Hindawi Publishing Corporation
Journal of Artificial Evolution and Applications
Volume 2008, Article ID 861512, 9 pages
doi:10.1155/2008/861512

Research Article

A Discrete Particle Swarm Optimization Algorithm for Uncapacitated Facility Location Problem

Ali R. Guner[1] and Mehmet Sevkli[2]

[1] *Department of Industrial and Manufacturing Engineering, College of Engineering, Wayne State University, Detroit, MI 48202, USA*
[2] *Department of Industrial Engineering, Faculty of Engineering, Fatih University, 34500 Büyükçekmece, Istanbul, Turkey*

Correspondence should be addressed to Mehmet Sevkli, msevkli@fatih.edu.tr

Received 5 July 2007; Revised 27 December 2007; Accepted 12 March 2008

Recommended by T. Blackwell

A discrete version of particle swarm optimization (*DPSO*) is employed to solve uncapacitated facility location (*UFL*) problem which is one of the most widely studied in combinatorial optimization. In addition, a hybrid version with a local search is defined to get more efficient results. The results are compared with a continuous particle swarm optimization (*CPSO*) algorithm and two other metaheuristics studies, namely, genetic algorithm (*GA*) and evolutionary simulated annealing (*ESA*). To make a reasonable comparison, we applied to same benchmark suites that are collected from *OR*-library. In conclusion, the results showed that *DPSO* algorithm is slightly better than *CPSO* algorithm and competitive with *GA* and *ESA*.

1. INTRODUCTION

Efficient supply chain management has led to increased profit, increased market share, reduced operating cost, and improved customer satisfaction for many businesses. One strategic decision in supply chain management is facility location [1]. Location problems are classified into categories with some assumptions such as limiting the capacity and open number of sites. The uncapacitated facility location (*UFL*) problem assumes the cost of satisfying the client requirements has two components: a fixed cost of setting up a facility in a given site, and a transportation cost of satisfying the customer requirements from a facility. The capacities of all the facilities are assumed to be infinite [2].

1.1. Literature review

There are many different titles for the *UFL* problem in the literature: the problem of a nonrecoverable tools optimal system [3], the standardization and unification problem [4], the location of bank accounts problem [5], warehouse location problem [6], uncapacitated facility location problem [7], and so on. The academic interest to investigate this mathematical model reasoned different interpretations. *UFL* problem is one of the most widely studied problems in combinatorial optimization problems thus there is a very rich literature in operations research (*OR*) for this kind of problem [8]. All important approaches relevant to *UFL* problems can be classified into two main categories: exact algorithms and metaheuristics-based methods.

There is a variety of exact algorithms to solve the *UFL* problem, such as branch and bound [6, 9], linear programming [10], Lagrangean relaxation algorithms [11], dual approach (*DUALLOC*) of Erlenkotter [12], and the primal-dual approaches of Körkel [13]. Although Erlenkotter [12] developed this dual approach as an exact algorithm, it can also be used as a heuristic to find good solutions. It is obvious that since the *UFL* problem is NP-hard [14], exact algorithms may not be able to solve large practical problems efficiently. There are several studies to solve *UFL* problem with heuristics and metaheuristics methods. Alves and Almeida [15] proposed a simulated annealing algorithms and reported they produce high-quality solutions, but quite expensive in computation times. A new version of evolutionary simulated annealing algorithm (*ESA*) called distributed *ESA* presented by Aydin and Fogarty [16]. They stated that with implementing it they get good quality of solutions within short times. Another popular metaheuristic, tabu search algorithm, is applied by Al-Sultan and Al-Fawzan in [17]. Their application produces good solutions, but takes significant computing time and limits the applicability of the algorithm. Michel and Van Hentenryck [18] also applied tabu search and their proposed algorithm generates more robust solutions. Sun [19] examined tabu search procedure against the Lagrangean method and heuristic procedures reported by Ghosh [2]. Genetic algorithms (*GA*) are also applied by Kratica and Jaramillo [20, 21]. Finally, there are also artificial neural network approaches to solve *UFL* problems in Gen et al. [22] and Vaithyanathan et al. [23].

The particle swarm optimization (*PSO*) is one of the recent metaheuristics invented by Eberhart and Kennedy [24] based on the metaphor of social interaction and communication such as bird flocking and fish schooling. On one hand, it can be counted as an evolutionary method with its way of exploration via neighborhood of solutions (particles) across a population (swarm) and exploiting the generational information gained. On the other hand, it is different from other evolutionary methods in such a way that it has no evolutionary operators such as crossover and mutation. Another advantage is its ease of use with fewer parameters to adjust. In *PSO*, the potential solutions, the so-called particles, move around in a multidimensional search space with a velocity, which is constantly updated by the particle's own experience and the experience of the particle's neighbors or the experience of the whole swarm. *PSO* has been successfully applied to a wide range of applications such as function optimization, neural network training [25], task assignment [26], and scheduling problem [27, 28].

Since *PSO* is developed for continuous optimization problem initially, most existing *PSO* applications are resorting to continuous function value optimization [29–32]. Recently, a few researches applied *PSO* for discrete combinatorial optimization problems [26–28, 33–37].

1.2. UFL problem definition

In a *UFL* problem, there are a number of customers, m, to be satisfied by a number of facilities, n. Each facility has a fixed cost, fc_j. A transport cost, c_{ij}, is accrued for serving customer, i, from facility, j. There is no limit of capacity for any candidate facility and the whole demand of each customer has to be assigned to one of the facilities. We are asked to find the number of facilities to be established and specify those facilities such that the total cost will be minimized (1). The mathematical formulation of the problem can be stated as follows [14]:

$$Z = \min\left(\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\cdot x_{ij} + \sum_{j=1}^{n} fc_j \cdot y_j\right), \qquad (1)$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \text{ in } m, \qquad (2)$$

$$0 \le x_{ij} \le y_j, \quad y_j \in \{0;1\}, \qquad (3)$$

where $i = 1,\dots,m$; $j = 1,\dots,n$; x_{ij} represents the quantity supplied from facility i to customer j; y_j indicates whether facility j is established ($y_j = 1$) or not ($y_j = 0$).

Constraint (2) makes sure that all customers demands have been met by an open facility and (3) is to keep integrity. Since it is assumed that there is no capacity limit for any facility, the demand size of each customer is ignored, and therefore (2) established without considering demand variable.

It is obvious that since the main decision in *UFL* is opening or closing facilities, *UFL* problems are classified in discrete problems. On the other hand, *PSO* is mainly designed for continuous problem thus it has some drawbacks when applying *PSO* for a discrete problem. This tradeoff increased our curiosity to apply *PSO* algorithm for solving *UFL* problem.

The organization of the paper is as follows: in Section 2, the implementation of both continuous and discrete *PSO* algorithms for *UFL* problem is given with the details of how a local search procedure is embedded. Section 3 reports the experimental settings and results. There are three sets of comparisons: the first is between *CPSO* and *DPSO* algorithms; the second is between *CPSO* with local search ($CPSO_{LS}$) and *DPSO* with local search ($DPSO_{LS}$) algorithms; and the third is among $DPSO_{LS}$ with two other algorithms from the literature. Finally, Section 4 provides with the conclusion.

2. PSO ALGORITHMS FOR UFL PROBLEM

As mentioned Section 1, *PSO* is one of the population-based optimization technique inspired by nature. It is a simulation of social behaviour of a swarm, that is, bird flocking, fish schooling. Suppose the following scenario: a flock of bird is randomly searching for food in an area, where there is only one piece of food available and none of them knows where it is, but they can estimate how far it would be at each iteration. The problem here is "what is the best strategy to find and get the food?". Obviously, the simplest strategy is to follow the bird known as the nearest one to the food. *PSO* inventers were inspired of such natural process-based scenarios to solve the optimization problems. In *PSO*, each single solution, called a particle, is considered as a bird, the group becomes a swarm (population) and the search space is the area to explore. Each particle has a fitness value calculated by a fitness function, and a velocity of flying towards the optimum, the food. All particles fly across the problem space following the particle nearest to the optimum. *PSO* starts with initial population of solutions, which is updated iteration-by-iteration. Therefore, *PSO* can be counted as an evolutionary algorithm besides being a metaheuristics method, which allows exploiting the searching experience of a single particle as well as the best of the whole swarm.

2.1. Continuous PSO algorithm for UFL problem

The continuous particle swarm optimization (*CPSO*) algorithm used here is proposed by the authors, Sevkli and Guner [33]. The *CPSO* considers each particle has three key vectors: position (X_i), velocity (V_i), and open facility (Y_i). X_i denotes the ith position vector in the swarm,

Table 1: An illustration of deriving open facility vector from position vector for a 6-customer 5-facility problem.

ith particle vectors	Particle dimension (k)				
	1	2	3	4	5
Position vector (X_i)	1.8	3.01	−0.99	0.72	−5.45
Velocity vector (V_i)	−0.52	2.06	3.56	2.45	−1.44
Open facility vector (Y_i)	1	1	0	0	1

$X_i = [x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}]$, where x_{ik} is the position value of the ith particle with respect to the kth dimension ($k = 1, 2, 3, \ldots n$). V_i denotes the ith velocity vector in the swarm, $V_i = [v_{i1}, v_{i2}, v_{i3}, \ldots, v_{in}]$, where v_{ik} is the velocity value of the ith particle with respect to the kth dimension. Y_i represents the opening or closing facilities based on the position vector (X_i), $Y_i = [y_{i1}, y_{i2}, y_{i3}, \ldots, y_{in}]$, where y_{ik} represents opening or closing kth facility of the ith particle. For an *n-facility* problem, each particle contains n number of dimensions.

Initially, the position and the velocity vectors are generated as continuous uniform random variables, using the following rules:

$$x_{ij} = x_{\min} + (x_{\max} - x_{\min}) \times r_1,$$
$$v_{ij} = v_{\min} + (v_{\max} - v_{\min}) \times r_2, \tag{4}$$

where $x_{\min} = -10.0$, $x_{\max} = 10.0$, $v_{\min} = -4.0$, $v_{\max} = 4.0$ which are consistent with the literature [38], and r_1 and r_2 are uniform random numbers in [0, 1] for each dimension and particle. The position vector $X_i = [x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}]$ corresponds to the continuous position values for the n facilities, but it does not represent a candidate solution to calculate a total cost (fitness value). In order to create a candidate solution, a particle, the position vector is converted to a binary variable, $Y_i \leftarrow X_i$, which is also a key element of a particle. In other words, a continuous set is converted to a discrete set for the purpose of creating a candidate solution, particle. The fitness of the ith particle is calculated by using open facility vector (Y_i). For simplicity, let $f_i(Y_i \leftarrow X_i)$ be denoted as f_i.

In order to ascertain how to derive an open facility vector from position vector, an instance of 5-facility problem is illustrated in Table 1. Position values are converted to binary variables using following formula:

$$y_i = \lfloor |x_i| (\operatorname{mod} 2) \rfloor. \tag{5}$$

In (5), the absolute value of a position value is first divided by 2 and then the remainder is floored to nearest integer number. Then it is assigned to corresponding element of the open facility vector. For example, fifth element of the open facility vector, y_5, can be calculated as follows:

$$\lfloor |-5.45| (\operatorname{mod} 2) \rfloor = \lfloor 5.45 (\operatorname{mod} 2) \rfloor = \lfloor 1.45 \rfloor = 1. \tag{6}$$

Table 2: An example of 5-facility to 6-customer.

Facility locations		1	2	3	4	5
Fixed cost		12	5	3	7	9
Customers	1	2	3	6	7	1
	2	0	5	8	4	12
	3	11	6	14	5	8
	4	19	18	21	16	13
	5	3	9	8	7	10
	6	4	7	9	6	0

Considering the 5-facility to 6-customer example shown in Table 2, the total cost of *open facility vector* (Y_i) can be calculated as follows:

$$\begin{aligned}
&\text{total cost} \\
&\quad = \{\text{open facilities fixes costs } (fc_j) \\
&\qquad + \min(\text{cost of supply from open} \\
&\qquad\qquad \text{facilities to customer } i[ci_j])\} \\
&\quad \cdot \{(12 + 5 + 9) + \min(2, 3, 1) \\
&\qquad + \min(0, 5, 12) + \min(11, 6, 8) \\
&\qquad + \min(19, 18, 13) + \min(3, 9, 10) + \min(4, 7, 0)\} \\
&\quad = \{(26) + (1 + 0 + 6 + 13 + 3 + 0)\} \\
&\quad = \{26 + 23\} = \{49\}.
\end{aligned} \tag{7}$$

For each particle in the swarm, let define $P_i = [p_{i1}, p_{i2}, \ldots, p_{in}]$, as the personal best, where p_{ik} denotes the position value of the ith personal best with respect to the kth dimension. The personal bests are determined just after generating Y_i vectors and their corresponding fitness values. In every generation, the personal best of each particle is updated based on its position vector and fitness value. Regarding the objective function, $f_i(Y_i \leftarrow X_i)$, the fitness values for the personal best of the ith particle, P_i, is denoted by $f_i^{pb} = f(Y_i \leftarrow P_i)$. At the beginning, the personal best values are equal to position values ($P_i = X_i$), explicitly $P_i = [p_{i1} = x_{i1}, p_{i2} = x_{i2}, p_{i3} = x_{i3}, \ldots, p_{in} = x_{in}]$ and the fitness values of the personal bests are equal to the fitness of positions, $f_i^{pb} = f_i$.

Then the best particle in the whole swarm is selected as the global best. $G = [g_1, g_2, g_3, \ldots, g_n]$ denotes the best position of the globally best particle, $f_g = f(Y \leftarrow G)$, achieved so far in the whole swarm. At the beginning, global best fitness value is determined as the best of personal bests over the whole swarm, $f_g = \min\{f_i^{pb}\}$, with its corresponding position vector X_g, which is to be used for $G = X_g$, where $G = [g_1 = x_{g1}, g_2 = x_{g2}, g_3 = x_{g3}, \ldots, g_n = x_{gn}]$ and corresponding $Y_g = [y_{g1}, y_{g2}, y_{g3}, \ldots, y_{gn}]$ denotes the open facility vector of the global best found.

```
Begin
  Initialize particles (population) randomly
  For each particle
   Calculate open facility vectors (1)
   Calculate fitness value using open facility vector
   Set to position vector and fitness value
   as personal best (P_i^t)
   Select the best particle and its position
   vector as global(G^t)
  End
  Do {
   Update inertia weight
   For each particle
     Update velocity (8)
     Update position(9)
     Find open facility vectors
     Calculate fitness value using open
     facility vector (1)
     Update personal best(P_i^t)
     Update the global best (G^t) value with
     position vector
    End
   Apply local search (for CPSO_LS) to global best
  } While (Maximum Iteration is not reached)
End
```

Algorithm 1: Pseudocode of *CPSO* algorithm for *UFL* problem.

The velocity of each particle is updated based on its personal best and the global best in the following way:

$$v_{ik}(t+1) = w \cdot v_{ik}(t) + c_1 r_1 (p_{ik}(t) - x_{ik}(t)) + c_2 r_2 (g_k(t) - x_{ik}(t)), \quad (8)$$

where w is the inertia weight used to control the impact of the previous velocities on the current one, t is generation index, r_1 and r_2 are different random numbers for each dimension and particle in [0, 1], and c_1 and c_2 are the learning factors which are also called social and cognitive parameters. The next step is to update the positions.

$$x_{ik}(t+1) = x_{ik}(t) + v_{ik}(t+1). \quad (9)$$

After updating position values for all particles, the corresponding open facility vectors can be determined by their fitness values in order to start a new iteration if the predetermined stopping criterion has not met yet. In this study, we employed the *gbest* model of Kennedy et al. [38] for *CPSO*, which is elaborated in the pseudocode given below.

2.2. Discrete PSO algorithm for UFL problem

The discrete *PSO* (*DPSO*) algorithm used here is first proposed by Pan et al. [37] for the no-wait flowshop scheduling problem. We employed the DPSO algorithm for UFL problem. In DPSO, each particle is based on only the open facility vector $Y_i = [y_{i1}, y_{i2}, y_{i3}, \ldots, y_{in}]$, where y_{ik} represents opening or closing kth facility of the ith particle. For an *n-facility* problem, each particle contains n number of dimensions. The dimensions of Y_i are binary random numbers. The fitness of the ith particle is calculated by $f_i(Y_i)$.

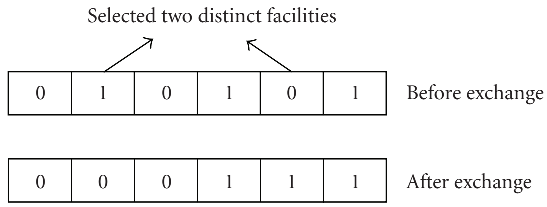

Figure 1: Exchange operator.

The open facility vector (Y_i) of the particle i at iteration t can be updated as follows [37]:

$$Y_i^t = c_2 \oplus F_3(c_1 \oplus F_2(w \oplus F_1(Y_i^{t-1}), P_i^{t-1}), G^{t-1}), \quad (10)$$

$$\lambda_i^t = w \oplus F_1(Y_i^{t-1}). \quad (11)$$

Equation (10) consists of three components: the first component (11) is the *velocity* of the particle. F_1 represents the exchange operator (Figure 1) which is selecting two distinct facilities from the open facility vector, Y_i^{t-1}, of particle and swapping randomly with the probability of w. In other words, a uniform random number, r, is generated between 0 and 1. If r is less than w then the exchange operator is applied to generate a perturbed Y_i vector of the particle by $\lambda_i^t = F_1(Y_i^{t-1})$, otherwise current Y_i is kept as $\lambda_i^t = Y_i^{t-1}$.

$$\delta_i^t = c_1 \oplus F_2(\lambda_i^t, P_i^{t-1}). \quad (12)$$

The second component (12) is the *cognition* part of the particle representing particle's own experience. F_2 represents the one-cut crossover (Figure 2) with the probability of c_1. Note that λ_i^t and P_i^{t-1} will be the first and second parents for the crossover operator, respectively. It is resulted either in $\delta_i^t = F_2(\lambda_i^t, P_i^{t-1})$ or in $\delta_i^t = \lambda_i^t$ depending on the choice of a uniform random number

$$X_i^t = c_2 \oplus F_3(\delta_i^t, G^t). \quad (13)$$

The third component (13) is the *social* part of the particle representing experience of whole swarm. F_3 represents the two-crossover (Figure 3) operator with the probability of c_2. Note that δ_i^t and G^{t-1} will be the first and second parents for the crossover operator, respectively. It is resulted either in $Y_i^t = F_3(\delta_i^t, G^{t-1})$ or in $Y_i^t = \delta_i^t$ depending on the choice of a uniform random number. In addition, one-cut and two-cut crossovers produce two children. In this study, we selected one of the children randomly.

The corresponding Y_i vectors are determined with their fitness values so as to start a new iteration if the predetermined stopping criterion has not met yet. We apply the *gbest* model of Kennedy and Eberhart for DPSO. The pseudocode of DPSO is given in Algorithm 2.

2.3. Local search for CPSO and DPSO algorithm

Apparently, *CPSO* and *DPSO* conduct such a rough search that they produce premature results, which do not offer satisfactory solutions. For this reason, it is inevitable to embed

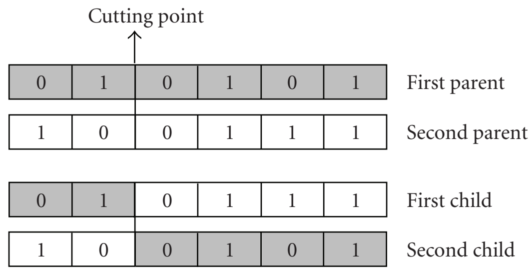

Figure 2: One-cut crossover operator.

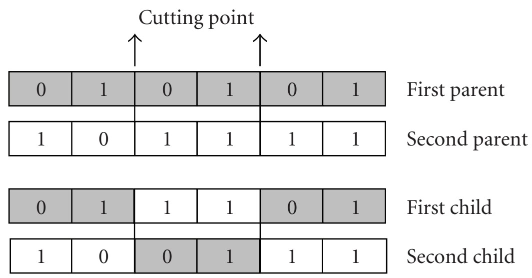

Figure 3: Two-cut crossover operator.

Table 3: Benchmarks tackled with the sizes and the optimum values.

Benchmarks		
Problems	Size ($m \times n$)	Optimum
Cap71	16×50	932615.75
Cap72	16×50	977799.40
Cap73	16×50	1010641.45
Cap74	16×50	1034976.98
Cap101	25×50	796648.44
Cap102	25×50	854704.20
Cap103	25×50	893782.11
Cap104	25×50	928941.75
Cap131	50×50	793439.56
Cap132	50×50	851495.33
Cap133	50×50	893076.71
Cap134	50×50	928941.75
CapA	100×1000	17156454.48
CapB	100×1000	12979071.58
CapC	100×1000	11505594.33

a local search algorithm to *CPSO* and *DPSO* so as to produce more satisfactory solutions. In this study, we have applied a simple local search method to neighbours of the global best particle in every generation. In *CPSO* global best has three vectors, so local search is applied to the position vector (X_i). Since *DPSO* has one vector. Local search is applied only this vector (Y_i).

The neighbourhood structure with which neighbour solutions are determined to move is one of the key elements in metaheuristics. The performance of the hybrid algorithm depends on the efficiency of the neighbourhood structure. For both algorithms, flip operator is employed as a neighbourhood structure. It is defined as: picking randomly one position value (i) of the global best, then changing its value with using formula (14) for $CPSO_{LS}$ and formula (15) for $DPSO_{LS}$. Since binary and continuous values are stored in Y_i and X_i vectors, respectively, the equations are slightly different

$$g_i \longleftarrow g_i + 1, \tag{14}$$

$$g_i \longleftarrow 1 - g_i. \tag{15}$$

The local search algorithm applied in this study is sketched in Algorithm 3. The global best found at the end of each iteration of *CPSO* and *DPSO* is adopted as the initial solution by the local search algorithm. In order not to lose the best found and to diversify the solution, the global best is modified with two facilities (η and κ) which are randomly chosen. Then flip operator is applied to as long as it gets a better solution. The final produced solution, s , is replaced with the old global best if its fitness value is better than the initial one.

```
Begin
  Initialize open facility vector
  Calculate fitness value using Y_i(1)
  Do
    Find personal best (P_i^t)
    Find global best (G^t)
    For Each Particle
      Apply velocity component (11)
      Apply cognition component (12)
      Apply social component (13)
      Calculate fitness value using Y_i(1)
    Apply local search (for DPSO_LS)
    to global best
  While (Maximum Iteration is not reached)
End
```

Algorithm 2: Pseudocode of *DPSO* algorithm for *UFL* problem.

3. EXPERIMENTAL RESULTS

The experimental study has been completed in three stages; first, we compared the *CPSO* and *DPSO* algorithms without local search, then we compared these algorithms with local search ($CPSO_{LS}$ and $DPSO_{LS}$) with respect to their solution quality; finally, $DPSO_{LS}$ results are compared with other two metaheuristics, namely, genetic algorithm (*GA*) and evolutionary simulated annealing algorithm (*ESA*). Experimental results provided in this section are carried out over 15 benchmark problems well-known by the researchers of *UFL problem*. The benchmarks are undertaken from the *OR library* [39], a collection of benchmarks for operations research (*OR*) studies. There are currently 15 *UFL* test problems in the *OR*-library. Among these test problems, 12 are relatively small in size ranging from $m \times n = 50 \times 16$ to $m \times n = 50 \times 50$. The other three are relatively large with $m \times n = 1000 \times 100$. The benchmarks are introduced in Table 3 with their optimum values. Although the optimum

Table 4: Experimental results gained for *CPSO* and *DPSO* without local search.

	CPSO			DPSO		
Problem	ARPE	HR	ACPU	ARPE	HR	ACPU
Cap71	0.026	0.83	0.1218	0.000	1.00	0.0641
Cap72	0.050	0.83	0.1318	0.000	1.00	0.0651
Cap73	0.034	0.73	0.1865	0.000	1.00	0.0708
Cap74	0.095	0.00	0.1781	0.000	1.00	0.0693
Cap101	0.183	0.00	0.8818	0.000	1.00	0.3130
Cap102	0.135	0.33	0.7667	0.000	1.00	0.3062
Cap103	0.145	0.00	0.9938	0.000	1.00	0.3625
Cap104	0.286	0.60	0.6026	0.002	0.93	0.2021
Cap131	0.911	0.00	3.6156	0.173	0.13	2.5464
Cap132	0.756	0.00	3.5599	0.090	0.17	2.6328
Cap133	0.496	0.00	3.7792	0.042	0.43	2.5292
Cap134	0.691	0.23	3.3333	0.000	1.00	1.7167
CapA	21.242	0.00	29.5739	8.654	0.00	24.8972
CapB	10.135	0.00	27.1318	4.918	0.00	22.0652
CapC	8.162	0.00	27.6149	4.545	0.00	23.1340

Table 5: Experimental results of $CPSO_{LS}$ and $DPSO_{LS}$.

	CPSO_{LS}			DPSO_{LS}		
Problem	ARPE	HR	ACPU	ARPE	HR	ACPU
Cap71	0.000	1.00	0.0146	0.000	1.00	0.0130
Cap72	0.000	1.00	0.0172	0.000	1.00	0.0078
Cap73	0.000	1.00	0.0281	0.000	1.00	0.0203
Cap74	0.000	1.00	0.0182	0.000	1.00	0.0109
Cap101	0.000	1.00	0.1880	0.000	1.00	0.1505
Cap102	0.000	1.00	0.0906	0.000	1.00	0.0557
Cap103	0.000	1.00	0.2151	0.000	1.00	0.1693
Cap104	0.000	1.00	0.0370	0.000	1.00	0.0344
Cap131	0.000	1.00	1.4281	0.000	1.00	0.4922
Cap132	0.000	1.00	1.0245	0.000	1.00	0.2745
Cap133	0.000	1.00	1.3651	0.000	1.00	0.4516
Cap134	0.000	1.00	0.3635	0.000	1.00	0.0594
CapA	0.037	1.00	16.3920	0.051	0.53	14.5881
CapB	0.327	0.63	19.6541	0.085	0.40	17.6359
CapC	0.091	0.00	17.4234	0.036	0.13	15.7685

values are known, it is really hard to hit the optima in every attempt of optimization. Since the main idea is to test the performance of *CPSO* and *DPSO* algorithm with *UFL* benchmark, the results of both algorithms are provided in Table 4 as the solution quality: *average relative percent error (ARPE), hit to optimum rate,* (HR) and *average computational processing time (ACPU)* in seconds. *ARPE* is the percentage of difference from the optimum and defined as following:

$$\mathrm{ARPE} = \sum_{i=1}^{R} \left(\frac{H_i - U}{U} \right) \times \frac{100}{R}, \qquad (16)$$

where H_i denotes the ith replication solution value, U is the optimal value provided in the literature, and R is the number of replications. HR provides the ratio between the number of runs yielded the optimum and the total numbers of experimental trials.

Obviously, the higher the HR the better quality of solution, while the lower the *ARPE* the better quality. The computational time spent for *CPSO* [33] and *DPSO* cases are obtained as time to get best value over 1000 iterations, while for $CPSO_{LS}$ and $DPSO_{LS}$ cases are obtained as time to get best value over 250 iterations. All algorithms and other related software were coded with Borland C++ Builder 6 and run on an Intel Centrino 1.7 GHz PC with 512 MB memory.

There are fewer parameters used for the *DPSO* and $DPSO_{LS}$ algorithms and they are as follows: the size of the population (swarm) is the number of facilities, the social and

Table 6: Summary of results gained from different algorithms for comparison.

	Deviation from optimum [33]			Average CPU		
Problem	GA [21]	ESA [16]	$DPSO_{LS}$	GA [21]	ESA [16]	$DPSO_{LS}$
Cap71	0.00	0.00	0.000	0.287	0.041	0.0130
Cap72	0.00	0.00	0.000	0.322	0.028	0.0078
Cap73	0.00033	0.00	0.000	0.773	0.031	0.0203
Cap74	0.00	0.00	0.000	0.200	0.018	0.0109
Cap101	0.00020	0.00	0.000	0.801	0.256	0.1505
Cap102	0.00	0.00	0.000	0.896	0.098	0.0557
Cap103	0.00015	0.00	0.000	1.371	0.119	0.1693
Cap104	0.00	0.00	0.000	0.514	0.026	0.0344
Cap131	0.00065	0.00008	0.000	6.663	2.506	0.4922
Cap132	0.00	0.00	0.000	5.274	0.446	0.2745
Cap133	0.00037	0.00002	0.000	7.189	0.443	0.4516
Cap134	0.00	0.00	0.000	2.573	0.079	0.0594
CapA	0.00	0.00	0.051	184.422	17.930	14.5881
CapB	0.00172	0.00070	0.085	510.445	91.937	17.6359
CapC	0.00131	0.00119	0.036	591.516	131.345	15.7685

```
Begin
Set globalbest open facility vector (Yg)
to s0 (for DPSO_LS)
Set globalbest position vector (Xg)
to s0 (for CPSO_LS)
Modify s0 based on η, κ and set to s
Set 0 to loop
Repeat:
   Apply Flip to s and get s1
   if (f(s1) ≤ f(s0))
   Replace s with s1
   else
   loop = loop + 1
Until loop < n is false.
if (f(s) ≤ f(s0)) Replace Yg with s (for DPSO_LS)
if (f(s) ≤ f(s0)) Replace Xg with s (for CPSO_LS)
End.
```

Algorithm 3: Pseudocode for local search.

cognitive probabilities, c_1 and c_2, are set as $c_1 = c_2 = 0.5$, and inertia weight, w, is set to 0, 9. Each problem solution run is conducted for 30 replications. There were two termination criteria that have been applied for every run: first, one is getting the optimum solution, the other is reaching the maximum iteration number that is chosen for obtaining the result in a reasonable *CPU* time.

The performance of *CPSO* algorithm looks not very impressive as the results produced within the range of time over 1000 iterations. The *CPSO* search found 6 optimal solutions, whereas the *DPSO* algorithm found 12 among 15 benchmark problems. The *ARPE* index which is expected lower for good solution quality is very high for *CPSO* when applied CapA, CapB, and CapC benchmarks and none of the attempts for these benchmarks hit the optimum value. As come to the *ARPE* index of *DPSO*, it is better than the *ARPE* index of *CPSO,* but not satisfactory as expected. In term of *CPU*, *DPSO* is better than *CPSO* as well. When the results are investigated statistically with using the t-test with 99% levels of confidence, the *DPSO* produced significantly better fitness results than *CPSO* when CapA, CapB, and CapC fitness results are excluded. It may be possible to improve the solutions quality by carrying on with algorithms for a further number of iterations, but, then the main idea and useful motivation of employing the heuristics, that is, getting a better quality within shorter time, will be lost. This fact imposed that it is essential to empower *CPSO* and *DPSO* with hybridizing with a local search algorithm. Thus a simple local search algorithm is employed in this case for that purpose, as mentioned before.

The results of $CPSO_{LS}$ and $DPSO_{LS}$ are shown in Table 5. The performance of $CPSO_{LS}$ and $DPSO_{LS}$ looks very impressive compared to the *CPSO* and the *DPSO* algorithms with respect of all three indexes. *HR* is 1.00 which means 100% of the runs yield with optimum for all benchmark except CapB and CapC for $CPSO_{LS}$ and except CapA, CapB, and CapC for $DPSO_{LS}$. On the other hand, it should be mentioned that $DPSO_{LS}$ found optimum solutions of all instances, while $CPSO_{LS}$ found optimums except for CapC. The *ARPE* index results of $CPSO_{LS}$ and $DPSO_{LS}$ are very small for both algorithms and very similar to each other thus there is no meaningful difference. When the results are compared statistically with using the t-test with 99% levels of confidence, the $CPSO_{LS}$ and $DPSO_{LS}$ can be considered as equal. In term of *CPU*, $CPSO_{LS}$ consumed 18% more time than $DPSO_{LS}$ thus we can say that the results of $DPSO_{LS}$ are slightly better than $CPSO_{LS}$.

The experimental study is also carried out as a comparative work in Table 6. A genetic algorithm (*GA*) introduced by Jaramillo et al. [21] and an evolutionary simulated annealing algorithm (*ESA*) proposed by Aydin and Fogarty [16] are taken for the comparison. These algorithms were coded and run under the same condition with the $DPSO_{LS}$ algorithm. The performance of $DPSO_{LS}$ algorithm looks slightly better than *GA* in both two indexes. Especially, in respect of *CPU* time $DPSO_{LS}$ much more better than *GA*. Comparing with *ESA,* both algorithms deviations from optimum are very similar. However, especially for CapA, CapB, and CapC; *GA* and *ESA* consume more *CPU* time than $DPSO_{LS}$ algorithm.

4. CONCLUSION

In this paper, one of the recent metaheuristics algorithms called *DPSO* is applied to solve *UFL* benchmark problems. The algorithm has been tested on several benchmark problem instances and optimal results are obtained in a reasonable computing time. The results of *DPSO* with local search ($DPSO_{LS}$) are compared with results of a *CPSO* [33] with local search ($CPSO_{LS}$) and two other metaheuristics approaches, namely, *GA* [21] and *ESA* [16]. It is concluded that the $DPSO_{LS}$ is slightly better than the $CPSO_{LS}$ and competitive with GA and ESA.

The main purpose of this paper is testing performance of *CPSO* and *DPSO* algorithms under the same condition. When compared *CPSO*, *DPSO* proves to be a better algorithm for *UFL* problems. It also should be noted that, since *CPSO* considers each particle based on three key vectors; position (X_i), velocity (V_i), and open facility (Y_i). So, *CPSO* allocates more memory than *DPSO* for each particle. In addition, to the best our knowledge, this is the first application of discrete *PSO* algorithm applied to the *UFL* problem.

REFERENCES

[1] D. Simchi-Levi, P. Kaminsky, and E. Simchi-Levi, *Designing and Managing the Supply Chain, Concepts, Strategies and Case Studies*, McGraw-Hill, Boston, Mass, USA, 2000.

[2] D. Ghosh, "Neighborhood search heuristics for the uncapacitated facility location problem," *European Journal of Operational Research*, vol. 150, no. 1, pp. 150–162, 2003.

[3] V. L. Beresnev, E. H. Gimady, and V. T. Dementev, *Extremal Problems of Standardization*, Nauka, Novosibirsk, Russia, 1978.

[4] E. H. Gimady, *The Choice of Optimal Scales in One Problem Class of Location or Standartization*, Manage Systems, Nauka, Novosibirsk, Russia, 6th edition, 1970.

[5] G. Corneuéjols, M. L. Fisher, and G. L. Nemhauser, "Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms," *Management Science*, vol. 23, no. 8, pp. 789–810, 1977.

[6] B. M. Khumawala, "An efficient branch and bound algorithm for the warehouse location problem," *Management Science*, vol. 18, no. 12, pp. B718–B733, 1972.

[7] J. Krarup and P. M. Pruzan, "The simple plant location problem: survey and synthesis," *European Journal of Operation Research*, vol. 12, no. 1, pp. 36–81, 1983.

[8] P. B. Mirchandani and R. L. Francis, Eds., *Discrete Location Theory*, Wiley-Interscience, New York, NY, USA, 1990.

[9] A. Klose, "A branch and bound algorithm for an *UFLP* with a side constraint," *International Transactions in Operational Research*, vol. 5, no. 2, pp. 155–168, 1998.

[10] T. J. Van Roy, "A cross decomposition algorithm for capacitated facility location," *Operations Research*, vol. 34, no. 1, pp. 145–163, 1986.

[11] J. Barcelo, Å. Hallefjord, E. Fernandez, and K. Jörnsten, "Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing," *OR Spektrum*, vol. 12, no. 2, pp. 79–88, 1990.

[12] D. Erlenkotter, "A dual-based procedure for uncapacitated facility location," *Operations Research*, vol. 26, no. 6, pp. 992–1009, 1978.

[13] M. Köerkel, "On the exact solution of large-scale simple plant location problems," *European Journal of Operational Research*, vol. 39, no. 2, pp. 157–173, 1989.

[14] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey, "The uncapacitated facility location problem," in *Discrete Location Theory*, pp. 119–171, John Wiley & Sons, New York, NY, USA, 1990.

[15] M. L. Alves and M. T. Almeida, "Simulated annealing algorithm for the simple plant location problem: a computational study," *Revista Investigaćão Operacional*, vol. 12, 1992.

[16] M. E. Aydin and T. C. Fogarty, "A distributed evolutionary simulated annealing algorithm for combinatorial optimisation problems," *Journal of Heuristics*, vol. 10, no. 3, pp. 269–292, 2004.

[17] K. S. Al-Sultan and M. A. Al-Fawzan, "A tabu search approach to the uncapacitated facility location problem," *Annals of Operations Research*, vol. 86, pp. 91–103, 1999.

[18] L. Michel and P. Van Hentenryck, "A simple tabu search for warehouse location," *European Journal of Operational Research*, vol. 157, no. 3, pp. 576–591, 2004.

[19] M. Sun, "Solving the uncapacitated facility location problem using tabu search," *Computers & Operations Research*, vol. 33, no. 9, pp. 2563–2589, 2006.

[20] J. Kratica, D. Tošic, V. Filipović, and I. Ljubić, "Solving the simple plant location problem by genetic algorithm," *RAIRO Operations Research*, vol. 35, no. 1, pp. 127–142, 2001.

[21] J. H. Jaramillo, J. Bhadury, and R. Batta, "On the use of genetic algorithms to solve location problems," *Computers & Operations Research*, vol. 29, no. 6, pp. 761–779, 2002.

[22] M. Gen, Y. Tsujimura, and S. Ishizaki, "Optimal design of a star-LAN using neural networks," *Computers & Industrial Engineering*, vol. 31, no. 3-4, pp. 855–859, 1996.

[23] S. Vaithyanathan, L. I. Burke, and M. A. Magent, "Massively parallel analog tabu search using neural networks applied to simple plant location problems," *European Journal of Operational Research*, vol. 93, no. 2, pp. 317–330, 1996.

[24] R. C. Eberhart and J. Kennedy, "New optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science (MHS '95)*, pp. 39–43, Nagoya, Japan, October 1995.

[25] F. Van den Bergh and A. P. Engelbecht, "Cooperative learning in neural networks using particle swarm optimizers," *South African Computer Journal*, vol. 26, pp. 84–90, 2000.

[26] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems*, vol. 26, no. 8, pp. 363–371, 2002.

[27] A. Allahverdi and F. S. Al-Anzi, "A *PSO* and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application," *Computers & Operations Research*, vol. 33, no. 4, pp. 1056–1080, 2006.

[28] M. F. Tasgetiren, Y.-C. Liang, M. Sevkli, and G. Gencyilmaz, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop

sequencing problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1930–1947, 2007.

[29] R. C. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," in *Evolutionary Programming VII*, vol. 1447 of *Lecture Notes in Computer Science*, pp. 611–616, Springer, Berlin, Germany, 1998.

[30] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks (ICNN '95)*, vol. 4, pp. 1942–1948, Perth, Australia, November-December 1995.

[31] S. Naka, T. Genji, T. Yura, and Y. Fukuyama, "A hybrid particle swarm optimization for distribution state estimation," *IEEE Transactions on Power Systems*, vol. 18, no. 1, pp. 60–68, 2003.

[32] H. Yoshida, K. Kawata, Y. Fukuyama, and Y. Nakanishi, "A particle swarm optimization for reactive power and voltage control considering voltage stability," in *Proceedings of the International Conference on Intelligent System Application to Power System (ISAP '99)*, pp. 117–121, Rio de Janeiro, Brazil, April 1999.

[33] M. Sevkli and A. R. Guner, "A continuous particle swarm optimization algorithm for uncapacitated facility location problem," in *Ant Colony Optimization and Swarm Intelligence*, vol. 4150 of *Lecture Notes in Computer Science*, pp. 316–323, Springer, Berlin, Germany, 2006.

[34] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC '97)*, vol. 5, pp. 4104–4108, Orlando, Fla, USA, October 1997.

[35] P.-Y. Yin, "A discrete particle swarm algorithm for optimal polygonal approximation of digital curves," *Journal of Visual Communication and Image Representation*, vol. 15, no. 2, pp. 241–260, 2004.

[36] C.-J. Liao, C.-T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization for flowshop scheduling problems," *Computers & Operations Research*, vol. 34, no. 10, pp. 3099–3111, 2007.

[37] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem," *Computers & Operations Research*, vol. 35, no. 9, pp. 2807–2839, 2008.

[38] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*, Morgan Kaufmann, San Francisco, Calif, USA, 2001.

[39] J. E. Beasley, "*OR*-Library," 2005, http://people.brunel.ac.uk/mastjjb/jeb/info.html.