Discrete Particle Swarm Optimization for the Orienteering Problem

Zülal Sevkli and F. Erdoğan Sevilgen

Abstract—**In this paper a novel discrete Particle Swarm Optimization (PSO) algorithm is proposed to solve the Orienteering Problem (OP). Discrete evolution is achieved by re-defining all operators and operands used in PSO. To obtain better results, Strengthened-PSO which improves both exploration and exploitation during the search process is employed for experimental evaluation. Our proposed algorithm either achieves or improves the best known solutions compared to previous heuristics for the OP.**

I. Introduction

THE Orienteering Problem (OP) is a subset selection version of well-known Traveling Salesman Problem with profits. The objective of the Orienteering Problem is to construct a path starting at an origin and ending at a destination that maximizes the total profit without violating prescribed limits. The problem is inspired from and named after an outdoor sport usually played on mountains or forest areas.

The Orienteering Problem can be described in detail as follows: Let G= (N, E) be a graph where N denotes the set of nodes (control points) and E denotes the set of edges between the nodes in N. Each node n_i in N ($1 \le i \le k$) has a score $s_i \ge 0$ and the scores of n_1 (the origin) and n_k (the destination) are 0; i.e., $s_1=s_k=0$. Each edge between n_i and n_j has a cost $d_{i,j}$ associated with it. The objective of the OP is to find an acyclic path from n_1 to n_k that maximizes the total score of the nodes on the path without violating a cost constraint (e.g., the total cost of the edges on the path should be less than a specified limit, Tmax.) Because of the limitation, the path does not necessarily visit all control points. Golden et al. [1] prove that the OP is NP-hard. The mathematical model of the OP can be found in [2].

In this paper, we consider the Euclidean OP where nodes represent points in the Euclidean plane. In such a problem, the graph is a complete graph and the score of an edge in the graph is the Euclidean distance between the nodes.

The OP is used to model many practical problems such as inventory routing, customer or vehicle assignment [3] and production scheduling [4]. Additionally, a variation of the OP with time windows has applications to problems including postal delivery or school bus routing [5].

Zülal Sevkli is with the Computer Engineering Department, Fatih University, Büyükçekmece, Istanbul 34500 Turkey, (phone: +90-212-866-33-00 fax:+90-212-866-34-12 e-mail: zsevkli@fatih.edu.tr)

Fatih Erdoğan Sevilgen is with the Computer Engineering Department, Gebze Institute of Technology, Gebze, Kocaeli, Turkey, (email: sevilgen@gyte.edu.tr)

In literature, exact methods, problem several specific heuristics and some meta-heuristics have been used to solve the OP. A comprehensive survey of methods used for the OP is presented in [6]. Meta-heuristics successfully applied to the OP include population-based methods such as genetic algorithm [7], ant colony [8] and particle swarm optimization [15] and trajectory-based methods such as variable neighborhood search [6] and tabu search [8]. Their results are compared with the best known results for 67 benchmark problems from the four test sets presented in [9].

In this paper, we present an algorithm based on Particle Swarm Optimization (PSO) to solve the OP. PSO is a meta-heuristic method initially proposed for solving continuous optimization problems by Eberhart and Kennedy [10]. Later, it has been applied to combinatorial optimization problems as well. There are two popular approaches for application of PSO to discrete problems: One is to incorporate an extra conversion process in order to transform continuous PSO solution space to discrete problem solution space [11], [14], [15]. In this approach, both solution quality and run-time performance would be hindered since the algorithm always switches between two different search spaces. The second approach is to redefine all operators and components of the PSO in order to perform discrete processing [12], [13]. Although, the adaptation is more challenging (several operators and components should be redefined) there is a consistency between solution space and algorithm's search domain. Our algorithm is based on this second approach; we propose a simple but effective discrete operations for evaluations in PSO. Moreover, we used the operations in a powerful variant of PSO, the Strenghtened-PSO (StPSO) which is proposed in [15]. In StPSO, the general flow of PSO is preserved but both exploration and exploitation mechanisms are improved.

Discrete StPSO (DStPSO) is tested using 107 benchmark problems. It achieves the best known solutions for all instances within a reasonable amount of time. Moreover it finds a new best tour for one of these benchmark problems. The results show that our algorithm is a promising alternative not only for the OP but also for similar discrete problems.

II. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based method in which a swarm includes n individuals called particles. Each particle has a d-dimensional position vector representing a candidate solution and a d-dimensional velocity vector expressing the current tendency of the

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

particle during its search journey. The initial swarm can be constructed randomly or by using some predetermined values. At each step, the velocity of each particle is re-evaluated based on the particle's inertia as well as the social interaction (swarm's experience) and personal experience of the particle. The experience of each particle is usually captured by its local best position (pbest). The experience of the swarm is captured by the global best position (gbest). In the course of several iterations, particles make use of this experience and are supposed to move towards the optimum position.

```
 1 Procedure PSO
 2  Initialize Parameters
 3  Initialize Population
 4  For each particle
 5   Evaluate
 6   Update local best
 7  Update global best
 8  Do
 9   For each particle
10     Update velocity and position
11     Evaluate
12     Update local best
13   Update global best
14  While (Not Terminated)
15 End Procedure
```

Fig. 1. The pseudo-code of the standard PSO algorithm

Pseudo-code of the standard PSO algorithm is illustrated in Fig 1. Optimization is achieved in the course of several iterations of update-evaluate steps. During the update step (line 10), the velocity and the position vector of each particle are calculated by using the equations (1) and (2). In these equations, $v_{i,j}^t$ and $p_{i,j}^t$ are the velocity and the position values of the jth dimension ($1 \le j \le d$) of the ith particle ($1 \le i \le n$) at iteration *t*, respectively. The parameter w is the inertia weight. The inertia weight *w* serves as a balancing factor between exploration and exploitation. Large inertia weight, at the beginning of the search, enables more exploration, while smaller inertia weight facilitates more exploitation later. The parameters c_1 and c_2 are coefficients of learning factors, which are the weights for contributions of personal experience and social interaction. The stochastic behavior of PSO is achieved by random numbers r_1 and r_2 which are positive numbers generally uniformly distributed in [0, 1].

$$v_{i,j}^t = w v_{i,j}^{t-1} + c_1 r_1 (pbest_{i,j}^{t-1} - p_{i,j}^{t-1}) + c_2 r_2 (gbest_j^{t-1} - p_{i,j}^{t-1}) \quad (1)$$

$$p_{i,j}^t = p_{i,j}^{t-1} + v_{i,j}^{t-1} \quad (2)$$

After the update step, the fitness function value is calculated (line 11) for each particle based on its position (the candidate solution represented by the particle.) The local best position, pbest, of each particle (line 12) and the global best position, gbest, of the swarm (line 13) are updated if the candidate solution is better than pbest or gbest, respectively. The stopping condition (line 14) of the update-evaluate iterations is usually the attainment of a maximum number of iterations or a maximum number of iterations between two improvements.

A. Strenghtened PSO

Since PSO is introduced, it has been adapted to solve several optimization problems. Meanwhile some shortcomings like premature convergence or lack of intensification around the local best locations are observed [22], [23]. To improve search efficiency and rectify these deficiencies in the standard algorithm, researches proposed several modifications to PSO.

One of the latest improved PSO versions is Strengthened-PSO (StPSO). Pseudo-code of the StPSO algorithm is illustrated in Fig 2. Main focus of StPSO is on pioneering-particles which achieves or enhances the swarm's experience. These particles are assumed to be either converged or potentially converged. They are processed in two steps: Firstly, an external local search is initiated for each pioneering-particle (line 11). After the local search, only the local experience of each pioneering-particle are updated. Swarm experience is not changed by using the local search result. This step strengthens exploitation mechanism in PSO. Secondly, at the same iteration, the random moving strategy is performed on each pioneering particle in order to force the particle to continue exploration with its past experience (line 13). In this way, the exploration mechanism of PSO is improved and premature convergence is largely avoided. The other particles continue to search the solution space as in the standard PSO algorithm.

```
 1 Procedure StPSO
 2  Initialize Parameters
 3  Initialize Population
 4  For each particle
 5    Evaluate
 6    Update local best (lbest)
 7  Update global best (gbest)
 8  Do
 9   For each particle
10      If (fitness equal to fitness of gbest)
11        position←LocalSearch(particle)
12        Update local best
13        Assign new random position
14      Else
15        Update velocity
16        Update position
17      Evaluate
18      Update local best
19   Update global best
20  While (Not Terminated)
21 End Procedure
```

Fig. 2. The pseudo-code of the StPSO algorithm

B. Local Search

Similar to the original StPSO algorithm, we employ Reduce Variable Neighborhood Search (RVNS) in StPSO as the local search method. RVNS is a variation of the Variable Neighborhood Search (VNS) which is introduced as an

optimization method for combinatorial optimization problems [16]. In VNS, solution space is searched with a systematic change of neighborhood. It has two main steps named LocalSearch and Shake. VNS becomes RVNS if LocalSearch step is removed. RVNS can be used both as a general optimization method and as a local search for problems where exhaustive local search is expensive [15], [17].

The pseudo-code of RVNS is given in Fig. 3, where X_k (line 2) represents k^{th} neighborhood structure $X_k(s)$ represents the set of solutions in the k^{th} neighborhood of the solution s. Starting from a solution and the first neighborhood (s and X_1, respectively), during each iteration of the inner loop, a random solutions s' is selected from the current neighborhood (line 7). If s' is better than s, it replaces s and the search continues with the first neighborhood (line 8-10). Otherwise, the algorithm switches to the next neighborhood structure (line 12). If all neighborhood structures are exhausted, the inner loop is initiated all over again starting from the first neighborhood. The outer loop is repeated until a stopping condition is met. The maximum number of iterations or the maximum number of iterations between two improvement can be used as a stopping condition.

Both the choice and the order of neighborhood structures are critical for the performance of the RVNS algorithm. If they are large, the algorithm explores almost entire solution space quickly with big jumps. On the other hand, narrow neighborhood sizes lead to small footsteps which in turn results in fast exploitation ability.

```
 1 Procedure RVNS_LocalSearch (particle p)
 2  Define neighborhood structures Xk
    (k=1,...,kmax)
 3  Use position of p as initial solution s
 4  while stopping condition is not met do
 5     k ← 1
 6     while k ≤ kmax do
 7       s' ← Shake(s), s'∈  Xk(s)
 8       if (Fitness(s') > Fitness(s))
 9          s ← s'
10          k ← 1
11       else
12          k ← k+1
13     end-while
14  end-while
15 End-Procedure
```

Fig. 3. The pseudo-code of the RVNS algorithm

III. Discrete Particle Swarm Optimization for OP

In this section, we describe our application of Discrete StPSO (DStPSO) to solve the OP. In DStPSO, the position and velocity vector and all continuous operators are redefined in order to make processing in discrete OP solution space.

Both position and velocity of a particle are lists including control points. The length of a list is variable. The position list represents a candidate OP solution (a path). The path starts from the starting point, visits all control points in the position list, in order, and ends at the destination point. The position list is generated so that the represented solution is always feasible i.e., the total cost of the edges on the path is less than Tmax. The velocity list of a particle includes some of the control points which are not in the particle's position list. The change in the position by the velocity is performed by inserting the control points in the velocity list into the position list. So, velocity lists and position lists are always kept as disjoint lists.

The DStPSO algorithm preserves the general flow of the StPSO which is illustrated in Fig 2. The algorithm starts with initialization of the particles in the swarm. The position list and velocity list of each particle are initialized by using a random permutation of control points. To obtain a position list, starting from the first control point in the permutation, the control points are inserted between the starting point and the destination point, one by one, until the prescribed cost limit is exceeded. In other words, let $(x_{i_1}, x_{i_2}, ..., x_{i_d})$ be a permutation of control points, the position list obtained from the permutation is $(x_{i_1}, x_{i_2}, ..., x_{i_z})$ and velocity list is $(x_{i_{z+1}}, x_{i_{z+2}}, ..., x_{i_d})$ where x_{i_z} $(z \le d)$ is a cut point. Note this generation method produces a feasibile solution $(c_{0i_1} + c_{i_1 i_2} + c_{i_2 i_3} + ... + c_{i_{z-1} i_z} + c_{i_z d+1} \le T_{max})$ which has the maximum score $(\sum_{j=1}^{z} s_{i_j})$ amongst all possible cut points.

A. Discrete Operators

In DStPSO, equations (1) and (2) which are used in standard PSO are redefined as well. New velocity and position update equations are presented in the following.

$$v_i^t = \left[w_i \otimes v_i^{t-1}\right] \oplus \left[(c_1 \otimes (pbest_i^{t-1} \Theta p_i^{t-1})) \oplus (c_2 \otimes (gbest^{t-1} \Theta p_i^{t-1}))\right] \quad (3)$$

$$p_i^t = p_{i,}^{t-1} \circ v_i^t \quad (4)$$

$$v_i^t = P_{all} \Theta p_i^t \quad (5)$$

In (3), the velocity is re-calculated based on particle's inertia as well as gbest and pbest by using new operators. This velocity is employed in (4) to obtain the new position. For the next iteration, velocity of the particle is re-evaluated (5) by using P_{all} and new position. P_{all} is a permutation of all control points in the problem set. The new operators which are used in above equations are explained in the following.

Θ operator: The operator calculates the difference between two positions. Let p_1 and p_2 be two position lists. The result $(p_1 \Theta p_2)$ is a velocity list. The velocity list includes control points which are in p_1 but not in p_2. If p_1 is *gbest*, the control points in the new velocity vector have high priority (HP), If p_1 is *pbest*, the control points in the velocity vector have low priority (LP). Otherwise the control points have no priority (NP).

⊗ operator: This operator generates a new velocity by randomly selecting some control points from a velocity list. The left operand is a co-efficient ($0<c<1$) and the right operand is a velocity list (v). The co-efficient represents the probability of a control point in v to stay in the resulting velocity list. The result can be calculated as follows; Starting from the first control point in v, a random number which is distributed uniformly in the range [0, 1] is generated for each control point. If the random number is less than the c, the control point is appended the resulting velocity list. The operation does not alter the priority attributes.

⊕ operator: This operator combines two velocity lists. Remember that a velocity list includes a priority attribute, high-priority (HP), low-priority (LP) or non-priority (NP). The operation is performed based on priority values of its operands. If an operand has high priority and the other has low priority, the result ($v_{HP} \oplus v_{LP}$) is obtained by placing the control points that are in the intersection of these lists at the beginning, then remaining control points in v_{HP} then the ones in v_{LP}. If one of the operands has no priority, the result ($v_{HP/LP} \oplus v_{NP}$) includes the control points in $v_{HP/LP}$ at the beginning and remaining control points in v_{NP} are appended at the end.

◦ operator: This operator performs position update. It takes a position list and a velocity list as operands. The velocity list includes the control points which are to be inserted into the position list. We apply *node insert for increasing profit* (explained in next section) for each control point in the velocity list.

Example:

1	p_{all}	$=(2\,3\,4\,5\,6\,7\,8)$
2	p_i^t	$=(4\,7)$
3	$v_{i\_NP}^t$	$=(2\,3\,5\,6\,8)$
4	$pbest_i^t$	$=(3\,7\,6)$
5	$gbest^t$	$=(8\,6\,4\,2)$
6	c_1	$=0.5$
7	c_2	$=0.5$
8	w	$=0.7$
9	$v_{pbest\_LP}^{t+1}$	$=(pbest_i^{t-1} \Theta p_i^{t-1})$
10	$v_{pbest\_LP}^{t+1}$	$=(3\,7\,6)\Theta(4\,7)=(3\,6)$
11	$v_{gbest\_HP}^{t+1}$	$=(gbest_i^{t-1} \Theta p_i^{t-1})$
12	$v_{gbest\_HP}^{t+1}$	$=(8\,6\,4\,2)\Theta(4\,7)=(8\,6\,2)$
13	$v_{i\_NP}^{t+1}$	$=\left[0.7 \otimes v_{i\_NP}^t\right] \oplus \left[(0.5 \otimes v_{pbest\_LP}^{t+1}) \oplus (0.5 \otimes v_{gbest\_HP}^{t+1})\right]$
14	$v_{i\_NP}^{t+1}$	$=(3) \oplus [(3\,6) \oplus (8\,6)]$
15	$v_{i\_NP}^{t+1}$	$=(3) \oplus (6\,8\,3)$
16	$v_{i\_NP}^{t+1}$	$=(6\,8\,3)$

Fig. 4. A numerical example of the velocity update in DStPSO

A numerical example of the equation (3) is presented in Fig 4. In the example, the set N includes 9 control points {1,2,3,…,9}. The control points, 1 and 9, are the starting and destination points, respectively. The values of lists during t^{th} iteration of the DStPSO algorithm are given in lines 1-5. The values of the co-efficients (c_1, c_2 and w) are listed in line 6-8. The random selection of control points is performed in line 13. The random values for the control points in $v_{i\_NP}^t$ are (0.75, 0.30, 0.80, 0.94, 0.88), the result of $\left[0.7 \otimes v_{i\_NP}^t\right]$ is calculated as follows: For each control points in $v_{i\_NP}^t$, its random number is compared with 0.7. If the random number is less than 0.7, this control point is appended the new velocity list. Because only the random value of the control point 3, 0.30, is less than 0.7, the new velocity list is (3). The two other velocity lists are generated in similar way using random sequences (0.45, 0.3) and (0.1, 0.4, 0.8). Lastly, all three velocity lists are merged by using ⊕ operator (in lines 15-16).

B. Neighborhood Structures

This sub-section presents the neighborhood structures employed in local search part (RVNS) of StPSO. Note that RVNS accepts the pinoeering particle as an initial solution. In the following, neighborhood structures are explained in the same order as they are used in RVNS.

Node Insert for increasing profit:

In this structure, a random control point which is outside the solution is selected and it is inserted in the solution between starting and destination points using the cheapest insertion method. If the resulting solution satisfies the cost constraint, a new solution in the neighborhood is obtained. Otherwise, a control point which has a lower score than inserted control point and minimizes the cost when it is removed is selected. If removal of the control point makes the solution feasible (respect the cost constraint), the control point is removed and a new solution in the neighborhood is obtained after one insertion and one deletion. In either way, the feasible solution having more profit than the old one is generated and RVNS continues to search with this neighborhood. If a feasible solution is not obtained, RVNS switches the following neighborhood.

Node Insert for decreasing cost:

The aim of this structure is to decrease the cost of the solution. A random control point in the solution is re-inserted into the solution using the cheapest insertion method. If the total cost of the solution is decreased, a new solution in the neighborhood is obtained succesfully and RVNS continues with the first neighborhood. If THRESHOLD times back to back no improvement is observed from first two neighborhoods, RVNS switches the following neighborhood.

Path Invert:

This neighborhood is also called 2-opt move. In travelling salesman based problems, 2-opt local search algorithm is frequently used in order to decrease lenght of the tour. In 2-

opt local search algorithm, a route crossing over itself is reordered to make it a 2-opt tour (cross-free). The basic operation, 2-opt move, is to cut two edges in the tour and reconnect the resulting paths to form a new tour. Since the same efect can be obtaıned by inverting a subsequence of control points on the tour, we call this neighborhood as path invert. In this neighborhood structure, path invert is employed on the solution path many times until a shorter path is obtained.

IV. Experimental Results

DStPSO algorithm is tested on 107 benchmark problems. Problems are categorized into six datasets. Problems in the same dataset share the same control points with different T_{max} value. The properties of the datasets are given in Table 1. First three datasets are provided by Tsiligirides [18] and the others are provided by Chao [9].

Experiments are performed on an Intel P4 2.8 GHz PC with 1 GB of memory. In the experimentation, the swarm includes 10 particles. The parameter values are $c_1 = c_2 = 0.5$, the initial value of the inertia weight w for all particle is 0.7. At each iteration, inertia weight is decreased by 2%. DStPSO algorithm is terminated when *gbest* list has no improvement 300 times consecutively. For RVNS, the THRESHOLD parameter value is 10 and the stopping condition is back to back 20 times no improvement. The parameter values are determined experimentally.

TABLE I.
DATASETS PROPERTIES

	# of Points	# of Problems	$minT_{max}$	$maxT_{max}$
Dataset1	32	18	5	85
Dataset2	21	11	15	45
Dataset3	33	20	15	110
Dataset4	32	18	5	85
Dataset5	66	26	5	130
Dataset6	64	14	15	80
	Total	107		

Each problem is run 10 times and the results are compared based on computation time (CPU), relative percentage error (RPE) and standard deviation (Std. Dev.) in the repetitions. RPE is the error in the best solution through the repetitions with respect to the best known solution. It indicates whether an algorithm finds the best known solution through the repetitions

For the first four datasets, the results of the DStPSO are given in Table 2 with the results of the following algorithms:

- CGW: Heuristic algorithm [9] (experimented datasets 1 – 6 on a SUN 4/370 workstation).
- GA: Genetic algorithm [7] (experimented datasets 1 – 4 on an Intel P4 1.33 GHz PC with 256 MB Ram)

TABLE II.
COMPARISONS PREVIOUS STUDIES' RESULTS WITH DSTPSO RESULTS ON DATASET1-4

	Algorithms	Dataset1	Dataset2	Dataset3	Dataset4	Avg
RPE	CGW	0.00	0.00	0.00	0.11	0.03
	GA	0.00	0.00	0.00	0.00	0.00
	ACO	0.00	0.00	0.00	0.00	0.00
	VNS	0.00	0.00	0.00	0.00	0.00
	GLS	3.17	1.36	0.65	-	1.73
	StPSO	0.14	0.00	0.15	0.00	0.07
	DStPSO	0.00	0.00	0.00	0.00	0.00
Std. Dev.	CGW	-	-	-	-	-
	GA	1.29	2.63	3.38	1.36	2.17
	ACO	0.59	1.66	0.72	0.48	0.86
	VNS	2.73	2.49	6.04	2.82	3.52
	GLS	-	-	-	-	-
	StPSO	2.13	3.78	4.43	1.94	3.10
	DStPSO	0.00	0.34	0.00	0.13	0.12
CPU	CGW	19.46	5.07	18.39	16.75	14.92
	GA	21.81	10.65	27.44	21.93	20.46
	ACO	3.39	0.98	4.32	4.23	3.23
	VNS	0.92	1.09	0.88	0.91	0.95
	GLS	0.52	0.25	0.55	-	0.44
	StPSO	4.72	2.12	5.22	4.60	4.16
	DStPSO	4.00	0.92	6.69	4.13	3.94

- ACO: Ant colony optimization [19] (experimented datasets 1 – 4 on an Intel Celeron 433 MHz PC with 128 MB Ram)
- VNS: Variable neighborhood search [6] (experimented datasets 1 - 6 on an Intel P4 2.6 GHz PC with 512 MB RAM
- GLS: Guided local search [20] (experimented datasets 1, 2, 3, 5, 6 on an Intel P4 2.8 GHz PC with 1 GB RAM) (same configuration as ours)
- StPSO: Strengthened PSO with continuous operator [15] (experimented datasets 1 - 4 on an Intel P4 2.8 GHz PC with 1 GB RAM) (same configuration as ours)

Results in Table 2 indicate that, GLS is the fastest method in the expense of larger RPE values. Both GLS and CGW could not find the best known solution for several problems (GLS and CGW are failed to find optimum solutions for 14 and 3 of 67 problems, respectively).

Although StPSO follows the same algorithm, particles fly on the continuous search space. When fitness calculation is necessary, the particle's position is transformed into discrete solution. Because the search space does not match with the solution space, StPSO could not find the best known solution for some problem as well.

DStPSO achieves the best known solutions for all problems like the algorithms GA, ACO and VNS. Moreover, the robustness of DStPSO is better than the others (check std. dev. values). DStPSO is producing results closer to the best known solutions which make it a promising technique for solving the OP. The best results obtained from DStPSO are also compared with the best results of the previous studies for datasets 5 and 6. The comparison is given in Table 3 and Table 4. In the tables, the symbol "+" indicates that DStPSO produces a better score than the other algorithm. The symbol "-" means that a worse score is obtained. Empty cell means that DStPSO and competitor produce the same score.

TABLE III .
COMPARISONS PREVIOUS STUDIES' RESULTS WITH DSTPSO RESULTS ON DATASET5

	CGW		VNS		GLS		DStPSO		Comparision		
Tmax	Score	CPU	Score	CPU	Score	CPU	Score	CPU	CGW	VNS	GLS
5	10	1.1	10	3.3	10	0	10	0.2			
10	40	0.5	40	4.1	40	0.2	40	0.2			
15	120	4.3	120	11.3	120	1.3	120	1.0			
20	195	6.2	205	15.0	175	1.7	205	0.5	+		+
25	290	73.4	290	10.0	290	1.6	290	0.6			
30	400	54.8	400	7.4	400	2	400	0.6			
35	460	32.4	465	16.9	465	2.8	465	0.6	+		
40	575	98.9	575	54.2	575	3.2	575	14.6			
45	650	58.1	650	10.8	640	2.9	650	1.1			+
50	730	68.1	730	77.7	710	3	730	2.1			+
55	825	65.2	825	48.9	825	3.7	825	3.0			
60	915	84.6	915	107.2	905	4.1	915	1.8			+
65	980	82.2	980	67.7	935	3.3	980	4.6			+
70	1070	119	1070	56.5	1070	3.5	1070	6.1			
75	1140	116.7	1140	57.1	1140	4.4	1140	8.0			
80	1215	108.9	1215	125.2	1195	3.1	1215	24.7			+
85	1270	132.5	1270	118.3	1265	2.3	1270	7.1			+
90	1340	502.4	1340	166.7	1300	2.3	1340	16.6			+
95	1380	467.1	1395	147.6	1385	2.2	1395	22.1	+		+
100	1435	128.6	1465	180.2	1445	2.1	1465	37.6	+		+
105	1510	316.3	1520	336.3	1505	2.2	1520	89.1	+		+
110	1550	469.9	1560	276.5	1560	2	1560	45.1	+		
115	1595	474.6	1595	306.3	1580	1.7	1595	123.0			+
120	1635	358	1635	215.1	1635	1.2	1635	145.8			
125	1655	268.9	1670	193.8	1665	1.3	1665	153.0	+	-	
130	1680	32.1	1680	70.6	1680	0.7	1680	186.2			
Avg	948.6	158.6	952.3	103.2	942.8	2.2	952.1	34.4	7+	0+	12+
	Summary of DStPSO vs. Previous Heuristic Methods								0-	1-	0-

TABLE IV .
COMPARISONS PREVIOUS STUDIES' RESULTS WITH DSTPSO RESULTS ON DATASET6

	CGW		VNS		GLS		DStPSO		Comparision		
Tmax	**Score**	**CPU**	**Score**	**CPU**	**Score**	**CPU**	**Score**	**CPU**	**CGW**	**VNS**	**GLS**
15	96	13	96	85.8	96	0.8	96	0.8			
20	294	27.9	294	171.1	294	2	294	0.5			
25	390	238.9	390	255.7	390	2.5	390	0.7			
30	474	74.5	474	275.9	474	4.5	474	16.2			
35	570	139.8	576	342.7	552	2.3	576	26.6	+		+
40	714	137.9	714	406.8	702	2.5	714	21.6			+
45	816	205	816	467.8	780	3.2	816	19.1			+
50	900	231.6	900	513.6	888	2.1	900	24.8			+
55	984	246.2	984	570.6	972	1.4	984	15.8			+
60	1044	264.8	1062	599.4	1062	2.3	1062	92.4	+		
65	1116	232.6	1116	628.4	1110	1.8	1116	49.8			+
70	1176	231	1188	659.7	1188	1.5	1188	53.1	+		
75	1224	223.1	1236	687.3	1236	2.1	1236	57.3	+		
80	1272	212.3	1272	610.7	1260	1.5	**1284**	81.8	+	+	+
Avg	790.7	177.0	794.1	448.3	786.0	2.2	795.0	32.9	5+	1+	7+
	Summary of DStPSO vs. Previous Heuristic Methods								0-	0-	0-

DStPSO is superior for 19 problem instances in datasets 5 and 6 compared to recently published GLS. It produces better results than CGW for several problem instances, as well. The results obtained have almost the same quality as the results of the VNS. However, DStPSO achieves these results in a shorter CPU time. Moreover, an improved result for Tmax = 80 is obtained for dataset 6 by DStPSO. The tour solution in the improved result is as follows: The improved path: 1-2-4-7-12-17-11-16-22-29-37-44-50-55-59-56-51-45-38-30-23-31-24-18-13-19-14-10-15-21-28-36-43-35-42-49-54-48-41-34-27-20-26-33-25-32-39-46-40-47-52-57-53-58-61-63-6

V. CONCLUSION

In this paper, a PSO based algorithm for solving the OP problem is proposed. In our algorithm, discrete evaluation is achieved by new discrete operators defined in this paper. To achieve better results, new operators are employed in StPSO whose exploration and exploitation mechanisms are enhanced by re-initiating velocities and embedding a local search method, respectively. The performance of resulting Discrete StPSO (DStPSO) is examined based on solution quality and CPU time. For all 107 benchmark problems, DStPSO finds or improves the best known solutions by providing better robustness.

With superior performance on OP, DStPSO proves itself to be a promising candidate for solving subset selection type optimization problems where the solution is not necessarily includes all the objects in the problem. In future, we would like to investigate the applicability of DStPSO to such problems especially to the team orienteering problem.

REFERENCES

[1] B. Golden, L. Levy, R. Vohra, "The Orienteering Problem", Naval Res. Logist, Vol.34(3), pp. 307—318, 1987.

[2] A.C. Leifer, M.B. Rosenwein, "Strong Lineer Programming relaxations for orienteering problem", European Journal of Operation. Research, Vol.73, pp.517-523, 1994.

[3] B.L. Golden, A. Assad, R. Dahl, "Analysis of a large-scale vehicle routing problem with inventory component", Large Scale Systems, Vol. 7, pp. 181-190, 1984.

[4] E. Balas, "The prize collecting traveling salesman problem", Networks, Vol.19, pp. 621-636, 1989.

[5] M.G. Kantor, M.B. Rosenwein, "The orienteering problem with time windows", J. Oper. Res. Soc., Vol. 43(6), pp. 629-635, 1992.

[6] Z. Sevkli, E. Sevilgen, "Variable Neighborhood Search for the Orienteering Problem", LNCS, Vol. 4263, pp.134-143, 2006.

[7] F.M. Tasgetiren, A.E. Smith, "A genetic algorithm for the orienteering problem", Proceedings of the Congress on Evolutionary Computation (CEC'00), San Diego, CA, 2000.

[8] Y.-C. Liang, S. Kulturel-Konak, A.E. Smith, "Meta Heuristic For the Orienteering Problem" Proceedings of the Congress on Evolutionary Computation (CEC'02), Hawaii, 2002.

[9] I-M. Chao, B.L. Golden, E.A. Wasil, "A fast and effective heuristic for the orienteering problem", Eur. J. Oper. Res., Vol. 88(3), pp. 475-489, 1996.

[10] R.C. Eberhard, J. Kennedy, "New optimizer using Particle Swarm Theory", Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995.

[11] F.M. Tasgetiren, M. Sevkli, Y.-C. Liang, G. Gencyilmaz, "Particle swarm optimization algorithm for single machine total weighted tardiness problem", Proceedings of the Congress on Evolutionary Computation (CEC'04), Portland, Oregon, 2004.

[12] M.F Tasgetiren, P.N. Suganthan, Q.Q. Pan, "A discrete particle swarm optimization algorithm for the generalized traveling salesman problem", Proceedings of the 9th Annual Conference on Genetic and Evolutionary computation, GECCO '07, pp. 158-167, 2007.

[13] K. Rameshkumar, R.K. Suresh, K.M. Mohanasundaram, "Discrete Particle Swarm Optimization (DPSO) Algorithm for

Permutation Flowshop Scheduling to Minimize Makespan", ICNC, Vol.3, pp. 572-581, 2005

[14] Q. Zhu, L. Qian, Y. Li, S. Zhu, "An Improved Particle Swarm Optimization Algorithm for Vehicle Routing Problem with Time Windows", Proceedings of the Congress on Evolutionary Computation, (CEC'06), pp.1386 – 1390, 2006.

[15] Z Sevkli, E. Sevilgen, "StPSO: Strengthened Particle Swarm Optimization ", Turkish Journal Of Electrical Engineering & Computer Sciences, Vol. 18(3) (to be published)

[16] N. Mladenovic, P. Hansen, "Variable Neighborhood Search", Computers & Operations Research, Vol.24(11), pp.1097-1100, 1997.

[17] P. Hansen, N. Mladenovic, "Variable neighborhood search: principles and applications", European Journal of Operation. Research, Vol.130, pp. 449-467, 2001.

[18] T. Tsiligirides, "Heuristic methods applied to orienteering", Journal of Operation Research Society, Vol.35(9), pp. 797-809, 1984.

[19] Y.C. Liang, A.E. Smith. "An ant colony approach to the orienteering problem", Journal of the Chinese Institute of Industrial Engineers, Vol. 23(5), pp. 403-414, 2006.

[20] P. Vansteenwegen, W. Souffrşau, GV Berghe, DV Oudheusden, "A guided local search metaheuristic for team orienteering problem", Europen Journal of Operational Research, Vol. 196, pp 118-127, 2009.

[21] Y. Shi, R.C. Eberhart, "A modified particle swarm optimizer", Proceedings of IEEE Congress on Evolutionary Computation, pp. 69-73, 1998.

[22] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions", IEEE Transaction on Evolutionary Computation, Vol. 10(3), pp. 281-295, 2006.

[23] F. Van den Berg, A. Engelbrecht, "A new locally convergent particle swarm optimizer", Proceeding of IEEE Conference on System, Man and Cybernetics, pp. 96-101, 2002.