life problems. The problem is considered to be a classical challenge (Loiola et al., 2007). Many real-life applications can be formulated as a QAP. These include, but not limited to, facility layout problems, scheduling problems, travelling salesman problems, maximum clique problem in the information sciences, the bin packing problem, graph partitioning, memory layout in digital signal processors, keyboard layout problem, backboard wiring to minimize the connection between the electronic components, and reaction chemistry analysis (Burkard, Karisch, & Rendl, 1997; Loiola et al., 2007).

Meta-heuristics have been an attractive choice to solve the QAP. Several nature-inspired meta-heuristics have been applied to QAP which include Tabu Search (TS; Skorin-Kapov, 1990; Taillard,1991; Skorin-Kapov, 1994; Bland & Dawson, 1991; Chakrapani & Skorin-Kapov, 1993; Misevicius, 2005; James, Rego, & Glover, 2009), Evolutionary Strategy (ES; Nissen, 1994), Genetic Algorithm (GA; Bui et al., 1994; Drezner & Marcoulides, 2004; El-Baz, 2004), and Ant Colony Optimization (ACO; Dorigo, Maniezzo, & Colorni, 1996; Gambardella, Taillard, & Dorigo, 1999; Maniezzo & Colorni, 1999; Stützle & Dorigo, 1999). In the literature, TS and local search are found to be more effective for the QAP (Loiola et al., 2007). Based on this fact, several memetic or hybrid approaches which combine TS or local search with the traditional meta-heuristics have been proposed and proven to be effective (Benlic & Hao, 2013, 2015; Drezner, 2008; Stützle, 2006).

Most of these paradigms have to be modified to adapt to the requirements of combinatorial optimization problems, and

* Corresponding author. Tel.: +966 114679953.
E-mail addresses: faizalhafiz@gmail.com (F. Hafiz), adnnour@ksu.edu.sa (A. Abdennour).

http://dx.doi.org/10.1016/j.eswa.2015.09.032
0957-4174/© 2015 Elsevier Ltd. All rights reserved.

considerable research efforts have been dedicated for this purpose. However, Particle Swarm Optimization (PSO) which has proven to be very useful in various other applications (AlRashidi & El-Hawary, 2009; Kulkarni & Venayagamoorthy, 2011; Mangat, 2012), has received little attention at this front. The major benefit of PSO over other nature-inspired paradigms is its simplicity. PSO was first proposed by Kennedy and Eberhart (1995) and it has found numerous applications. The applications are limited mostly to continuous domain (AlRashidi & El-Hawary, 2009; Kulkarni & Venayagamoorthy, 2011; Mangat, 2012), which is due in part to the fact that PSO in its canonical form is not readily suitable for discrete optimization and even more so for QAP.

The solution of QAP requires a unique assignment of a facility for each location. However, unlike ACO, PSO is not inherently discrete and, hence, cannot be used directly on assignment problems. The main reason for this limitation is the *Euclidian-distance*-based learning concept used in PSO. It is essential to develop a new learning concept, if PSO or any of its variants is to be used for solving QAP. This can be viewed as a huge untapped potential; if a proper framework along with a new learning concept is developed, it will enable the use of existing research efforts previously developed for the PSO in continuous domain. This is the main motivation of the present work.

The essence of the PSO is the incorporation of social learning through shared experience. Since its inception, many modifications and improvements have been proposed to PSO, differing on the bases of information sharing (neighborhood topology), learning exemplar, or parameter control. The underlying idea, however, remains the same.

At a given iteration, the basic PSO update rules of the d^{th} dimension for the i^{th} particle in a D-dimensional search space can be given by Kennedy and Eberhart (1995) and Yuhui and Eberhart (1999):

$$v_{i,\ d} = (w \times v_{i,d}) + c_1 \times r_1 \times (pbest_{i,d} - x_{i,d}) + c_2 \times r_2 \times (gbest_d - x_{i,d}) \tag{2}$$

$$x_{i,\ d} = x_{i,d} + v_{i,d} \tag{3}$$

where, x and v are the position and the velocity, respectively; w, c_1 and c_2 are the inertia weight and the acceleration constants, respectively. The variable $pbest_d$ is the personal best position attained by the d^{th} dimension of the particle and $gbest_d$ is the d^{th} dimension of the global best position attained by the swarm.

As seen in (2) and (3), the notion of learning in PSO is based on the distance. Each particle's next move is determined based on its own experience and that of its peers', which is defined as the *Euclidean* distance between the particle's current position and the corresponding learning exemplar. Through the search, the particle tries to minimize this distance by flying towards its learning exemplar(s) and, in this process, it is expected to find new promising regions in the fitness landscape. The biggest challenge for the algorithm's application to combinatorial problems is to find an appropriate distance measure; since the Euclidian distance cannot be used for problems of this type.

To overcome this, Kennedy and Eberhart proposed the binary version of the algorithm in Kennedy and Eberhart (1997). In the binary version of PSO, the concept of the velocity is redefined to reflect the particle's probability of changing state. Several variants of the binary PSO were proposed, they differ mainly in the way new positions are created from the velocity (Wang, Wang, Fu, & Zhen, 2008; Menhas, Wang, Fei, & Ma, 2011; Shen et al., 2014). In another method to discretize PSO, a rank-based approach was used (Liu, Wang, & Jin, 2007). In this approach, the original velocity and position update equations are retained. To generate the feasible solution, the dimension of the new position is ranked based on the velocity values. This approach is not suitable for QAP problems, as the ranking based approach does not necessarily reflect the quality of the solution.

In Hu, Eberhart, and Shi (2003), a PSO variant is proposed for an assignment problem. In this variant, each particle is represented as a permutation and the velocity is used to indicate the probability of swapping with the learning exemplar. Since the swapping can be done to mimic only one learning exemplar (neighborhood best or personal best), the particle loses either social or cognitive learning ability. In addition, for each particle, only one swap is performed (probability of which is controlled by velocity); which severely limits the swarm's exploration capability.

Another popular approach is to rely on the set-based operations to discretize PSO (Clerc, 2004; Correa, Freitas, & Johnson, 2006; Neethling & Engelbrecht, 2006; Veenhuis, 2008; Chen et al., 2010; Langeveld & Engelbrecht, 2011; Langeveld & Engelbrecht, 2012). In Correa et al. (2006), a set-based approach is used for attribute selection in data mining applications; where the position is represented by attribute set and the velocity is a two dimensional array. The velocity array stores the likelihood of each attribute's selection and, during each iteration, this likelihood is influenced by the learning exemplars and particle's previous position itself. However, the approach is specifically designed for the attribute selection problem and it is not generic. Neethling and Engelbrecht (2006) proposed a set-based variant in which a new particle position is generated from the elements of the personal best, neighborhood best, and other random elements based on three different probabilities. However, this approach is found to be lacking in the exploratory capabilities (Langeveld & Engelbrecht, 2011). Veenhuis (2008) proposed another set-based approach to PSO for problems whose search dimensions are not known in advance. However, the proposed set-based operations lead to set bloating effect in both the position and the velocity and requires a complex mechanism to solve this problem. A set-based approach with redefined operators was proposed in Clerc (2004), Chen et al. (2010), and Langeveld and Engelbrecht (2012). Clerc (2004) redefined the mathematical operators used in the velocity update equation in the discrete PSO applied to TSP. In Langeveld and Engelbrecht (2011, 2012), similar redefined operators are used for the Multidimensional Knapsack Problem (MKP). These approaches are generic, but the proposed set-based operation leads to increased complexity of an otherwise fairly simple PSO algorithm. In another set-based variant (Chen et al., 2010), a set with possibilities was introduced to solve the TSP. In the proposed approach, the velocity stores the learning elements for each dimension along with their possibilities. The new position is created from the corresponding velocity using problem specific heuristics; hence, the approach is not generic and cannot be easily applied to QAP.

Another approach to discretize PSO uses the fuzzy mapping to represent the position and velocity (Pang, Wang, Zhou, & Dong, 2004; Liu, Abraham, & Clerc, 2007a; Liu, Abraham, & Zhang, 2007b). Each particle's position and the velocity is given by $n\times n$ matrix (n is the search dimension of the problem), wherein each element of the matrix represents the degree of membership of a given facility to a corresponding location. The velocity update equation of the canonical PSO is retained and the new position is decoded from the velocity using heuristics based on maximum membership.

The common limitation of these approaches (Clerc, 2004; Correa et al., 2006; Neethling & Engelbrecht, 2006; Veenhuis, 2008; Chen et al., 2010; Langeveld & Engelbrecht, 2011; Langeveld & Engelbrecht, 2012; Pang et al., 2004; Liu et al., 2007a,b) is that each one is specifically designed as a new algorithm to solve a specific problem. Hence, they lack the ability to harness the existing research efforts dedicated to improving PSO's performance in the continuous domain.

The main objective of this work is to develop a simple and effective approach that enables PSO to optimize discrete and combinatorial problems, while preserving its simplicity. The name "discretized PSO" is used to refer to the PSO (or any of its variants) when applied for discrete/combinatorial problems. This "discretization" approach is very important in the sense that it enables the transfer of existing research efforts from the continuous to the discrete domain for QAP applications. For this purpose, a new probabilistic representation for the particle's velocity is proposed. Based on the proposed framework, PSO and five well-known variants are discretized and applied to the QAP. In addition, the proposed framework is compared with the other similar approaches to discretize PSO and the Ant Systems.

The article is organized as follows: the proposed framework is explained in Section 2. PSO and five of its variants are discretized following the proposed framework in Section 3. A comparative evaluation is presented in Section 4 and the conclusions are presented in Section 5.

2. Proposed discretization framework

In this section, the details of the proposed algorithm are presented. The velocity and position update procedures are explained through an example.

2.1. Position encoding

The encoding scheme employed is simple, each particle represents a valid permutation π, i.e., each dimension of the particle represents a location and each value represents the corresponding facility. For example, for $n = 4$, the particle $\{4\ 3\ 2\ 1\}$ indicates that the fourth facility is assigned to the first location, third facility to second location and so on.

2.2. Velocity update

In the proposed approach, each velocity element represents the selection probability of a facility for a particular location. A similar approach was used by Kennedy & Eberhart (1997) in their binary version of the PSO. However, in our approach, the velocity is a guide map for all the particles, based on which a facility is selected for each location of the particle. In each iteration, the velocity is updated based on the particle's performance.

To understand the velocity update process, consider a QAP with n instances (n-dimensional search space). For this search process, each particle's velocity is represented by a matrix of size $n \times n$,

$$v_i = \begin{bmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{bmatrix} \tag{4}$$

where p represents the probability of selection of a facility for a particular location, i.e., each element of the ith row ($p_{ij}, j = 1, 2 \ldots n$) represents a selection probability of a jth facility for the ith location. During each iteration, the velocity is updated based on the particle's performance and the learning. In the proposed framework, the learning process is completed by identifying beneficial assignment of the facility to a particular location, i.e., x_i identifies the facility assignments for each location through learning from different exemplars (personal-best, local-best, global-best). Since these facility assignments represent more promising solutions, their selection probability is increased during the update process. To identify the beneficial elements from the learning exemplar, a set difference operator, *setdiff*, is used. For two positions A and B the difference operator is defined as:

$$\text{setdiff}\,(A, B) = \{e \in A \text{ and } e \notin A \cap B\} \tag{5}$$

which is similar to set subtraction operation defined in Clerc (2004), Correa et al. (2006), Neethling and Engelbrecht (2006), Veenhuis (2008), Chen et al. (2010), and Langeveld and Engelbrecht (2011).

The velocity update process is completed in four steps. In the first step, beneficial facility assignments are found from the learning exemplar(s). Based on this identification, corresponding probabilities are updated in the next step. In the third step, the particle's performance relative to the swarm is evaluated and compared to its previous fitness. In the final step, based on the particle's own performance, probabilities corresponding to its position are updated.

To understand the velocity update process, consider a QAP with four instances ($n = 4$), with a given particle $x_i = \{4\ 3\ 2\ 1\}$ whose personal-best $pbest_i$ is $\{3\ 4\ 2\ 1\}$ and global-best $gbest$ is $\{2\ 3\ 4\ 1\}$. To complete the learning from exemplars, the *setdiff* operator is applied as per (5),

$$L_1 = \text{setdiff}\,(pbest_i, x_i) = \text{setdiff}\,(\{3\ 4\ 2\ 1\},\ \{4\ 3\ 2\ 1\}) = \{3\ 4\ \Phi\ \Phi\}$$
$$\text{similarly, } L_2 = \text{setdiff}\,(gbest, x_i) = \{2\ \Phi\ 4\ \Phi\}$$

where, Φ is a null matrix, L_1 represents the cognitive learning set and L_2 represents the social learning set.

In the second step of the learning process, based on identification of good assignments, the selection probability is increased. For the example considered here, the cognitive learning set, L_1, indicates that assigning the third and the fourth facilities to the first and second locations, respectively, is beneficial. Similarly, the social learning set, L_2, suggests that assigning the second and fourth facilities to the first and third locations, respectively, is likely to give good results.

To incorporate this learning, first, the permutation vectors of the position (x_i) and learning sets (L_1, L_2) are converted to a permutation matrix of dimension $n \times n$ based on the following function:

$$\lambda_{ij}(x) = \begin{cases} 1, & \text{if } x_i = j \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Based on (6), the permutation matrices of the position and the learning sets can be evaluated as follows:

$$\begin{aligned} \lambda(x_i) &= \lambda(\{4\ 3\ 2\ 1\}) = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \\ \lambda(L_1) &= \lambda(\{3\ 4\ \Phi\ \Phi\}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\ \lambda(L_2) &= \lambda(\{2\ \Phi\ 4\ \Phi\}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

and velocity of the ith-particle, v_i, is updated as follows:

$$\begin{aligned} v_i &= v_i + (c_1 r_1 \times \lambda(L_1)) + (c_2 r_2 \times \lambda(L_2)) \\ &= \begin{bmatrix} p_{11} & p_{12} + c_2 r_2 & p_{13} + c_1 r_1 & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} + c_1 r_1 \\ p_{31} & p_{32} & p_{33} & p_{34} + c_2 r_2 \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix} \end{aligned} \tag{7}$$

As seen in (7), based on the learning sets, probability of the beneficial assignments is increased; p_{13} and p_{24} (from cognitive learning) and p_{12}, p_{34} (from social learning) are increased by learning coefficients $c_1 r_1$ and $c_2 r_2$, respectively.

In the third step, the particle's own contribution to the velocity update is evaluated. If the particle's fitness is improved compared to the previous position, its corresponding probability is increased. Otherwise, it is decreased to encourage more exploration. The amount of increase/decrease is determined by the particle's fitness ranking among its peers (δ_i) and the inertia weight (w). For the i^{th} particle, this update, Δ_i, is given by (for a minimization problem),

```
1. Evaluate the learning sets
   L1 = setdiff (pbest_i, x_i)
   L2 = setdiff (gbest, xi)
2. Evaluate the effect of the current
   position, Evaluate $\Delta_i$ as per (8)
3. Evaluate velocity as per (10)
```

Fig. 1. Pseudo code for the velocity update of the ith particle following the proposed discretizing framework.

$$\Delta_i = \begin{cases} +(w \times \delta_i), & \text{if } \frac{f_i^{t-1}}{f_i^{t-2}} < 1 \\ -(w \times \delta_i), & \text{if } \frac{f_i^{t-1}}{f_i^{t-2}} \geq 1 \end{cases} \tag{8}$$

where, $\delta_i = 1 - (\frac{f_i^{t-1}}{max(F^{t-1})})$, $f_i^{(t-1)}$ and $f_i^{(t-2)}$ are the fitness of the ith particle in the previous two iterations, and $F^{(t-1)}$ is a vector representing the entire swarm's finesses in the previous iteration.

In the final step, the probabilities corresponding to the previous position are updated based on the following rule:

$$v_i = v_i + (\Delta_i \times \lambda(x_i)) = \begin{bmatrix} p_{11} & p_{12}+c_2r_2 & p_{13}+c_1r_1 & p_{14}+\Delta_i \\ p_{21} & p_{22} & p_{23}+\Delta_i & p_{24}+c_1r_1 \\ p_{31} & p_{32}+\Delta_i & p_{33} & p_{34}+c_2r_2 \\ p_{41}+\Delta_i & p_{42} & p_{43} & p_{44} \end{bmatrix} \tag{9}$$

It is worth noting that all of these steps can be summarized in the following single velocity update rule:

$$v_i = (\Delta_i \times \lambda(x_i)) + (c_1r_1 \times \lambda(L_1)) + (c_2r_2 \times \lambda(L_2)) \tag{10}$$

It can be observed that the proposed velocity update rule is very similar to the velocity update rule (2) of the PSO algorithm; all the essential components of the original algorithm are preserved. The pseudo code for the velocity update procedure is shown in Fig. 1.

2.3. Position update

The position update procedure requires, for each *location*, the selection of *facility* from the *n* available facilities. For this purpose, we extend the Kennedy and Eberhart's initial idea of using velocity as probability in the binary version of the PSO (Kennedy & Eberhart, 1997). The proposed algorithm is built on this notion.

The updated velocity represents the corresponding particle's learning from its peers and self-experience, accumulated over the search process. The velocity given by (10) represents an accumulated experience of particle *i*. Before the new position is created, the velocity is converted into a probability matrix using the following equation:

$$nv_i = \begin{bmatrix} \frac{v_{11}}{\sum_{j=1}^n v_{1j}} & \cdots & \frac{v_{1n}}{\sum_{j=1}^n v_{1j}} \\ \vdots & \ddots & \vdots \\ \frac{v_{n1}}{\sum_{j=1}^n v_{nj}} & \cdots & \frac{v_{nn}}{\sum_{j=1}^n v_{nj}} \end{bmatrix} \tag{11}$$

The normalized velocity (11) now serves as a guide map for the creation of new position. Each row of a velocity matrix stores the probability of all *n* facilities for the selection for the location whose index is the same as that of the row's, i.e., ith row of the velocity matrix stores the selection probability of all *n* facilities for the ith location. The creation of new position requires, for each location, a selection of a single facility out of a pool of *n* facilities, i.e., selection of a single facility from each row of a velocity matrix.

```
1. Evaluate normalized velocity, nv_i, as
   per (11)
2. Set new position, POS = {Φ}
   // Assign elements with highest
   probability to POS //
3. for j = 1 to n
     a. Select the facility with highest
        probability in the jth row and
        assign it to the jth location of
        the new position POS(j)
4. end
5. Identify the unique elements in 'POS'
   and set repetitions to null Φ
   // Assign null elements in POS with
   elements form previous position //
6. if POS has null elements
        Old POS = {e|e ∈ x_i and e ∉ POS}
     a. Identify the null elements in POS
     b. Assign elements from Old POS to
        respective dimension
7. end
   // Assign null elements in POS with
   elements form universal set E //
8. if POS has null elements
        E' = {e|e ∈ E and e ∉ POS}
     a. Identify the null elements in POS
     b. Assign elements from E' to
        respective dimension
9. end
10. Update the position, x_i = POS^T
```

Fig. 2. Pseudo code for the position update following the proposed discretizing framework.

For the QAP, it is essential to make sure that newly constructed position represents a valid solution (permutation). With the aforementioned approach to create the new position, it is possible that some of the locations have repeated assignments, i.e., the same facility is assigned to more than one location, which represents a non-feasible solution (permutation) to the QAP. To ensure valid permutation, it is crucial to make sure that the assignment to each location is unique. To overcome this problem, each particle's new position is created in three steps. In each step, a different source set is used for the selection of a facility for a particular location. The first source set is velocity, the second is the particle's current position and the last is the universal set, $E = \{1, 2,\ldots, n\}$ (for a QAP with *n* instances), containing all the facilities. This approach is similar to the position update procedure used in Chen et al. (2010). The pseudo code for the position update is depicted in Fig. 2.

To understand the position update procedure, assume that the velocity v_i of the particle x_i after velocity update procedure (as a result of (10)) is,

$$v_i = \begin{bmatrix} 0.31 & 1.38 & 0.72 & 0.70 \\ 0.95 & 0.76 & 1.07 & 1.27 \\ 0.03 & 1.42 & 0.64 & 2.59 \\ 1.06 & 0.18 & 0.70 & 0.65 \end{bmatrix}$$

The first step is to normalize the velocity as per (11), to give probabilities for selection of each element

$$nv_i = \begin{bmatrix} 0.10 & 1.44 & 0.23 & 0.22 \\ 0.23 & 0.19 & 0.26 & 0.31 \\ 0.01 & .030 & 0.14 & 0.55 \\ 0.41 & 0.07 & 0.27 & 0.25 \end{bmatrix}$$

The normalized velocity, nv_i, serves as a first source to create the new position for the i^{th} particle. To generate the new position, *POS*, an element with the highest probability is selected from each row: $POS = \{2\ 4\ 4\ 1\}^T$.

In the next step, feasibility is checked for valid permutations, and only unique assignments are retained from the constructed position, *POS*. As the fourth facility is assigned to both second and third locations, null Φ is assigned to the third location: $POS = \{2\ 4\ \phi\ 1\}^T$.

In the next step, locations with null assignments are identified. To complete the construction of the new position, the identified locations are assigned with the respective facilities from their current position (the second source set). To assign facility to the third location in *POS*, particle's current position, $x_i = \{4\ 3\ 2\ 1\}$, is checked. However, using assignment from x_i for the third location (2nd facility) will not result in a feasible solution ($\{2\ 4\ 2\ 1\}^T$) and hence it is not used.

Since the third location is still to be assigned, the universal set, E, is used as the source set, $E' = \{e | e\ \in E \text{ and } e \notin POS\}$; hence, the new position will be: $POS = \{2\ 4\ 3\ 1\}^T$ and, finally, the new position of the i^{th} particle is updated as: $x_i = POS^T = \{2\ 4\ 3\ 1\}$.

It is possible to generate a feasible solution from velocity only, by using a scheme similar to the decoding scheme used to generate the solution from the membership mapping in Pang et al. (2004) and Liu et al. (2007a, b). However, this will lead to inclusion of elements with lower probability into the new position and will reduce the solution quality.

It is pertinent to note that a likelihood matrix (of $n \times n$ size for facility assignment), somewhat similar to the proposed velocity representation, has been used before in multi-parent crossover (MPX) for Hybrid Genetic Algorithm in Misevicius and Kilda (2005) and Misevicius and Rubliauskas (2005). However, its purpose is completely different. In MPX, the likelihood matrix represents a frequency of facility assignment to a particular location among all the parents taking part in crossover. In essence, it is a common guide map derived from the participating parents to generate a new solution or children and does not have any memory. On the contrary, the velocity matrix, proposed here, represents the learning accumulated by an individual particle over the search iterations. The velocity matrix is exclusive to each particle and it is updated based on the corresponding particle's learning exemplar and its own personal experience.

2.4. Refresh gap

PSO suffers from premature convergence due to the lack of diversity in the later stages of the search process (Clerc & Kennedy, 2002). To solve NP hard problems such as the QAP, it is necessary to maintain the swarm's diversity throughout the search process. One of the possible alternatives is to use the "restart", whenever stagnation is detected (James et al., 2009). In the context of the proposed framework to discretize PSO, *restart* can be interpreted as reinitializing the particle's velocity. Since the velocity is the main source for the creation of the new position in the proposed approach, reinitializing the velocity leads to reinvigoration in the swarm's exploration capability. However, the swarm's experience is retained in the process; only velocity is reinitialized and personal and global best are not altered. Fig. 3 shows the pseudo code of the generic PSO (GPSO) algorithm (Kennedy & Eberhart, 1995; Yuhui & Eberhart, 1999) to solve the assignment problems, based on the proposed framework.

```
1.  Set the search parameters, c_1, c_2, w_0, w_f and
    RG (refresh gap) and stagnation count
2.  Initialize the swarm of ps particles,
    X = {x_1, ....x_ps}, with random permutations
3.  Initialize the velocity  of each particle as
    (n×n) matrix of uniformly distributed random
    number between [0,1]
4.  Evaluate the fitness of the swarm, personal
    and global best
5.  for m = 1 to Max Epochs
    // Refresh Gap, Reinitialize velocity
    if Stagnation is detected //
6.        if cnt ≥ RG
           a. Reinitialize the velocity  of each
              particle as (n×n) matrix of
              uniformly distributed random number
              between [0,1]
           b. Set stagnation count, cnt to zero
          end // end of refresh gap loop
    // Velocity and Position Update//
7.        Evaluate the inertia weight,
            $w(m) = w_0 - \frac{(w_0 - w_f) \times m}{Max\ Epochs}$
8.        for i= 1 to ps
            a.  Update the velocity of the $i^{th}$
                particle as per fig. 1
            b.  Update the position of the $i^{th}$
                particle as per fig. 2
9.        end // end of position update loop
    // Fitness Evaluation //
10.       Store the old fitness of the swarm, F
11.       Evaluate the swarm's fitness as per
          new position
12.       Evaluate personal and global best
    // Check for stagnation //
13.       if  global best does not improve
               Increase the stagnation count,
                 cnt = cnt+1
14.       end  // end of check for stagnation
15. end // end of iterations
```

Fig. 3. Pseudo code for the generic PSO (GPSO) following the proposed discretizing framework. GPSO uses decreasing inertia weight with iterations, w_0 and w_f are initial and final inertia weight, respectively.

To assess the contribution of each source set (velocity, particle's current position, and universal set) in the new position, the basic PSO algorithm based on the pseudo code given in Fig. 3 is executed with a

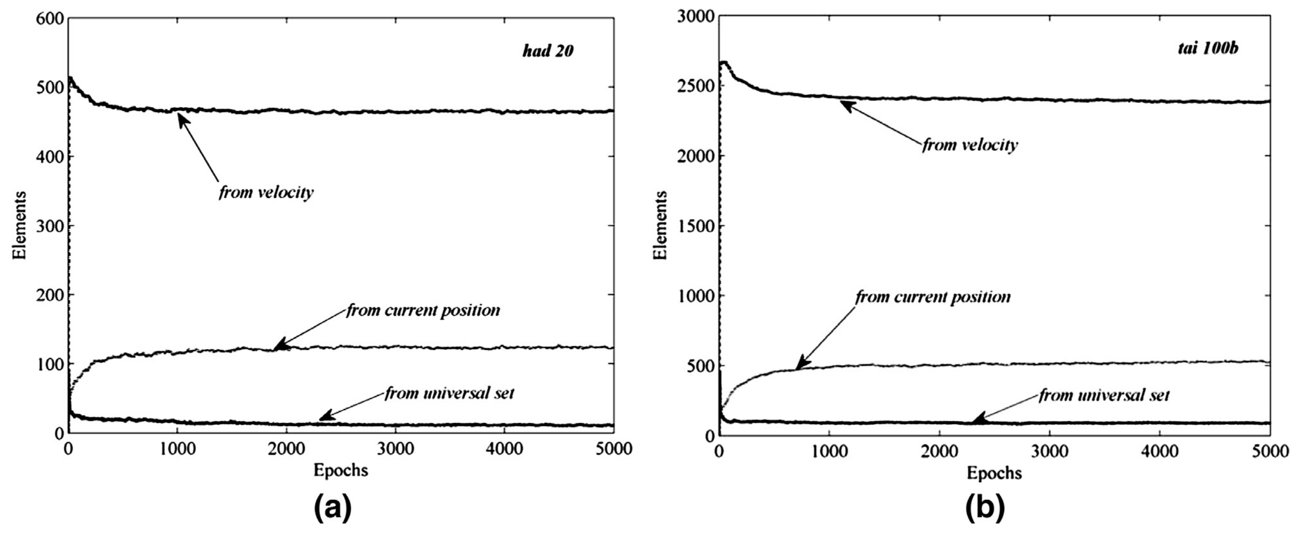


Fig. 4. Results of the experiment to identify the source of the elements in the new position with (a) *had20* and (b) *tai100b*. The results are averaged over 10 runs with maximum iteration set to 5000. The swarm size, *ps*, was set to 30. In each iteration for *had20* total number of elements will be 600 ($ps \times n$) and the same for *tai100b* will be 3000 ($ps \times n$).

provision to record each element's source, i.e., identifying the source of each assigned element. Fig. 4 shows the results of this experiment for *had20* and *tai100b* (Burkard et al., 1997) assignment problems (covered in more detail in Section 4). For each iteration, the total number of elements will be ($ps \times n$), where *ps* is the swarm size. Fig. 4(a) and (b) show the number of elements from each source set for each iteration. As seen in the results, most of the elements come from the velocity, followed by the particle's current position, and very few from the universal set. Even at very late stages of the search process with $ps = 30$, for *had20*, 466 out of 600 terms (77%) position updates come from velocity. Likewise, for *tai100b*, 2385 out of 3000 terms (79%) are updated through velocity. This is much better than the earlier set-based approach proposed in Chen et al. (2010).

3. Discretization of common PSO variants

Balancing between exploration and exploitation is essential for the robust performance of any search algorithm (Hafiz & Abdennour, 2013). To solve the premature convergence problem for the generic PSO, many of improvements have been proposed (Ratnaweera, Halgamuge, & Watson, 2004; Mendes et al., 2004; Parsopoulos et al., 2004; Liang & Suganthan, 2005; Liang et al., 2006). Most of these approaches can be classified on the basis of parameter control (Ratnaweera et al., 2004), changes in the information sharing neighborhood topology (Yuhui & Eberhart, 1999; Kennedy & Mendes, 2002; Mendes, Kennedy, & Neves, 2004), and learning exemplars (Parsopoulos et al., 2004; Liang et al., 2006). Since the proposed approach to discretize the PSO retains the essence of the original algorithm, it is also possible to easily update and discretize the PSO variants. In general, it is possible to discretize any PSO variant following the proposed framework if there is no Euclidian distance involved in the learning exemplar.

In this section, based on the proposed approach, several of the PSO variants are discretized to solve the QAP. Since it is not possible to accommodate all important variants of PSO, we have selected representative variants from each important area of improvement, e.g., improvement based on updated information sharing and improvement based on modified learning exemplar. The selected variants include, two canonical versions generic PSO (GPSO; Yuhui & Eberhart, 1999) and local best PSO (LPSO; Kennedy & Mendes, 2002), Comprehensive Learning PSO (CLPSO; Liang & Suganthan, 2005), Unified PSO (UPSO; Parsopoulos et al., 2004), Dynamic Multi Swarm PSO (DMS-PSO; Liang et al., 2006), and Fully Informed Particle Swarm (FIPS; Mendes et al., 2004).

3.1. Variants based on neighborhood

The essence of the PSO is the learning through information sharing with the peers. The manner in which the information is shared has a profound effect on the algorithm's performance. The earlier versions of the PSO used the global star approach (GPSO) (Kennedy & Eberhart, 1995; Yuhui & Eberhart, 1999) in which the information from the best performing particle is shared immediately with all the peers. The algorithm's convergence is much faster with this approach; though it increases the chances of being trapped in local minima (Clerc & Kennedy, 2002). As a remedy, the information sharing was limited to a small group of particles, neighbors (Kennedy & Mendes, 2002). The pseudo code for the discrete GPSO, based on the proposed approach, is as given in Fig. 3.

3.1.1. Local best PSO (LPSO)

The neighborhood best or the local best approach (LPSO) slows down the information propagation within the swarm, which allows the particles to explore more on their own and reduces the probability to be trapped in the local minima. Kennedy and Mendes experimented with the various neighborhood topologies (Kennedy & Mendes, 2002). In this work, we have selected the Ring topology for the local version of the algorithm.

The velocity update process for the discrete version of LPSO is almost the same as GPSO. The only difference is the change in the learning set; the particle learns from the best particle in its neighborhood instead of the best particle in the swarm. For the LPSO, the first step in the velocity update procedure of Fig. 1 is modified as follows:

Evaluate the learning sets:

- Locate the particle with the best experience in the neighborhood, set its personal best as a social learning exemplar, $lbest_i$

$$L_1 = setdiff\,(pbest_i, x_i)$$
$$L_2 = setdiff\,(lbest_i, x_i)$$

The position update procedure is the same as that for discrete GPSO, shown in Fig. 2.

```
1.  Initialize the swarm and the search parameters
2.  Divide the swarm in $k$ sub-swarms
3.  Evaluate the fitness of the swarm, personal best
    and local best in each sub-swarm
4.  for $m$ = 1 to (0.90 × Max Epochs)
5.        if mod(m, R)==0
              Regroup and redistribute particles in $k$
              swarms
6.        end  // end of if
7.  Update Each Swarm using local version of PSO
8.  Regroup all particles in one swarm
9.  for $m$ = 1 to (0.10 × Max Epochs)
         Update swarm using global best version of the
         PSO
10. end // end of iterations
```

Fig. 5. Pseudo code for the DMS-PSO following the proposed discretizing framework. '*R*' is regrouping period.

3.1.2. *Dynamic multi swarm PSO (DMS-PSO)*

The DMS-PSO (Liang et al., 2006) employs an innovative approach to the neighborhood topology. In the DMS-PSO, the swarm is classified into several sub-swarms. Each sub-swarm uses its own local best as a learning exemplar. To avoid the local minima, after a pre-defined number of iterations, *R*, the swarm is regrouped and, again, randomly redistributed into sub-swarms. Through this process, the information obtained by each sub-swarm is shared with the other sub-swarms.

The velocity update process of the discrete version of the DMS-PSO is similar to Fig. 1, except for the social learning; here, the particle learns from the sub-swarm's best particle's experience. For the DMSPSO, the first step in the velocity update procedure of the Fig. 1 is modified as follows:

Evaluate the learning sets

- Locate the particle with the best experience in the sub-swarm which includes the i^{th} particle, set it's personal best as a social learning exemplar, $\mathbf{group\_best_k}$

$L_1 = setdiff\,(pbest_i, x_i)$
$L_2 = setdiff\,(group\_best_k, x_i)$

The position update process is similar to that in Fig. 2. The sub-swarms are regrouped into one swarm after 90% of the total iterations. For the rest 10% of the iterations, the search is completed through the global best version of the PSO. The pseudo code of the DMS-PSO is shown in Fig. 5.

3.2. *Variants based on learning exemplar*

Apart from the information sharing, the other important feature that affects the PSO's search behavior is the information source, i.e., the learning exemplar. The original algorithm uses the particle's own personal best position and the swarm's best position as a learning exemplar. Quite a few variants, based on different learning exemplars, have been proposed to overcome the PSO's main problem, i.e., lack of diversity at the end of the search process. We have selected three popular variants as a representative in this category: the UPSO, the CLPSO and the FIPS.

3.2.1. *Unified Particle Swarm Optimization (UPSO)*

The UPSO, as the name suggests, unifies both the local and global versions of the PSO by unification factor, *u*. The velocity update equation of UPSO is be given by Parsopoulos et al. (2004),

$$v_i = (u\, v_{gi}) + (r_n\,(1-u)\, v_{li}) \tag{12}$$

where, r_n is a random number with normal distribution,

$$v_{gi} = w\, v_i + c_1 r_1 \times (pbest_i - x_i) + c_2 r_2 \times (gbest - x_i) \tag{13}$$

and

$$v_{li} = w\, v_i + c_1 r_1 \times (pbest_i - x_i) + c_2 r_2 \times (lbest_i - x_i) \tag{14}$$

Eq. (12) can further be simplified to:

$$v_i = w_u\, v_i + R_1 \times (pbest_i - x_i) + R_2 \times (gbest - x_i) + R_3 \times (lbest_i - x_i) \tag{15}$$

with,

$$w_u = w\, r_n u + (1-u)\, w \tag{16}$$

$$R_1 = c_1 r_1\, r_n u + (1-u)\, c_1 r_1 \tag{17}$$

$$R_2 = c_2 r_2\, r_n u \tag{18}$$

$$R_3 = (1-u)\, c_2 r_2 \tag{19}$$

The expanded version of the velocity update rule reveals that a particle learns from two exemplars (global and local best) other than itself; the probability of learning is determined by R_2 and R_3, respectively. The velocity update rule for discrete UPSO is given by (20) and the pseudo code for the velocity update process based on this rule is depicted in Fig. 6.

$$v_i = (\Delta_i \times \lambda(x_i)) + (R_1 \times \lambda(L_1)) + (R_2 \times \lambda(L_2)) + (R_3 \times \lambda(L_3)) \tag{20}$$

where, $L_1 = setdiff\,(pbest_i, x_i)$, $L_2 = setdiff\,(lbest_i, x_i)$ and $L_3 = setdiff\,(gbest, x_i)$ are learning sets; Δ_i is evaluated as follows:

$$\Delta_i = \begin{cases} +\,(w_u \times \delta_i), & \text{if } \frac{f_i^{t-1}}{f_i^{t-2}} < 1 \\ -\,(w_u \times \delta_i), & \text{if } \frac{f_i^{t-1}}{f_i^{t-2}} \geq 1 \end{cases} \tag{21}$$

```
1. Locate the particle with the best experience in the
   neighborhood, set it's personal best as a social learning
   exemplar, $lbest_i$
2. Evaluate the learning sets,
   $L_1$ = setdiff ($pbest_i$, $x_i$)
   $L_2$ = setdiff ($lbest_i$, $x_i$)
   $L_3$ = setdiff ($gbest$, $x_i$)
3. Evaluate $w_u$,$R_1$,$R_2$ and $R_3$ as per (16)-(19)
4. Evaluate the effect of the current position,$\Delta_i$ as per (21)
   // update the velocity matrix //
5. for j = 1 to n
           Evaluate velocity as per (20)
6. End
```

Fig. 6. Pseudo code for the velocity update of the ith particle following the proposed discretizing framework for UPSO. R_1, R_2 and R_3 are evaluated as per (15)–(17).

```
1. Evaluate the learning sets,
   $L_1$ = setdiff ($pbest_{fi}$, $x_i$)
2. Evaluate the effect of the current
   position, $\Delta_i$ as per (8)
   // update the velocity matrix //
3. for j = 1 to n
     Evaluate velocity as per (23)
4. End
```

Fig. 7. Pseudo code for the velocity update of the i^{th} particle following the proposed discretizing framework for CLPSO.

3.2.2. Comprehensive Learning PSO (CLPSO)

In the comprehensive learning strategy proposed in Liang and Suganthan (2005), each particle learns from only one exemplar. Each dimension of the learning exemplar, $pbest_f$, can learn from its own personal best, or that of any other randomly selected particle, based on the learning probability P_c. The learning exemplar for the particle is regenerated if it does not improve for a predefined number of generations, or learning gap, LG. The velocity update rule for CLPSO is given by the following equation:

$$v_i = w\,v_i + cr \times (pbest_{fi} - x_i) \tag{22}$$

and the discrete version by,

$$v_i = \ (\Delta_i \ \times \lambda(x_i)) + \ (cr \times \lambda(L_1)) \tag{23}$$

with, $L_1 = setdiff\,(pbest_{fi}, x_i)$. Based on (23), the velocity update process for the discrete version of the algorithm is given in Fig. 7; again, the position update procedure is the same as shown in Fig. 2. The pseudo code for the discrete CLPSO is depicted in Fig. 8.

3.2.3. Fully Informed Particle Swarm (FIPS)

In FIPS, the particle learns from all the neighbors rather than a single best neighbor. The objective is to use the information from all the neighbors to update the particle's velocity. FIPS uses the following rule (Mendes et al., 2004) for the velocity update:

$$v_i = \chi\,(v_i + \varphi \times (p_m - x_i)) \tag{24}$$

$$p_m = \frac{\sum_k^N W(k)\varphi_k \times p_k}{\sum_k^N W(k)\varphi_k} \tag{25}$$

and $\varphi_k = \frac{\varphi_{\max}}{N}$, for a neighborhood with N neighbors. Several different versions of FIPS were proposed in Mendes et al. (2004), based on different ways to evaluate the function "W" in (25). In this work, FIPS with a ring topology is selected; as it performed better than other versions. As revealed by (25), all neighbors contribute to create the learning exemplar.

The main inspiration behind FIPS is to utilize the information from all the neighbors and keep the particle *fully informed*. It is clear that (25) cannot be used directly for the combinatorial optimization. Several approaches were attempted to obtain learning exemplars suitable for combinatorial optimization; which include rank order value (ROV; Liu et al., 2007), logical operation, and probability-based selection. In ROV, as the name suggests, the result of (25) is a rank (based on the actual value) to generate valid permutations. In logical-operation approach, a combination of AND and OR operations is used to derive the exemplar from the neighbors. In the probability-based selection, each element of the neighbors is assigned a probability, randomly drawn from the interval [0,1]. Each element of the exemplar is selected from one of the neighbors based on the assigned probability.

Apart from these three approaches, a simplified approach was used to solve the problem of learning exemplar; since the core contribution of the FIPS is to learn from all the neighbors, the nature of the FIPS will be retained even when the particle learns from personal bests of both the neighbors instead of a single exemplar derived from them. To confirm the hypothesis, an experiment was carried out on eight problem instances selected from Table 1 (covered in more detail in the following section). Table 2 lists the results of FIPS, discretized using the aforementioned approaches to create the learning exemplar. Both ROV-based and logical-operation based approaches performed poorly. The performance of the probability-based selection was not satisfactory as well. The best overall performance is achieved with the simplified approach to derive the learning exemplar.

Note that the original idea of the FIPS is retained; all neighbors contribute to the probability update procedure, and the overemphasis on a single neighbor is avoided. As ring topology is used, the particle learns from two learning sets, the best experiences of two neighbors; resulting discretized velocity rule is given by (26). The velocity update process for the FIPS based on this discretizing approach is shown in Fig. 9.

$$v_i = \ (\Delta_i \ \times \lambda(x_i)) + \ (\varphi_k \ \times \ \lambda(L_1)) + \ (\varphi_k \ \times \ \lambda(L_2)) \tag{26}$$

where, $L_1 = setdiff\,(pbest_{i\_n1}, x_i)$ and $L_2 = setdiff\,(pbest_{i\_n2}, x_i)$ are learning sets. $pbest_{i\_n1}$ and $pbest_{i\_n2}$ are personal bests of neighbors 1 and 2 of the i^{th} particle, respectively.

```
1.  Initialize the swarm and the search parameters
2.  for m = 1 to Max Epochs
3.      for i= 1 to ps
4.        if learning flag_i ≥ LG
             a. Regenerate the learning exemplar for the i^th
                particle
             b. Set learning flag of the i^th particle to zero
5.        end  // end of learning gap loop
6.        Update the velocity of the i^th particle as per fig. 7
7.        Update the position of the i^th particle as per fig. 2
8.       end // end of position update loop
9.  end // end of iterations
```

Fig. 8. Pseudo code for the CLPSO following the proposed discretizing framework. LG is the learning gap; if particle's fitness does not improve for LG generations its learning exemplar will be regenerated.

Table 1
Benchmark problems.

Category	Benchmark	*D*	BKV
Cat-1: Randomly generated	*Had20*	20	6922
	Lipa40b	40	476581
	Rou20	20	725522
	Tai20a	20	703482
	Tai30a	30	1818146
	Tai40a	40	3139370
	Tai50a	50	4938796
	Tai60a	60	7205962
	Tai80a	80	13499184
	Tai100a	100	21052466
Cat-2: Based on grid distances	*Nug30*	30	6124
	Sko42	42	15812
	Sko49	49	23386
	Sko81	81	90998
	Sko90	90	115534
	Sko100a	100	152002
	Sko100d	100	149576
	Tho150	150	8133398
	Wil50	50	48816
Cat-3: Real life like	*Tai20b*	20	122455319
	Tai30b	30	637117113
	Tai40b	40	637250948
	Tai50b	50	458821517
	Tai60b	60	608215054
	Tai80b	80	818415043
	Tai100b	100	1185996137
	Tai150b	150	498896643
Cat-4: Real life	*Bur26a*	26	5426670
	Chr15a	15	9896
	Chr25a	25	3796
	Els19	19	17212548
	Esc64a	64	116
	Kra30a	30	88900
	Kra30b	30	91420
	Ste36a	36	9526

4. Results and comparisons

4.1. Benchmarks

For the evaluation of the proposed approach, at total of 35 most commonly used problem instances are selected from the QAPLIB online repository (Burkard et al., 1997). The problem instances are selected to include cases from each subcategory, i.e., randomly generated (1–10), based on the grid distances (11–19), real-life like (20–27) and real life problems (29–35). Further, for each sub-category of instances, both low and high dimensions are selected, to see the effect of dimensionality on the algorithm performance. Table 1 shows the selected benchmarks, their category, and respective best-known values (BKV) so far.

4.2. Search parameters

For each PSO variant, a swarm of 30 particles is used. The other search parameters are set as per the suggestion in the respective works (Yuhui & Eberhart, 1999; Kennedy & Mendes, 2002; Liang et al., 2006; Parsopoulos et al., 2004; Liang & Suganthan, 2005; Mendes et al., 2004). The search parameters of all the variants are summarized in Table 3.

To counter the stagnation, a concept of refresh gap was introduced in the proposed framework and the additional parameter associated with it needs to be calibrated. For this purpose, we tried different values of refresh gap for each PSO variant on the selected set of the QAP benchmarks. To obtain the unbiased performance, two benchmarks were selected from each category; one with lower dimension and the other with a higher dimension. The refresh gap was set at 5, 10, 25, 50 and 100% of iterations and the cumulative best error (obtained in 10 runs) was recorded. Fig. 10 shows the results of this experiment, the refresh gap has direct influence on the results; a lower refresh gap allows for more restarts, which leads to better performance, for all the PSO variants. In addition, as expected, it can be observed that the performance of all PSO variants was inadequate, when the refresh gap was set at 100% of iterations, i.e., no restart is provided. It was found that, refresh gaps in the range of 5–10% of iteration provide adequate results.

4.3. Evaluation of PSO variants

4.3.1. Comparative evaluation

The discretized version of the basic PSO (GPSO) and its five variants (LPSO, DMSPSO, UPSO, CLPSO, FIPS) as per the proposed framework were applied to the test suite of 35 benchmarks given in Table 1. The algorithms were implemented using MATLAB (Release 2013a) on a computer with Intel i7 processor and 8GB RAM. The maximum number of function evaluations (Max FEs) is set to 500,000 for each problem. Tables 4–7 show the results obtained for each category of the problems after 10 runs of all algorithm variants.

For each category, the algorithms are ranked according to their average percentage best error (PBE) compared to the best-known value (BKV). In addition, if the best value is achieved more than once during the 10 runs, it is indicated by the value in parentheses along with the PBE.

Table 2
Results obtained with different approaches to create Learning Exemplar for FIPS with 500,000 FES averaged over 10 runs.

Problem	ROV based		Logical operation based		Probability based		Simplified	
	PBE	Dev.	PBE	Dev.	PBE	Dev.	PBE	Dev.
had20	3.52	0.24	0.72	0.52	**0 (2)**	0.31	0.06	0.25
ta100a	11.56	0.14	9.04	0.74	5.02	0.60	**4.87**	0.26
nug30	16.26	0.66	7.90	1.34	0.95	0.76	**0.75**	0.82
tho150	16.49	0.18	13.59	0.60	7.23	0.34	**4.35**	0.72
bur26a	1.53	0.22	0.30	0.23	**0.00**	0.08	0.02 (3)	0.10
kra30b	22.36	1.02	9.57	2.20	**0.32**	1.30	0.84	1.28
tai20b	9.12	1.27	0.45	0.60	**0.00**	4.20	**0.00**	0.52
ta150b	23.20	0.34	18.34	1.75	5.78	1.65	**3.82**	0.88
Average PBE	13.01	-	7.49	-	2.76	-	2.10	-

The best result obtained for each instance is shown in bold face.

```
1.  Locate the neighbors personal best
    experiences and set them as social learning
    exemplars, $pbest_{i\_n1}$ and $pbest_{i\_n2}$
2.  Evaluate the learning sets,
    $L_1$ = setdiff ($pbest_{i\_n1}$, $x_i$)
    $L_2$ = setdiff ($pbest_{i\_n2}$, $x_i$)
3.  Evaluate the effect of the current position, $\Delta_i$
    as per (8)
4.  for j = 1 to n // update the velocity matrix
            Evaluate velocity as per (26)
5.  End
```

Fig. 9. Pseudo code for the velocity update of the *i*th particle following the proposed discretizing framework for FIPS.

Table 3
Search parameters of PSO variants.

Variant	Inertia weight	Parameters
GPSO (Kennedy & Eberhart, 1995)	*Decreasing inertia with iteration from $w_0 = 0.9$ to $w_f = 0.4$*	$c_1 = 2, c_2 = 2$
LPSO (Kennedy & Mendes, 2002)	*Decreasing inertia with iteration from $w_0 = 0.9$ to $w_f = 0.4$*	$c_1 = 2, c_2 = 2$
DMS-PSO (Liang & Suganthan, 2005)	*Decreasing inertia with iteration from $w_0 = 0.9$ to $w_f = 0.2$*	$c_1 = 2, c_2 = 2$, *sub swarm size, $m = 3$ regrouping period, $R = 5$*
UPSO (Parsopoulos & Vrahatis, 2004)	*Constant inertia $w = 0.729$*	$c_1 = 1.494, c_2 = 1.494$, *unification factor, $u = 0.1$*
CLPSO (Liang et al., 2006)	*Decreasing inertia with iteration from $w_0 = 0.9$ to $w_f = 0.7$*	$c_1 = 1.494$, *learning gap, LG = 5*
FIPS (Mendes et al., 2004)	*Constriction factor, $w = 0.7298$*	-

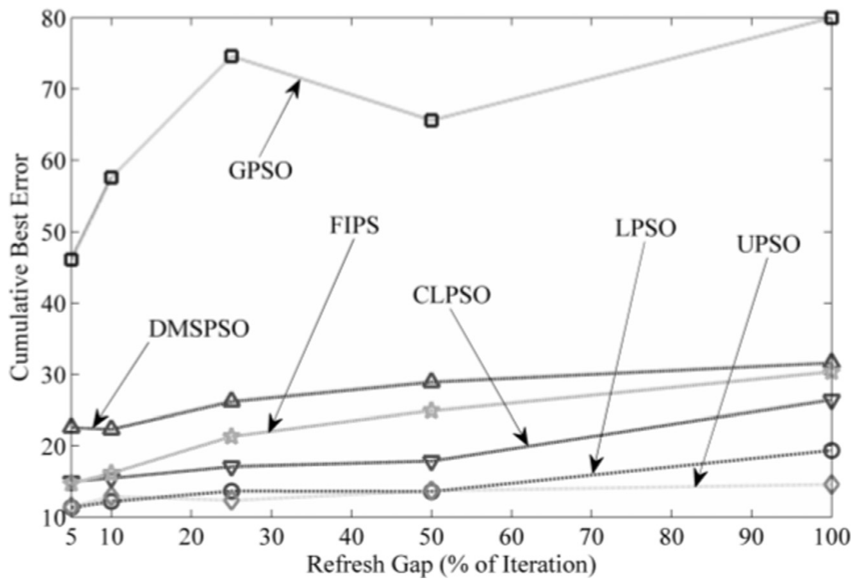


Fig. 10. Variation in cumulative PBE with change in refresh gap, RG.

Table 4
Results obtained for Category 1 problems with 500,000 FEs averaged over 10 runs.

Problem		GPSO	LPSO	DMSPSO	UPSO	CLPSO	FIPS
had20	PBE	0.87	0.12	0.55	0.06	**0 (2)**	0.06
	Dev.	0.73	0.45	0.52	0.24	0.29	0.25
	Time (minutes)	4.87	4.91	9.01	6.20	5.08	5.41
lipa40b	PBE	20.09	2.73	19.35	**0 (2)**	**0 (2)**	**0 (4)**
	Dev.	0.91	4.92	0.49	8.07	8.04	8.84
	Time (minutes)	5.59	6.94	10.37	9.03	5.86	7.56
rou20	PBE	3.80	1.64	2.28	**0.56**	2.05	1.83
	Dev.	1.27	0.75	1.83	1.42	0.77	0.77
	Time (minutes)	4.68	5.14	8.42	6.57	5.80	5.63
tai20a	PBE	5.68	2.65	3.62	2.80	2.33	**1.72**
	Dev.	1.75	0.67	1.36	0.88	1.08	1.27
	Time (minutes)	4.93	5.60	8.49	6.32	5.91	5.97
tai30a	PBE	8.05	2.56	4.69	**2.50**	2.72	2.54
	Dev.	0.57	0.69	1.01	0.61	0.72	0.45
	Time (minutes)	5.27	6.83	9.36	8.27	5.75	7.36
tai40a	PBE	7.58	3.11	4.50	3.40	4.36	**3.08**
	Dev.	0.75	0.51	1.34	0.21	0.40	0.57
	Time (minutes)	5.70	7.34	10.42	9.08	5.69	8.33
tai50a	PBE	8.05	**3.57**	5.31	4.01	4.20	4.05
	Dev.	0.77	0.35	0.99	0.28	0.70	0.46
	Time (minutes)	6.15	8.06	11.71	10.52	6.66	9.12
tai60a	PBE	8.56	**3.46**	5.81	3.79	4.84	3.84
	Dev.	0.50	0.42	0.68	0.30	0.47	0.63
	Time (minutes)	6.86	8.89	13.42	11.72	7.16	10.01
tai80a	PBE	8.11	3.90	4.64	**3.43**	5.17	4.43
	Dev.	0.40	0.24	1.17	0.29	0.46	0.37
	Time (minutes)	8.35	10.50	16.08	15.04	8.02	11.84
tai100a	PBE	7.99	3.55	3.91	**3.51**	4.43	4.52
	Dev.	0.26	0.21	1.14	0.21	0.60	0.32
	Time (minutes)	10.96	12.66	19.72	19.09	9.84	14.27
	Average PBE	7.88	2.73	5.47	2.41	3.01	2.61
	Category rank	***6***	***3***	***5***	***1***	***4***	***2***

The best result obtained for each instance is shown in bold face.

The results indicate that, as expected, all PSO variants performed better than GPSO. For the problem instances in the first two categories (Table 4 and 5) UPSO performed best with 2.41% average PBE (Category 1) and 1.37% (Category 2) closely followed by LPSO (2.73% in Category 1 and 1.41% in Category 2) and FIPS (2.61% in Category 1 and 1.89% in Category 2).

In the third category (Table 6) with real-life like problem instances, again, UPSO performed best with 1.3% average PBE; while on the real-life problems in Category 4 (Table 7) CLPSO performed better than others with average PBE of 1.90%.

The last two rows of Table 7 show the average PBE of each algorithm for all the 35 problems in the test suit and the overall ranking of the algorithms on that basis. The overall performance indicate that UPSO (1.92% average PBE) outperformed all the variants, followed by FIPS (2.41%), CLPSO (2.61%) and LPSO (2.65%). The DMSPSO was outperformed by all the variants; its performance was better than GPSO only.

It is interesting to note that the local version of PSO (LPSO) which is a very basic version of PSO performed well compared to the other complex variants; it achieved the second rank for two problem categories and achieved the fourth overall rank. The results indicate that rather than generating complex learning exemplar (as in CLPSO) or complex neighborhood topologies (as in DMSPSO, which performed rather poorly) a simple information exchange mechanism with the neighbor particle(s) (as in LPSO and FIPS) is sufficient and able to achieve better performance for the QAP.

4.3.2. Fitness landscape analysis

To have some insight into PSO's search behavior on combinatorial problems, a fitness landscape analysis (Jones and Forrest, 1995; Merz and Freisleben, 2000; Vanneschi et al., 2003) was performed. UPSO was selected for this purpose, as it has the better overall performance compared to PSO variants. Traditionally, FLA has been used to estimate the problem hardness, indicated by the fitness distance co-efficient (FDC), r, defined for a set of distances from known optimum $D = \{d_1, d_2, \ldots\ldots d_n\}$ with corresponding finesses $F = \{f_1, f_2, \ldots. f_n\}$ as (Jones and Forrest, 1995):

$$c_{FD} = \frac{1}{n} \sum_{i=1}^{n} (f_i - \bar{f})\,(d_i - \bar{d}) \tag{27}$$

$$r = \frac{c_{FD}}{\sigma_F\,\sigma_D} \tag{28}$$

where c_{FD} is the covariance of F and D, $\bar{f}, \bar{d}$, σ_F and σ_D are mean and deviation of F and D, respectively. For a minimization problem, a positive high value of FDC, r, indicates a fall in fitness as the distance to the global optimum decreases, i.e. the *higher the r, the easier is the instance to solve.*

Note that the sampling of the problem landscape, as seen in (27) and (28), has a huge impact on r. Since, in most of the cases, the sampled landscape is likely to be a small part of the problem search space, the value of r is most likely to be biased. This is the major drawback of FLA. Nevertheless, since the objective is to understand the search behavior of UPSO, FLA is quite suitable for the task; FLA should be viewed as an indicator of both: algorithm's behavior and the problem landscape.

For the test purposes, for each instance, *hamming distance* from the *best known solution* (BKS) is used as a distance measure. To sample the fitness landscape, for each instance, 10 independent runs of UPSO were executed, each run is allowed to run for 500,000

Table 5
Results obtained for Category 2 problems with 500,000 FEs averaged over 10 runs.

Problems		GPSO	LPSO	DMSPSO	UPSO	CLPSO	FIPS
nug30	PBE	8.07	1.99	4.15	1.34	**0.69**	1.63
	Dev.	1.56	0.66	1.47	0.72	0.91	0.57
	Time (minutes)	5.04	6.98	10.04	8.26	5.85	7.15
sko42	PBE	7.55	1.67	4.44	**0.77**	1.08	0.92
	Dev.	1.36	0.55	1.24	0.75	1.03	0.76
	Time (minutes)	5.66	7.65	11.05	9.45	6.15	8.34
sko49	PBE	8.46	1.13	1.80	**0.67**	2.67	1.21
	Dev.	0.78	0.56	1.64	0.55	0.74	0.82
	Time (minutes)	6.19	8.26	11.72	10.32	6.46	9.03
sko81	PBE	8.26	**1.11**	4.58	1.72	2.67	2.01
	Dev.	0.40	0.41	0.80	0.30	0.52	0.28
	Time (minutes)	8.90	10.84	16.31	15.11	8.08	12.05
sko90	PBE	8.19	1.20	5.05	**1.00**	2.89	1.85
	Dev.	0.63	0.41	0.78	0.43	0.57	0.49
	Time (minutes)	9.73	11.78	18.35	16.59	8.61	12.95
sk100a	PBE	8.47	**1.45**	4.36	1.66	2.81	2.36
	Dev.	0.25	0.48	0.81	0.28	0.47	0.65
	Time (minutes)	10.82	12.88	17.94	19.10	9.68	14.49
sk100d	PBE	7.50	**1.52**	3.76	1.68	2.56	2.30
	Dev.	0.56	0.24	0.78	0.22	0.38	0.57
	Time (minutes)	10.76	12.87	19.96	18.74	9.38	14.36
tho150	PBE	10.28	**2.13**	4.55	2.74	3.76	4.21
	Dev.	0.54	0.39	1.36	0.39	0.41	0.69
	Time (minutes)	18.47	20.53	33.94	33.48	15.46	23.15
wil50	PBE	4.42	**0.45**	1.61	0.73	0.65	0.49
	Dev.	0.55	0.36	0.87	0.32	0.58	0.37
	Time (minutes)	6.25	8.05	11.75	10.25	6.17	8.89
	Average PBE	7.91	1.41	3.81	1.37	2.20	1.89
	Category rank	***6***	***2***	***5***	***1***	***4***	***3***

The best result obtained for each instance is shown in bold face.

Table 6
Results obtained for Category 3 problems with 500,000 FEs averaged over 10 runs.

Problems		GPSO	LPSO	DMSPSO	UPSO	CLPSO	FIPS
tai20b	PBE	0.56	**0**	**0**	**0**	0.45 (3)	**0 (2)**
	Dev.	5.38	0.52	4.37	1.17	0.60	3.06
	Time (minutes)	4.74	4.84	8.75	6.79	6.92	5.15
tai30b	PBE	1.32	0.15	0.11	**0.01**	2.18	1.79
	Dev.	7.00	0.94	3.88	2.13	1.97	1.82
	Time (minutes)	5.03	6.40	10.37	7.43	6.84	6.48
tai40b	PBE	11.56	3.95	5.87	0.90	5.30	**0.36**
	Dev.	5.18	1.90	2.15	2.55	1.87	2.05
	Time (minutes)	5.59	7.62	10.55	7.83	6.68	8.21
tai50b	PBE	8.24	2.79	2.51	**1.03**	2.71	1.70
	Dev.	5.14	1.20	2.34	1.34	1.75	0.94
	Time (minutes)	6.12	8.34	12.94	7.37	6.78	9.16
tai60b	PBE	16.95	1.51	2.68	**1.03**	2.67	2.09
	Dev.	2.91	1.76	4.01	3.25	2.97	1.53
	Time (minutes)	7.10	9.22	12.77	13.45	7.34	10.15
tai80b	PBE	13.62	2.79	5.03	**2.51**	3.28	3.07
	Dev.	2.92	1.41	1.75	1.06	2.03	1.50
	Time (minutes)	8.63	10.88	15.70	8.51	8.76	12.25
tai100b	PBE	16.27	2.25	4.73	**1.53**	3.31	1.89
	Dev.	2.48	1.49	3.04	1.42	2.19	1.88
	Time (minutes)	10.93	12.95	19.48	8.55	9.74	14.47
tai150b	PBE	14.78	**3.01**	6.36	3.36	3.38	4.43
	Dev.	0.85	0.45	1.56	0.54	0.77	1.18
	Time (minutes)	18.39	20.76	31.24	33.53	15.42	23.12
	Average PBE	10.41	2.06	3.41	1.30	3.26	2.19
	Category rank	***6***	***2***	***5***	***1***	***4***	***3***

The best result obtained for each instance is shown in bold face.

Table 7
Results obtained for Category 4 problems with 500,000 FEs averaged over 10 runs.

Problems		GPSO	LPSO	DMS PSO	UPSO	CLPSO	FIPS
bur26a	PBE	0.72	0.14	0.20	**0.02**	0.15	0.02
	Dev.	0.08	0.08	0.17	0.11	0.12	0.11
	Time (minutes)	4.89	5.90	9.40	6.30	5.04	6.07
chr15a	PBE	0.83	**0**	**0 (4)**	**0**	**0**	**0 (2)**
	Dev.	10.89	11.28	16.78	4.85	6.80	8.02
	Time (min.)	5.26	5.42	10.69	7.88	6.42	5.15
chr25a	PBE	22.81	24.92	26.71	10.85	**4.79**	10.12
	Dev.	19.72	4.13	9.85	12.05	12.65	12.36
	Time (minutes)	5.55	6.79	10.31	9.07	7.05	7.14
els19	PBE	**0 (2)**	**0 (4)**	**0 (7)**	**0 (5)**	**0**	0.90
	Dev.	18.50	8.41	8.54	7.60	9.02	4.55
	Time (minutes)	4.96	5.58	10.64	10.45	6.98	5.41
esc64a	PBE	**0 (9)**	**0 (10)**	**0 (9)**	**0 (10)**	**0 (10)**	**0 (10)**
	Dev.	0.55	0.00	0.55	0.00	0.00	0.00
	Time (minutes)	7.67	10.56	18.20	11.81	10.26	10.99
kra30a	PBE	7.37	2.24	4.86	2.25	**2.13**	2.99
	Dev.	3.83	1.19	2.29	1.06	1.98	1.30
	Time (minutes)	5.09	7.13	9.85	14.75	7.55	7.91
kra30b	PBE	10.85	0.42	4.98	0.42	2.37	**0.19**
	Dev.	2.35	1.23	2.52	1.53	1.37	1.67
	Time (minutes)	5.16	7.09	9.77	19.21	7.33	7.40
ste36a	PBE	15.31	3.46	4.47	**3.21**	3.84	4.41
	Dev.	4.54	2.45	6.87	2.81	3.38	1.74
	Time (minutes)	5.55	7.49	10.26	8.70	6.72	8.22
	Average PBE	9.65	5.20	8.24	2.79	1.90	3.11
	Category rank	***6***	***4***	***5***	***2***	***1***	***3***
	Overall mean error	8.82	2.65	4.92	1.92	2.61	2.41
	Overall rank	***6***	***4***	***5***	***1***	***3***	***2***

The best result obtained for each instance is shown in bold face.

Table 8
Average distances and fitness distance co-efficient based on landscape sampling by UPSO.

CAT	Problem	*d*	*q*	*r*
Category 1: Randomly generated	*had20*	12.83	0.72	*0.30*
	lip40b	34.76	18.85	*0.85*
	rou20	16.86	4.26	*0.26*
	tai20a	18.32	6.04	0.07
	tai30a	28.75	6.24	0.05
	tai40a	39.10	7.11	0.02
	tai50a	48.81	7.40	0.05
	tai60a	58.84	7.20	0.01
	tai80a	79.18	6.66	−0.10
	tai100a	98.97	6.38	0.06
Category 2: Based on grid distances	*nug30*	27.82	6.45	*0.24*
	sko42	38.98	6.41	0.19
	sko49	47.20	5.50	0.01
	sko81	79.22	5.10	0.14
	sko90	88.57	4.97	0.10
	sko100a	98.49	4.73	0.08
	sko100d	98.01	4.94	0.13
	tho150	148.72	7.10	−0.03
	wil50	48.45	3.02	0.03
Category 3: Real life like	*tai20b*	12.43	11.13	0.16
	tai30b	29.11	16.50	−0.06
	tai40b	38.98	19.26	−0.02
	tai50b	48.86	17.07	0.07
	tai60b	59.04	13.13	−0.02
	tai80b	78.72	13.56	0.12
	tai100b	96.96	11.55	*0.35*
	tai150b	149.00	9.93	−0.02
Category 4: Real life	*bur26a*	22.82	0.48	0.08
	chr15a	10.64	96.38	*0.31*
	chr25a	22.84	106.69	0.16
	els19	10.77	45.04	*0.21*
	esc64a	62.85	24.39	−0.06
	kra30a	28.70	12.25	0.03
	kra30b	29.12	9.36	0.10
	ste36a	32.30	24.65	0.15

FEs. The test results are shown in Table 8 and Fig. 11. Table 8 shows, for each instance, the FDC(r) average distance of the samples from the BKS ($\overline{d}$) and quality of the samples (q) defined as, $q = 100 \times (\frac{\bar{f} - \text{BKV}}{\text{BKV}})$.

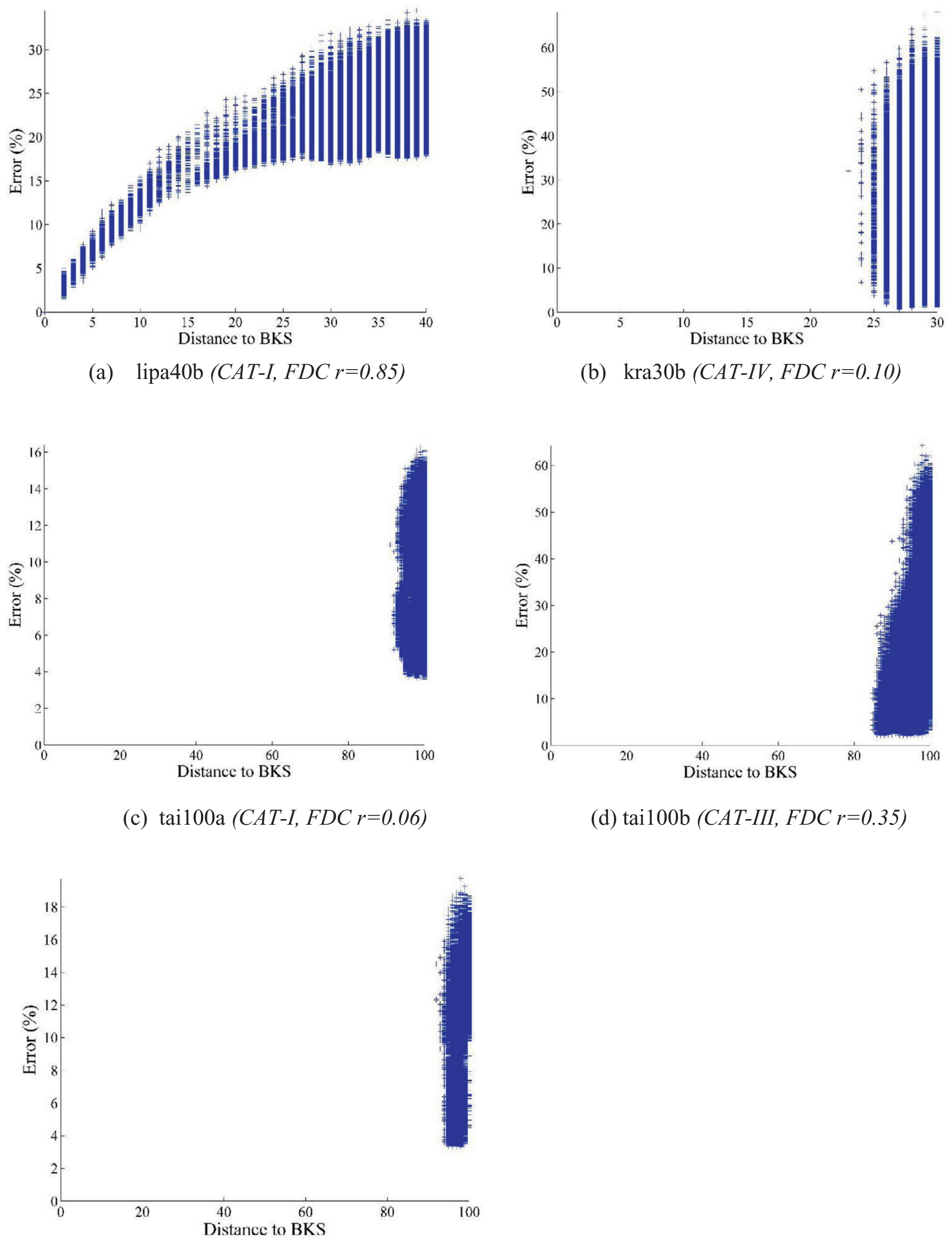
It can be observed that only eight instances (*had20, lipa40b, rou20, nug30, sko42, chr15a, els19 and tai100b*) have some sort of co-relation between fitness and distance to optimum. The only exception is *lipa40b*, which has high fitness distance correlation with $r = 0.85$, this can be further confirmed by the FDC plot in Fig. 11(a). For the other instances, the FDC r, is too small to be inferable; previous investigation (Vanneschi et al., 2003) suggests that the behavior of instance is unpredictable if the values of corresponding r falls in the range of $(-0.15, 0.15)$.

Fig. 11 shows the fitness distance co-relation plots obtained with UPSO for several instances. It can be observed that both *lipa40b* (Fig. 11(a)) and *tai100b* (Fig. 11(d)) have some fitness-distance co-relation which is exploited by the UPSO and it can provide better results for these instances. On the other hand, *tai100a* (Fig. 11(c)) and *sko100d* (Fig. 11(e)) do not display any relation between fitness and the distance and are comparatively harder to solve.

In general, as the search dimension increases, the effect of fitness–distance relationship is more pronounced. The *real-life like* (Category-3) and *real-life* (Category-4) instances are comparatively easier for UPSO to solve, as their landscape have some fitness-distance co-relation. However, randomly generated *taiXXXa* instances, due to lack of any fitness-distance relation, are proved to be harder to solve using UPSO.

4.4. Comparison with the other PSO discretization approaches

Since the main objective of this work is to design the discretization framework, it will be interesting to evaluate the performance of the proposed framework with the other attempts to discretize the PSO (Pang et al., 2004; Liu et al., 2007a, b; Helal and Abdelbar, 2014). The



(a) lipa40b *(CAT-I, FDC r=0.85)*

(b) kra30b *(CAT-IV, FDC r=0.10)*

(c) tai100a *(CAT-I, FDC r=0.06)*

(d) tai100b *(CAT-III, FDC r=0.35)*

(e) ska100d *(CAT-II, FDC r=0.13)*

Fig. 11. Fitness–distance scatter plot for different instances based on landscape sampling by UPSO. (a) *lipa40b*, (b) *kra30b*, (c) *tai100a*, (d) *tai100b*, and (e) *ska100d*.

first compared approach is the Fuzzy-PSO (Pang et al., 2004; Liu et al., 2007a,b). In this approach, position and velocity are both represented by $n \times n$ matrices and the original velocity and position update equation of the PSO are retained. The permutations are generated from the position matrix. The second approach (Helal and Abdelbar, 2014), used for comparison, relies on the set-based operations. It is pertinent to note that both of these approaches were applied to discretize the specific PSO variant and were not designed to be generic. For the purpose of this test, their position and velocity representation and update rules have been used to discretize the particular PSO variant.

For the comparative evaluation of the discretization approaches, we have selected two variants of the PSO: LPSO (due to its simplicity) and UPSO (because of its superior performance). For fair comparison, all the tests were carried out with the same FEs (*500,000 FEs*) and on the same platform (MATLAB *R2013a*, Windows PC with *Intel i7* processor and 8GB RAM). Tables 9 and 10 list the results of LPSO and UPSO based on the compared discretization approaches. Each table has three columns for both LPSO and UPSO. These columns correspond to the three discretization approaches used: *proposed approach, fuzzy approach* based on Liu et al. (2007) and *set based approach* based on Helal and Abdelbar (2014) The best result of each PSO variant on each instance is highlighted through bold-face. If the best solution is obtained more than once out of 10 runs, it is indicated by the number in parenthesis adjacent to PBE in both Tables 9 and 10.

As seen in Table 9 and 10, both the variants of PSO, LPSO and UPSO, provided better results with the proposed discretizing framework.

Table 9
Comparison with the other discretization approaches—1. Results obtained for Categories 1 and 2 problems with *500,000 FES* averaged over 10 runs.

Problem		LPSO			UPSO		
		Proposed	Based on (Helal & Abdelbar, 2014)	Based on (Peng et al., 2004)	Proposed	Based on (Peng et al., 2004)	Based on (Helal & Abdelbar, 2014)
had20	PBE	0.12	3.18	***0.09***	***0.06***	3.00	0.20
	Dev.	0.45	0.31	0.25	0.24	0.38	0.59
	Time (minutes)	8.00	6.88	5.44	10.27	10.65	9.62
lip40b	PBE	***2.73***	24.13	20.12	***0***	24.49	21.34
	Dev.	4.92	0.41	0.61	8.07	0.37	0.64
	Time (minutes)	11.78	17.33	19.94	12.98	19.60	20.42
rou20	PBE	***1.64***	9.55	1.92	***0.56***	9.67	5.79
	Dev.	0.75	0.68	1.21	1.42	0.75	0.85
	Time (minutes)	8.98	7.90	9.79	9.31	9.00	10.35
tai20a	PBE	***2.65***	8.49	2.66	**2.80**	10.80	2.90
	Dev.	0.67	1.30	1.44	0.88	0.61	1.79
	Time (minutes)	10.07	7.71	5.64	10.93	9.06	10.07
tai30a	PBE	***2.56***	11.29	4.80	**2.50**	11.77	5.48
	Dev.	0.69	0.45	0.95	0.61	0.26	0.97
	Time (minutes)	11.71	12.30	8.16	13.26	13.85	15.02
tai40a	PBE	***3.11***	12.81	6.93	***3.40***	12.79	8.02
	Dev.	0.51	0.36	0.67	0.21	0.48	0.91
	Time (minutes)	12.02	17.37	11.67	14.48	18.74	19.88
tai50a	PBE	***3.57***	13.09	8.98	**4.01**	13.68	9.57
	Dev.	0.35	0.36	0.34	0.28	0.20	0.49
	Time (minutes)	13.46	23.17	17.30	16.57	15.83	27.30
tai60a	PBE	***3.46***	13.41	8.75	***3.79***	13.01	10.39
	Dev.	0.42	0.13	0.41	0.30	0.30	0.41
	Time (minutes)	14.67	18.99	20.38	18.26	20.54	37.49
tai80a	PBE	***3.90***	12.21	9.31	***3.43***	12.04	9.08
	Dev.	0.24	0.15	0.26	0.29	0.20	0.50
	Time (minutes)	17.26	42.67	31.64	22.98	30.51	57.07
ta100a	PBE	***3.55***	11.50	8.66	***3.51***	11.61	9.01
	Dev.	0.21	0.11	0.40	0.21	0.09	0.34
	Time (minutes)	19.38	54.70	46.88	29.54	79.40	83.28
nug30	PBE	***1.99***	16.49 (2)	3.04	***1.34***	14.11	4.38
	Dev.	0.66	0.64	1.66	0.72	1.23	2.60
	Time (minutes)	10.52	10.95	12.48	13.27	16.17	13.53
sko42	PBE	***1.67***	15.75	7.08	***0.77***	15.39	8.18
	Dev.	0.55	0.55	0.92	0.75	0.57	1.42
	Time (minutes)	12.88	18.44	21.43	13.45	20.76	21.97
sko49	PBE	***1.13***	14.91	6.96	**0.67**	14.37	8.24
	Dev.	0.56	0.41	0.92	0.55	0.43	1.29
	Time (minutes)	13.64	17.64	26.15	14.75	25.32	27.68
sko81	PBE	***1.11***	13.30	8.14	***1.72***	13.72	9.00
	Dev.	0.41	0.31	0.19	0.30	0.19	0.49
	Time (minutes)	17.56	26.08	45.14	21.16	51.45	55.78
sko90	PBE	***1.20***	13.55	7.91	***1.00***	13.45	9.03
	Dev.	0.41	0.17	0.48	0.43	0.21	0.54
	Time (minutes)	18.78	30.23	62.80	24.51	37.97	43.06
sk100a	PBE	***1.45***	13.15	8.06	***1.66***	12.80	9.15
	Dev.	0.48	0.12	0.39	0.28	0.19	0.39
	Time (minutes)	19.94	58.26	74.10	29.25	69.13	79.69
sk100d	PBE	***1.52***	13.06	7.53	***1.68***	12.72	8.47
	Dev.	0.24	0.20	0.49	0.22	0.28	0.53
	Time (minutes)	20.07	58.28	76.71	27.43	68.30	81.39
tho150	PBE	***2.13***	16.44	10.94	***2.74***	16.33	12.31
	Dev.	0.39	0.18	0.50	0.39	0.17	0.37
	Time (minutes)	29.73	101.40	142.94	49.35	152.04	147.95
wil50	PBE	***0.45***	8.08	3.92	***0.73***	8.05	5.09
	Dev.	0.36	0.28	0.49	0.32	0.29	0.41
	Time (minutes)	9.45	22.72	26.01	16.15	15.84	28.31

Table 10
Comparison with the other discretization approaches—2. Results obtained for Categories 3 and 4 problems with *500,000 FES* averaged over 10 runs.

Problem		LPSO			UPSO		
		Proposed	Based on (Helal & Abdelbar, 2014)	Based on (Peng et al., 2004)	Proposed	Based on (Peng et al., 2004)	Based on (Helal & Abdelbar, 2014)
tai20b	PBE	***0***	3.48	***0 (2)***	***0***	6.83	1.57
	Dev.	0.52	2.32	0.77	1.17	1.29	1.15
	Time (minutes)	7.72	6.72	9.61	10.43	10.80	9.31
tai30b	PBE	***0.15***	22.35	1.66	***0.01***	17.30	2.70
	Dev.	0.94	2.31	1.99	2.13	4.18	4.09
	Time (min.)	10.79	12.22	8.10	12.55	13.65	15.13
tai40b	PBE	***3.95***	32.56	5.72	***0.90***	32.19	9.71
	Dev.	1.90	1.99	2.83	2.55	1.92	4.59
	Time (min.)	12.63	17.75	11.72	14.66	11.52	20.40
tai50b	PBE	***2.79***	28.97	7.36	***1.03***	31.66	9.87
	Dev.	1.20	2.66	1.86	1.34	2.43	3.51
	Time (minutes)	13.84	22.86	17.39	16.94	15.85	28.51
tai60b	PBE	***1.51***	33.25	9.99	***1.03***	34.71	15.27
	Dev.	1.76	1.87	1.64	3.25	1.49	3.00
	Time (minutes)	15.16	29.31	20.51	18.67	20.34	38.04
tai80b	PBE	***2.79***	33.93	16.58	***2.51***	34.38	18.82
	Dev.	1.41	0.51	0.90	1.06	0.46	1.65
	Time (minutes)	17.11	42.02	48.82	22.93	30.58	56.47
ta100b	PBE	***2.25***	32.47	14.95	***1.53***	32.25	17.70
	Dev.	1.49	0.53	0.75	1.42	0.69	1.37
	Time (minutes)	19.44	59.62	67.47	28.27	68.43	77.82
ta150b	PBE	***3.01***	23.30	15.21	***3.36***	22.98	15.16
	Dev.	0.45	0.34	0.61	0.54	0.39	0.73
	Time (minutes)	23.78	104.34	144.80	49.05	150.88	156.73
bur26a	PBE	***0.14***	1.57	0.13	***0.02***	1.78	0.22
	Dev.	0.08	0.20	0.09	0.11	0.13	0.23
	Time (minutes)	8.52	9.71	12.69	11.15	14.01	11.72
chr15a	PBE	***0.40***	53.35	7.24	***0.40***	65.32	4.41
	Dev.	11.28	13.03	10.61	4.85	9.44	12.30
	Time (minutes)	7.51	6.15	7.60	10.37	5.37	8.26
chr25a	PBE	***24.92***	143.99	39.15	***10.85***	155.32	63.70
	Dev.	4.13	14.52	9.52	12.05	9.76	20.81
	Time (minutes)	11.55	9.92	11.81	11.19	11.64	12.77
els19	PBE	***0 (4)***	18.01	***0***	***0 (6)***	16.81	1.29
	Dev.	8.41	5.28	2.03	7.60	5.33	7.31
	Time (minutes)	9.78	7.48	9.44	10.16	8.73	9.46
esc64a	PBE	***0 (10)***	41.38	***0 (3)***	***0 (10)***	39.66	3.45
	Dev.	0	3.33	2.00	0	3.27	4.88
	Time (minutes)	16.60	31.44	38.02	19.87	36.36	38.66
kra30a	PBE	***2.24***	24.41	5.08	***2.25***	25.78	7.60
	Dev.	1.19	1.25	1.98	1.06	1.05	3.14
	Time (minutes)	12.12	12.37	14.31	12.76	13.54	14.83
kra30b	PBE	***0.42***	23.77	4.46	***0.42***	22.89	9.29
	Dev.	1.23	0.86	1.71	1.53	1.47	2.25
	Time (minutes)	11.21	11.44	14.65	13.61	16.17	13.54
ste36a	PBE	***3.46***	59.14	11.65	***3.21***	55.26	22.15
	Dev.	2.45	2.24	3.94	2.81	3.26	8.83
	Time (minutes)	12.42	15.28	16.40	12.84	17.26	18.49

For further confirmation, *t-test* was performed. The *t-test* results are shown in Table 11. With *18 degrees of freedom*, critical *t-value* to prove the hypothesis is 2.10 for the 95% *confidence interval*. Table 11 shows Δt, the difference in *t-value* and critical *t-value*. Hence, the positive entry in the Table 11 indicates that the proposed approach is better than the compared approach with 95% certainty.

4.5. Comparison with other published techniques

For further evaluation, the best performing PSO variant, UPSO, is compared to: Robust Tabu Search (RoTS; Taillard, 1991), Ant Colony Optimization (Maniezzo & Colorni, 1999) and Migrating Bird Optimization (Duman, Uysal, & Alkaya, 2012).

RoTS has been included due to its simplicity, ruggedness and proven performance. Unlike PSO, RoTS operates on a single solution. In each iteration, the working solution is updated by performing a move (swapping of solution elements) based on partial-cost matrix and three level acceptance condition. The acceptance condition is determined by a tabu-list (to avoid previously visited local optima) and aspiration criteria (to encourage exploration. More details about the algorithm can be found in Taillard (1991).

The reason for choosing ACO is two-fold: First, the underlying concept (search by mimicking the collective social behavior) of ACO and

Table 11
Deviation from critical value between proposed and compared approaches for 95% confidence interval.

Problem	Δt between LPSO and LPSO-1 (Helal & Abdelbar, 2014)	Δt between LPSO and LPSO-2 (Ping et al.,2004)	Δt between UPSO and UPSO-1 (Helal & Abdelbar, 2014)	Δt between UPSO and UPSO-2 (Ping et al., 2004)
had20	15.53	*−4.32*	21.96	1.11
lip40b	3.32	0.79	2.65	1.67
rou20	23.40	*−1.36*	14.37	6.20
tai20a	14.17	*−0.09*	20.25	1.32
tai30a	31.14	5.78	39.99	9.55
tai40a	47.81	15.92	57.83	20.32
tai50a	58.49	31.95	87.37	34.20
tai60a	65.92	25.35	66.78	40.17
tai80a	89.34	46.38	73.32	29.67
tai100a	101.75	35.29	106.19	42.83
nug30	47.01	2.44	29.78	5.31
sko42	55.10	15.12	45.81	14.06
sko49	60.47	16.48	58.45	16.72
sko81	72.24	43.82	104.27	39.90
sko90	83.16	30.53	78.04	36.43
sk100a	70.93	31.38	100.04	48.30
sk100d	112.65	36.52	98.82	38.38
tho150	98.34	41.99	97.71	53.11
wil50	50.93	16.39	52.88	27.41
tai20b	8.50	*−1.45*	12.47	*−0.28*
tai30b	28.52	1.46	13.86	2.19
tai40b	30.81	0.90	28.40	6.23
tai50b	30.91	7.44	33.80	10.14
tai60b	38.11	8.39	26.96	9.23
tai80b	60.52	22.93	82.60	26.61
tai100b	57.23	21.46	56.50	22.96
tai150b	111.76	50.05	89.29	39.64
bur26a	23.52	*−0.61*	31.30	2.47
chr15a	9.15	0.18	18.52	2.04
chr25a	25.97	6.82	25.70	5.81
els19	4.25	*−2.70*	5.34	*−0.87*
esc64a	41.76	1.44	41.90	5.73
kra30a	39.94	4.17	45.51	6.68
kra30b	44.73	4.13	29.69	8.73
ste36a	52.36	6.34	40.04	6.18

PSO is similar. Second, as ACO is inherently discrete, it will be interesting to compare the search behavior of both algorithms. So far, several variants of the ACO have been proposed (Maniezzo, Colorni, & Dorigo, 1994; Gambardella et al., 1999; Maniezzo & Colorni, 1999; Stützle & Dorigo, 1999) which are dedicated to solve QAP's. From these variants we have selected the basic Ant Colony Systems (Maniezzo et al., 1994), as it does not include any *local search* routine and relies purely on the collective behavior of the ants to solve the problem.

The Migrating Bird Optimization (MBO) is inspired from the V flight formation of birds. It is similar to PSO in terms of information sharing, i.e., in MBO, each solution shares some of its neighboring solutions downward in the hierarchy. The similar information sharing mechanism was used in Hierarchical PSO (Janson & Middendorf, 2005). In MBO, neighbors obtained from the solution, by pairwise exchange, are shared with the other solutions.

For the comparative evaluation, RoTS and ACO are implemented on the same platform with the parameters settings suggested by their respective work. For each algorithm, 10 runs were performed. For fair comparison, each algorithm is allowed to run for the same number of attempts, i.e., 500,000 iterations for RoTS and $\frac{500{,}000}{\text{swarm size}}$ iterations for PSO and ACO.

The test results shown in Table 12 indicate that, in most of the cases, RoTS provided best results followed by UPSO; whereas ACO performed rather inadequately. The results are in agreement with the earlier observations (Loiola et al., 2007) which indicated that for QAP, algorithms with solution modification techniques (like local search, TS, variable neighborhood search etc.) are more powerful than algorithms with solution construction approach (similar to the proposed approach) and are likely to provide better results. Nonetheless, it is interesting to observe that UPSO outperformed RoTS on Category-3 instances (real-life like, *taiXXXb*), which can lead to an interesting possibility of a hybrid/memetic algorithm with traits form both, RoTS and UPSO, for overall best performance.

Table 13 lists the results of comparative evaluation with MBO. As the results of MBO were reported for n^3 iterations, for fair comparison, UPSO was allowed to run for the same number of attempts, i.e., $\frac{n^3}{\text{swarm size}}$ iterations. Table 13 shows the best result obtained out of 10 runs for each instance. The performance of both the algorithms is mostly comparable; UPSO provided better results on 17 problem instances whereas MBO performed better on 20 problem instances. Overall, UPSO (3.31 % Average PBE) performed slightly better than MBO (3.68 % Average PBE). Note that UPSO outperformed MBO on all *real-life instances* (*category-4*).

5. Conclusions

A simple framework based on probabilistic velocity representation is developed to discretize the PSO algorithm and its common variants to make them suitable for QAP applications. The proposed framework retains all the essential elements of the original algorithms. Each particle learns from its own and/or learning exemplar's experience. The cognitive and social learning process is controlled in exactly the same manner as the generic version (by acceleration constants, c_1 and c_2). In addition, a new way to include

Table 12
Results of comparative evaluation obtained over 10 runs—1.

Problem	UPSO		ACO		RoTS	
	PBE	Dev.	PBE	Dev.	PBE	Dev.
had20	0.06	0.24	2.37	0.59	**0 (3)**	0.18
lip40b	**0 (2)**	8.44	25.19	0.12	**0 (10)**	0.00
rou20	1.45	0.94	8.99	0.76	**0 (10)**	0.00
tai20a	0.80	1.29	10.24	1.11	**0 (10)**	0.00
tai30a	2.40	0.41	12.04	0.26	**0 (10)**	0.00
tai40a	3.29	0.51	12.78	0.41	**0.31**	0.12
tai50a	3.53	0.50	13.05	0.39	**0.82**	0.17
tai60a	3.46	0.38	12.84	0.31	**0.92**	0.17
tai80a	3.69	0.21	12.21	0.14	**1.14**	0.12
tai100a	3.79	0.18	11.38	0.15	**1.09**	0.10
nug30	0.62	0.77	13.91	1.05	**0 (10)**	0.00
sko42	1.14	0.64	14.66	0.76	**0 (5)**	0.02
sko49	1.19	0.47	14.92	0.39	**0 (4)**	0.02
sko81	1.35	0.33	13.81	0.22	**0.03**	0.03
sko90	1.16	0.39	13.37	0.22	**0.06**	0.06
sko100a	1.87	0.25	13.02	0.17	**0.05**	0.04
sko100d	1.50	0.28	12.75	0.25	**0.03**	0.08
tho150	2.95	0.27	16.54	0.12	**0.16**	0.10
wil50	0.68	0.22	7.86	0.31	**0 (3)**	0.01
tai20b	**0.00**	0.69	7.34	0.99	0.45	12.54
tai30b	**0.18**	2.19	22.72	1.76	0.26	8.30
tai40b	**0.23**	2.10	35.03	0.89	7.82	4.38
tai50b	0.75	2.11	33.31	1.62	**0.29**	4.31
tai60b	**1.04**	2.23	33.43	1.53	2.02	3.19
tai80b	**1.79**	1.51	31.74	1.24	3.77	1.30
tai100b	**2.43**	1.62	32.09	0.81	3.04	1.01
tai150b	3.40	0.42	23.83	0.15	**2.13**	0.55
bur26a	0.02	0.06	1.73	0.19	0	0.08
chr15a	**0.00**	11.85	34.26	13.67	**0 (7)**	9.93
chr25a	19.81	9.69	150.42	10.01	**0**	2.73
els19	**0 (5)**	8.40	18.64	5.28	**0**	13.79
esc64a	**0 (10)**	0.00	37.93	5.33	**0 (10)**	0.00
kra30a	2.36	1.65	25.10	1.39	**0 (7)**	0.66
kra30b	1.44	0.90	23.07	1.11	**0 (9)**	0.06
ste36a	3.02	2.85	55.22	3.67	**0 (9)**	0.66
Average PBE	*2.23*	-	*23.08*	-	*0.70*	-

The best result obtained for each instance is shown in bold face.

Table 13
Results of comparative evaluation obtained over 10 runs—2.

Problem	UPSO		MBO	
	PBE	Dev.	PBE	Dev.
had20	**0.12**	0.42	0.43	0.16
lip40b	**0.00**	4.24	3.13	5.16
rou20	**2.09**	1.52	3.38	0.64
tai20a	5.44	0.99	**4.56**	0.63
tai30a	4.13	0.51	**3.74**	0.44
tai40a	3.71	0.51	**3.22**	0.43
tai50a	**3.25**	0.54	3.48	0.35
tai60a	3.64	0.35	**3.48**	0.29
tai80a	3.61	0.27	**3.25**	0.13
tai100a	3.23	0.27	**2.75**	0.12
nug30	**1.70**	1.42	2.38	0.52
sko42	**1.16**	0.68	1.58	0.37
sko49	**1.24**	0.70	1.28	0.34
sko81	1.19	0.32	**0.88**	0.22
sko90	0.96	0.41	**0.87**	0.16
sko100a	1.04	0.42	**0.56**	0.19
sko100d	1.20	0.34	**0.60**	0.18
tho150	1.75	0.19	**0.63**	0.32
wil50	0.62	0.37	**0.56**	0.18
tai20b	**0.82**	4.93	2.12	0.95
tai30b	0.75	3.93	**0.50**	1.26
tai40b	**0.41**	2.87	0.49	2.18
tai50b	0.94	1.89	**0.75**	1.00
tai60b	0.56	1.71	**0.39**	0.77
tai80b	2.19	1.04	**1.88**	0.53
tai100b	1.10	1.27	**0.54**	0.93
tai150b	1.52	0.53	**1.41**	0.49
bur26a	**0.15**	0.08	**0.15**	0.04
chr15a	**20.27**	25.70	29.65	8.22
chr25a	**34.25**	11.84	36.99	7.91
els19	**1.10**	10.32	1.22	5.72
esc64a	**0 (10)**	0.00	**0 (10)**	0.00
kra30a	**3.43**	1.52	3.71	0.97
kra30b	**0.22**	1.79	2.58	0.97
ste36a	**4.62**	2.42	5.54	1.72
Average PBE	***3.31***	**-**	***3.68***	**-**

The best result obtained for each instance is shown in bold face.

the inertia weight and a restart mechanism to counter stagnation is proposed.

The proposed framework will enable to translate existing vast research efforts related to the PSO algorithm in the continuous domain, for the QAP applications. To demonstrate this fact, generic PSO (GPSO) and its five variants were discretized based on the proposed discretizing framework and applied to the suite of 35 well-known QAP problem instances. In the test carried out for comparative evaluation, UPSO outperformed all compared PSO variants. However, it was closely followed by FIPS and LPSO; which indicates that the variants based on simple information exchange mechanism are equally effective to solve the QAP.

As possible future work, a new PSO variant based on simple information exchange mechanism within neighbors can be developed for QAP, to include the advantage of speed and better performance. Moreover, for QAP, the local search techniques are found to be more successful compared to search methods based on solution construction, which include most of the meta-heuristics (including the proposed approach); combining both of them as cooperative elements in a single search algorithm is likely to give better results.

It is pertinent to mention that while implementing the discrete version of the PSO variants, the parameter values suggested by the original research in the continuous domain were used. Further research efforts may be directed toward evaluating the effects of these parameters for combinatorial optimization problems.

Acknowledgments

The author is grateful to the Deanship of Scientific Research, Research Center of College of Engineering, King Saud University for the financial support.

References

AlRashidi, M. R., & El-Hawary, M. E. (2009). A survey of particle swarm optimization applications in electric power systems. *IEEE Transactions on Evolutionary Computation, 13*, 913–918.
Benlic, U., & Hao, J.-K. (2013). Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation, 219*, 4800–4815.
Benlic, U., & Hao, J. K. (2015). Memetic search for the quadratic assignment problem. *Expert Systems with Applications, 42*(1), 584–595.
Bland, J., & Dawson, G. (1991). Tabu search and design optimization. *Computer-aided Design, 23*, 195–201.
Bui, T. N., & Moon, B. (1994). A genetic algorithm for a special class of the quadratic assignment problem. In *Quadratic assignment and related problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science: vol. 16* (pp. 99–116). AMS.
Burkard, R., Karisch, S., & Rendl, F. (1997). QAPLIB—A quadratic assignment problem library. *Journal of Global Optimization, 10*, 391–403.
Chakrapani, J., & Skorin-Kapov, J. (1993). Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research, 41*, 327–341.
Chen, W.-N., Zhang, J., Chung, H. S.-H., Zhong, W.-L., Wu, W.-G., & Shi, Y.-H. (2010). A novel set-based particle swarm optimization method for discrete optimization problems. *IEEE Transactions on Evolutionary Computation, 14*, 278–300.

Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering: vol. 141* (pp. 219–239). Springer.

Clerc, M., & Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation, 6*, 58–73.

Correa, E. S., Freitas, A. A., & Johnson, C. G. (2006). A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In *Proceedings of the 8th annual conference on genetic and evolutionary computation* (pp. 35–42). ACM.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 26*, 29–41.

Drezner, Z. (2008). Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Computers & Operations Research, 35*, 717–736.

Drezner, Z., & Marcoulides, G. A. (2004). A distance-based selection of parents in genetic algorithms. In *Metaheuristics: Computer decision-making* (pp. 257–278). Springer.

Duman, E., Uysal, M., & Alkaya, A. F. (2012). Migrating birds optimization: A new meta-heuristic approach and its performance on quadratic assignment problem. *Information Sciences, 217*, 65–77.

El-Baz, M. A. (2004). A genetic algorithm for facility layout problems of different manufacturing environments. *Computers & Industrial Engineering, 47*, 233–246.

Gambardella, L. M., Taillard, É., & Dorigo, M. (1999). Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society, 50*, 167–176.

Hafiz, F. M. F., & Abdennour, A. (2013). A team-oriented approach to particle swarms. *Applied Soft Computing, 13*, 3776–3791.

Helal, A. M., & Abdelbar, A. M. (2014). Incorporating domain-specific heuristics in a particle swarm optimization approach to the quadratic assignment problem. *Memetic Computing, 6*, 241–254.

Hu, X., Eberhart, R. C., & Shi, Y. (2003). Swarm intelligence for permutation optimization: A case study of n-queens problem. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium, 2003 (SIS'03)* (pp. 243–246). IEEE.

James, T., Rego, C., & Glover, F. (2009). Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 39*, 579–596.

Jones, T., & Forrest, S. (1995). Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In *ICGA: vol. 95* (pp. 184–192).

Janson, S., & Middendorf, M. (2005). A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 35*, 1272–1282.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks: vol. 4* (pp. 1942–1948). IEEE.

Kennedy, J., & Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. In *1997 IEEE international conference on systems, man, and cybernetics, 1997. Computational cybernetics and simulation: vol. 5* (pp. 4104–4108). IEEE.

Kennedy, J., & Mendes, R. (2002). Population structure and particle swarm performance. In *Proceedings of the world on congress on computational intelligence: vol. 2* (pp. 1671–1676). IEEE.

Koopmans, T. C., & Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica, 25*, 53–76.

Kulkarni, R. V., & Venayagamoorthy, G. K. (2011). Particle swarm optimization in wireless-sensor networks: A brief survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 41*, 262–267.

Langeveld, J., & Engelbrecht, A. P. (2011). A generic set-based particle swarm optimization algorithm. In *International conference on swarm intelligence* (pp. 1–10). ICSI.

Langeveld, J., & Engelbrecht, A. P. (2012). Set-based particle swarm optimization applied to the multidimensional knapsack problem. *Swarm Intelligence, 6*, 297–342.

Liang, J. J., Qin, A. K., Suganthan, P. N., & Baskar, S. (2006). Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation, 10*, 281–295.

Liang, J., & Suganthan, P. N. (2005). Dynamic multi-swarm particle swarm optimizer. In *Proceedings 2005 IEEE swarm intelligence symposium (SIS'05)* (pp. 124–129). IEEE.

Liu, B., Wang, L., & Jin, Y.-H. (2007). An effective PSO-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 37*, 18–27.

Liu, H., Abraham, A., & Clerc, M. (2007a). An hybrid fuzzy variable neighborhood particle swarm optimization algorithm for solving quadratic assignment problems. *Journal of UCS, 13*, 1309–1331.

Liu, H., Abraham, A., & Zhang, J. (2007b). A particle swarm approach to quadratic assignment problems. In *Soft computing in industrial applications* (pp. 213–222). Springer.

Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research, 176*, 657–690.

Mangat, V. (2012). Survey on particle swarm optimization based clustering analysis. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, & J. Zurada (Eds.), *Swarm and evolutionary computation: vol. 7269* (pp. 301–309). Berlin Heidelberg: Springer.

Maniezzo, V., Colorni, A., & Dorigo, M. (1994). *The ant system applied to the quadratic assignment problem.* Technical Report (TR/IRIDIA/94-28), IRIDIA. Université Libre de Bruxelles.

Maniezzo, V., & Colorni, A. (1999). The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering, 11*, 769–778.

Mendes, R., Kennedy, J., & Neves, J. (2004). The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation, 8*, 204–210.

Menhas, M. I., Wang, L., Fei, M.-R., & Ma, C.-X. (2011). Coordinated controller tuning of a boiler turbine unit with new binary particle swarm optimization algorithm. *International Journal of Automation and Computing, 8*, 185–192.

Merz, P., & Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation, 4*, 337–352.

Misevicius, A. (2005). A tabu search algorithm for the quadratic assignment problem. *Computational Optimization and Applications, 30*, 95–111.

Misevičius, A., & Kilda, B. (2005). Comparison of crossover operators for the quadratic assignment problem. *Information Technology and Control, 34*, 109–119.

Misevicius, A., & Rubliauskas, D. (2005). Performance of hybrid genetic algorithm for the grey pattern problem. *Information Technology and Control, 34*, 15–24.

Neethling, M., & Engelbrecht, A. P. (2006). Determining RNA secondary structure using set-based particle swarm optimization. In *IEEE congress on evolutionary computation* (pp. 1670–1677). IEEE.

Nissen, V. (1994). Solving the quadratic assignment problem with clues from nature. *IEEE Transactions on Neural Networks, 5*, 66–72.

Pang, W., Wang, K.-p., Zhou, C.-g., & Dong, L.-j. (2004). Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In *The fourth international conference on computer and information technology, 2004 (CIT'04)* (pp. 796–800). IEEE.

Parsopoulos, K., & Vrahatis, M. (2004). UPSO: A unified particle swarm optimization scheme. *Lecture Series on Computer and Computational Sciences, 1*, 868–873.

Ratnaweera, A., Halgamuge, S., & Watson, H. C. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation, 8*, 240–255.

Shen, M., Zhan, Z.-H., Chen, W.-N., Gong, Y.-J., Zhang, J., & Li, Y. (2014). Bi-velocity discrete particle swarm optimization and its application to multicast routing problem in communication networks. *IEEE Transactions on Industrial Electronics, 61*, 7141–7151.

Skorin-Kapov, J. (1990). Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing, 2*, 33–45.

Skorin-Kapov, J. (1994). Extensions of a tabu search adaptation to the quadratic assignment problem. *Computers & Operations Research, 21*, 855–865.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research, 174*, 1519–1539.

Stützle, T., & Dorigo, M. (1999). ACO algorithms for the quadratic assignment problem. In *New ideas in optimization* (pp. 33–50). McGraw-Hill.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel computing, 17*, 443–455.

Vanneschi, L., Tomassini, M., Clergue, M., & Collard, P. (2003). Difficulty of unimodal and multimodal landscapes in genetic programming. In *Genetic and Evolutionary Computation–GECCO 03* (pp. 1788–1799). Springer.

Veenhuis, C. B. (2008). A set-based particle swarm optimization method. In *Parallel problem solving from nature—PPSN X* (pp. 971–980).

Wang, L., Wang, X., Fu, J., & Zhen, L. (2008). A novel probability binary particle swarm optimization algorithm and its application. *Journal of Software, 3*, 28–35.

Yuhui, S., & Eberhart, R. C. (1999). Empirical study of particle swarm optimization. In *Proceedings of the 1999 congress on evolutionary computation (CEC 99): vol. 3* (p. 1950). IEEE.