An effective PSO-inspired algorithm for the team orienteering problem

Duc-Cuong DANG[a,b,*], Rym Nesrine GUIBADJ[a,c], Aziz MOUKRIM[a]

[a] *Université de Technologie de Compiègne, Département Génie Informatique*
Heudiasyc, CNRS UMR 7253, BP 20529, 60205 Compiègne, France
[b] *University of Nottingham, School of Computer Science*
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom
[c] *VEOLIA Transport, MERCUR subsidiary*
32, boulevard Gallieni, 92442 Issy-les-Moulineaux, France

Abstract

The team orienteering problem (TOP) is a particular vehicle routing problem in which the aim is to maximize the profit gained from visiting customers without exceeding a travel cost/time limit. This paper proposes a new and fast evaluation process for TOP based on an interval graph model and a Particle Swarm Optimization inspired algorithm (PSOiA) to solve the problem. Experiments conducted on the standard benchmark of TOP clearly show that our algorithm outperforms the existing solving methods. PSOiA reached a relative error of 0.0005% whereas the best known relative error in the literature is 0.0394%. Our algorithm detects all but one of the best known solutions. Moreover, a strict improvement was found for one instance of the benchmark and a new set of larger instances was introduced.

Keywords: Vehicle routing, knapsack problem, interval graph, optimal split, swarm intelligence.

Introduction

The term Orienteering Problem (OP), first introduced in [19], comes from an outdoor game played in mountainous or forested areas. In this game, each individual player competes with the others under the following rules. Each player leaves a specific starting point and tries to collect as many rewards as possible from a set of check points in a given time limit before returning to the same starting point. Each check point can reward each player at most once and each player is aware of the position of each check point as well as the associated amount of rewards. There always exists an optimal strategy to achieve the maximum amount of rewards. In general, finding such a strategy (or solving OP) is NP-Hard [19], the player should select a correct subset of check points together with determining the shortest Hamiltonian circuit connecting these points and the starting point. OP and its variants have attracted a good deal of attention in recent years [1, 6, 35, 38] as a result of their practical applications [14, 19, 24, 36] and their hardness [11, 18, 22]. Readers are referred to Vansteenwegen et al. [39] for a recent survey of these problems.

Adding the cooperative aspect to OP, without neglecting the competitive one, yields to the Team Orienteering Problem (TOP) [13]. In this problem, the players are partitioned into teams and players of a team work together to collect as many rewards as possible within the time limit. Each check point can reward each team at most once. The specific vehicle routing problem, analogous to this game that we also denote by TOP, is the problem where a limited number of vehicles are available to visit customers from a potential set, the travel time of each vehicle being limited by a time quota, each customer having a specific profit and being visited at most once. The aim of TOP is to organize an itinerary of visits so as to maximize the total profit. Solving this problem is also NP-Hard [13]. The applications of TOP include athlete recruiting [13], technician routing [8, 33] and tourist trip planning [38, 39]. In this paper, we are interested in TOP as

*Corresponding author. E-mail: duc-cuong.dang@hds.utc.fr.

First version submitted on March, 2012

the core variant of OP for multiple vehicles. This work was motivated by several lines of research first put forward by Veolia Environnement [8, 9].

As far as we know, there are only three exact algorithms for TOP [10, 12, 27]. In contrast to exact solving approaches, a number of heuristics and metaheuristics have been developed for TOP. Two fast heuristics were developed by Butt and Cavalier [11] and by Chao et al. [14]. Tang and Miller-Hooks [33] proposed a tabu search embedded in an adaptive memory procedure. Two tabu search approaches and two versions of a Variable Neighborhood Search (VNS) algorithm were developed by Archetti et al. [2]. Those four methods make use of infeasible tours and of a repairing procedure. Among these, the slow version of the VNS (SVNS) gave very good results on the standard benchmark. Later, Ke et al. [21] developed four versions of an Ant Colony Optimization (ACO) approach. A guided local search and a skewed variable neighborhood search were then proposed by Vansteenwegen et al. [37, 38]. More recently, Bouly et al. [9] introduced giant tours, i.e. permutations of all customers, to represent solutions of TOP and designed an effective Memetic Algorithm (MA). The results of MA [9] were as good as those of SVNS [2] with several strict improvements. Souffriau et al. [32] submitted two versions of a Path Relinking (PR) approach and independently produced the strict improvements. Like [2], PR approach uses a repairing procedure during the relinking phase to deal with infeasible tours. Those tours are obtained from a gradual combination of each of the random generated solutions with the best ones. The slow version of the Path Relinking (SPR), despite its name, required very small computational times. It is also worth mentioning that Tricoire et al. [35] proposed a VNS algorithm for a generalized version of OP and provided their results on the original TOP instances. Furthermore, there are two methods based on Particle Swarm Optimization (PSO) designed to TOP: Bonnefoy [7] developed a PSO algorithm combined with a linear programming technique whereas Muthuswamy and Lam [25] introduced a discrete version of PSO (DPSO) to solve TOP.

In short, three methods stand out as the state-of-the-art algorithms for TOP: the slow version of the VNS (SVNS) in [2], the MA algorithm in [9] and the slow version of the PR (SPR) in [32]. Unlike the other two, MA proposed an interesting technique to represent the solutions of TOP, known as giant tours. This technique was previously introduced in [5] for the Vehicle Routing Problem (VRP). According to a recent survey on heuristic solutions for variants of VRP [40], it is classified as an indirect representation of the solution space. Indeed, each giant tour represents a neighborhood of solutions from which the best one can easily be extracted by an evaluation process. A heuristic using this representation tends to have better visions during the search and a better chance to reach the global optimum. Several search algorithms exploiting this strategy have been discussed in [28] for the case of VRP and variants.

In this paper, we propose an effective PSO-inspired algorithm (PSOiA) for TOP. This work is based on our preliminary study of a PSO-based memetic algorithm (PSOMA), which was communicated in [16]. The main contribution of our paper is a faster evaluation process than the one proposed in [9]. This enables PSOiA and possibly further methods in the literature to examine a larger number of neighborhoods and explore faster the search space. Experiments conducted on the standard benchmark of TOP clearly show that PSOiA outperforms the existing solution methods of the literature. It achieves a relative error of 0.0005% and detects all but one of the best known solutions. Moreover, a strict improvement was found for one instance of the benchmark. The remainder of this paper is organized as follows. Section 1 provides a formal formulation of TOP. PSOiA and the new optimal split procedure are described in Section 2. The dynamic management of the parameters and computational results on the standard benchmark are described in Section 3. In section 4, we introduce a new set of large instances and provide the respective results. Finally, some conclusions and further developments are discussed in Section 5.

1. Formulation of the problem

TOP is modeled with a graph $G = (V \cup \{d\} \cup \{a\}, E)$, where $V = \{1, 2, ..., n\}$ is the set of vertices representing customers, $E = \{(i, j) \mid i, j \in V\}$ is the edge set, d and a are respectively departure and arrival vertices for vehicles. Each vertex i is associated with a profit P_i, and each edge $(i, j) \in E$ is associated with a travel cost $C_{i,j}$ which is assumed to be symmetric and satisfying the triangle inequality. A tour R is represented as an ordered list of q customers from V, so $R = (R[1], \dots, R[q])$. Each *tour* begins at the departure vertex and ends at the arrival vertex. We denote the total profit collected from a tour R as

$P(R) = \sum_{i=1}^{i=q} P_{R[i]}$, and the total travel cost/time as $C(R) = C_{d,R[1]} + \sum_{i=1}^{i=q-1} C_{R[i],R[i+1]} + C_{R[q],a}$. A tour R is feasible if $C(R) \leq L$ with L being a predefined travel cost/time limit. The fleet is composed of m identical vehicles. A *solution* S is consequently a set of m (or fewer) feasible tours in which each customer is visited at most once. The goal is to find a solution S such that $\sum_{R \in S} P(R)$ is maximized. One simple way of reducing the size of the problem is to consider only *accessible* customers. A customer is said to be accessible if a tour containing only this customer has a travel cost/time less than or equal to L. For mixed integer linear programming formulations of TOP see [10, 12, 21, 27, 39].

2. A PSO-inspired algorithm

Particle Swarm Optimization (PSO) is a swarm intelligence algorithm proposed by Kennedy and Eberhart [23] with the basic idea of simulating the collective behavior of wild animals in the nature. PSO was first used for optimization problems in continuous space as follows. A set known as a *swarm* of candidate solutions, referred to as *particles*, is composed of positions in the search space. The swarm explores the search space according to Equations (1) and (2). In these equations, x_i^t and v_i^t are respectively the vectors of position and velocity of particle i at instant t. Three values w, c_1 and c_2, called respectively *inertia*, *cognitive* factor and *social* factor, are parameters of the algorithm. Two values r_1 and r_2 are random numbers generated in the interval $[0, 1]$. Each particle i memorizes its best known position up to instant t as x_i^{lbest}, and the best known position up to instant t for the swarm is denoted as x^{gbest}.

$$v_i^{t+1} = w \cdot v_i^t + c_1 \cdot r_1 \cdot (x_i^{lbest} - x_i^t) + c_2 \cdot r_2 \cdot (x^{gbest} - x_i^t) \quad (1)$$
$$x_i^{t+1} = x_i^t + v_i^{t+1} \quad (2)$$

With this design, PSO is highly successful at performing optimizations in continuous space [3, 20]. In contrast, when applied to problems of combinatorial optimization, PSO encounters difficulties in interpreting positions and velocities, as well as in defining position update operators. As a result, there are a variety of discrete PSO variants (DPSO) [4], and it is difficult to choose an appropriate variant for any given combinatorial optimization such as TOP.

2.1. Basic algorithm

Our PSO works with a population of particles, so called the *swarm* and denoted S. Each particle memorizes its current position, i.e. a representation of a solution, and its best known position, called *local best position*, according to an evaluation process. A basic iteration of the algorithm consists of updating the position of each particle in the swarm. In the standard PSO, this update is influenced by PSO parameters and it takes into account the current position, the local best position and the global best position. In our method, each particle also has a small probability ph to be moved out of its current position and transfered to a completely new position. This new position is generated using a randomized heuristic. Moreover, each new position has pm probability to be improved through a local search process. The algorithm is stopped after $itermax$ consecutive position updates have failed to give rise to new local best. Because $itermax$ is usually set to be proportional to $\frac{n}{m}$ [9, 16], then from now when we say the stopping condition is k, that means $itermax = k \cdot \frac{n}{m}$.

For convenience, the current, local best and global best positions of a particle x are denoted respectively $S[x].pos$, $S[x].lbest$ and $S[best].lbest$. The global scheme is summarized in Algorithm 1. Its components are detailed in the next sections.

2.2. Position representation and evaluation

A position in our PSO is a permutation π of all accessible customers, usually referred to as a *giant tour*, in a particular problem scenario. The principle of the *split* technique that optimally extracts a solution from a giant tour was introduced by Bouly et al. [9] for TOP. The basic idea is the following. All possible subsequences of π, denoted by $(\pi[i], \ldots, \pi[i + l_i])$ or $\langle i, l_i \rangle_\pi$ for short, that can form a feasible tour of TOP

Algorithm 1: Basic algorithm

```
Data: $S$ a swarm of $N$ particles;
Result: $S[best].lbest$ best position found;
begin
    initialize and evaluate each particle in $S$ (see Section 2.3);
    $iter \leftarrow 1$;
    while $iter \leq itermax$ do
        foreach $x$ in $[1..N]$ do
            if $rand(0,1) < ph$ then
                move $S[x]$ to a new position (see Section 2.3);
            else
                update $S[x].pos$ (see Section 2.5);
            if $rand(0,1) < pm$ then
                apply local search on $S[x].pos$ (see Section 2.4);
            evaluate $S[x].pos$ (see Section 2.2);
            update $lbest$ of $S$ (see Section 2.6);
            if (update Rule 3 is applied) (see Section 2.6) then
                $iter \leftarrow 1$;
            else
                $iter \leftarrow iter + 1$;
end
```

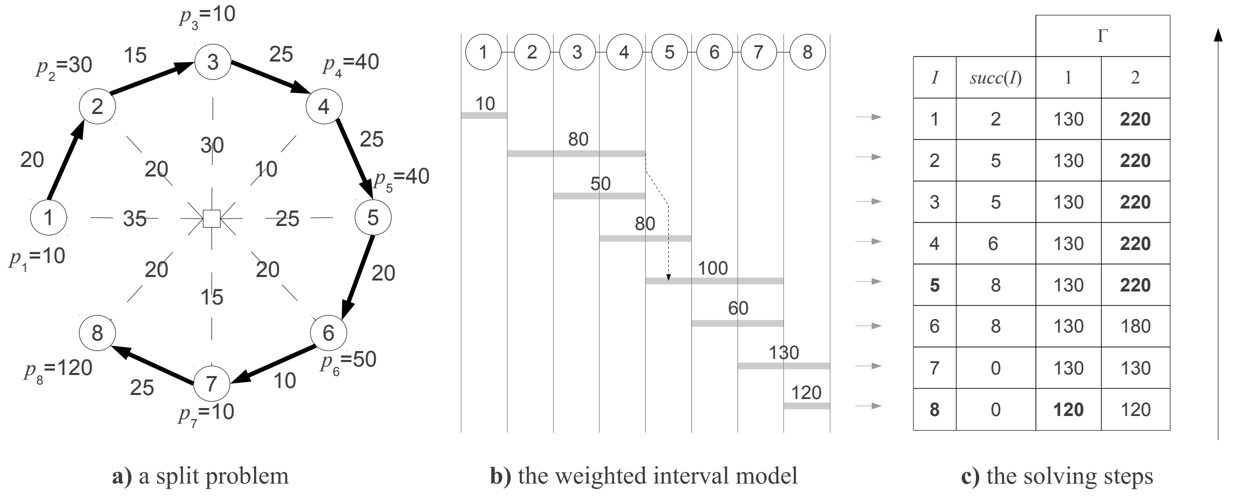
are considered. For convenience, we use the term *extracted tours* or simply tours in this section to refer to these subsequences. The goal of a *split* procedure is then to find a set of m distinct tours (without shared customer) such that the sum of their profits is maximized. Such a procedure guarantees that if a set of tours forming an optimal solution for the TOP is currently present as subsequences in a permutation π^*, the application of the split procedure on π^* will return the optimal TOP solution.

The authors of [9] proposed a split procedure for TOP. The algorithm requires to find the longest path in an acyclic auxiliary graph. This graph represents the *successor* relations between extracted tours, i.e. the possibility of a tour to follow another in a valid solution. They also introduced the notion of *saturated* tours, i.e. a tour in which l_i is maximal (denoted by l_i^{max}), and proved that solutions containing only saturated tours are dominant. Therefore, only *saturated* tours were considered in their procedure and the number of arcs in the acyclic graph is reduced. The *worst case* complexity of their procedure is $O(m \cdot n^2)$.

In this work, the limited number of saturated tours is exploited more efficiently to reduce the complexity of the evaluation process. Before going in the detail of our new split procedure, we recall the definition of a knapsack problem with conflicts (KPCG) [41] as follows. In a KPCG, we have a set of items to be put into a knapsack. A value and a volume are associated to each item. The knapsack has a limited volume, so it cannot generally hold all items. In addition to the knapsack volume, some items are in conflict with each other and they cannot be put in the knapsack together. The aim of the KPCG is to find a subset of items to fit into the knapsack such that the sum of their values is maximized. In such a problem, the conflicts between items are usually modeled with a graph, called *conflict graph*. We also recall the definition of an interval graph [34] as follows. A graph $G = (V, E)$ is called an interval graph if there is a mapping I from V to sets of consecutive integers (called *intervals*) such that for all i and j of V, $[i, j] \in E$ if and only if $I(i) \cap I(j) \neq \emptyset$. Then the following proposition holds for the split procedure of TOP.

Proposition 2.1. *The split procedure can be done optimally in $O(m \cdot n)$ time and space.*

Proof. Each possible tour extracted from a giant tour is in fact a set of positions of customers in the giant tour. Since these customers are adjacent in the giant tour, the positions are consecutive integers and the set of extracted tours can be mapped to the set of vertices of an interval graph X. Additionally, an edge of X (or



		Γ	
I	$succ(I)$	1	2
1	2	130	**220**
2	5	130	**220**
3	5	130	**220**
4	6	130	**220**
5	8	130	**220**
6	8	130	180
7	0	130	130
8	0	**120**	120

Figure 1: The new evaluation process for the same split problem described in [9] with 8 customers, $m = 2$ and $L = 70$.

a non-empty intersection between two sets of positions) indicates the presence of shared customers between the associated tours. As mentioned above, a split procedure looks for m tours without shared customer such that the sum of their profit is maximized. So this is equivalent to solve a knapsack problem with X as the conflict graph, a unitary volume for each item and m as the knapsack's volume. In this particular knapsack problem, the number of items is equal to the number of possible tours. This number is equal to n when only saturated tours are considered. Based on the work of Sadykov and Vanderbeck [30], we deduce that such a problem can be solved in $O(m \cdot n)$ time and space. □

Our new evaluation process is summarized as below. For each saturated tour starting with customer $\pi[i]$, we use $P[i]$ to denote the sum of profits of its customers. Its *first successor*, denoted by $succ[i]$, is computed as follows:

$$succ[i] = \begin{cases} i + l_i^{max} + 1 & \text{if } i + l_i^{max} + 1 \leq n \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

A two-dimensional array Γ of size $m \cdot n$ is used to memorize the maximum reachable profit during process. The algorithm then browses the saturated tours in reversed order, meaning from customer $\pi[n]$ to customer $\pi[1]$, and updates Γ based on the recurrence relation described in Equation 4.

$$\Gamma(i,j) = \begin{cases} \max\{\Gamma(succ[i], j-1) + P[i], \Gamma(i+1, j)\} & \text{if } 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

At the end, $\Gamma(1, m)$ corresponds to the profit of the optimal solution. A simple backtrack is then performed on Γ in order to determine the corresponding tours. That is to say if $\Gamma(succ[i], j-1) + P[i]$ is used over $\Gamma(i+1, j)$ in the relation, then the saturated tour starting with customer $\pi[i]$ belongs to the optimal solution.

Figure 1 depicts the same example of the split problem described in [9] but with the new evaluation process. More precisely, in this problem we have 8 customers with $\pi = (1, 2, 3, 4, 5, 6, 7, 8)$, profits (10, 30, 10, 40, 40, 50, 10, 120), $L = 70$ and $m = 2$. According to the distances given in the figure, the saturated tours are $\langle 1, 0\rangle$, $\langle 2, 2\rangle$, $\langle 3, 1\rangle$, $\langle 4, 1\rangle$, $\langle 5, 2\rangle$, $\langle 6, 1\rangle$, $\langle 7, 1\rangle$ and $\langle 8, 0\rangle$ with profits 10, 80, 50, 80, 100, 60, 130 and 120 respectively. The interval model is shown in Figure 1.b and the detail of the first successor relations as well as solving steps are given in Figure 1.c. The new algorithm actually returns the same solution composed of the same saturated tours (starting with customers 5 and 8) as expected in [9].

2.3. *Randomized heuristics*

Particle positions in the swarm, including local best positions, are initialized to a random sequence. In order to accelerate the algorithm, a small portion of the swarm containing N_{IDCH} particles will have their local best positions generated using a good heuristic. During the search, a faster heuristic is occasionally used to generate a completely new position for a particle. The heuristics that we use are randomized variants of the Iterative Destruction/Construction Heuristic (IDCH) of [9].

The core component of IDCH is a Best Insertion Algorithm (BIA). Our BIA considers a partial solution (which can be empty) and a subset of unrouted customers to be inserted in the solution. This constructive method then evaluates the insertion cost $\frac{C_{i,z}+C_{z,j}-C_{i,j}}{(P_z)^\alpha}$ of any unrouted customer z between any couple of successive customers i and j in a tour r. The feasible insertion that minimizes the cost is then processed and the method loops back to the evaluation of the remaining unrouted customers. If more than one possible insertion minimizes the insertion cost, one of them is chosen at random. This process is iterated until no further insertions are feasible, either because no tour can accept additional customers, or because all the customers are routed. The only parameter of BIA is α and it is set to 1 in [9, 32]. In this work, a random value of α is generated each time BIA is called. This generation makes our IDCH less predictable and actually a randomized heuristic. The computational method used to generate α is detailed in Section 3.

IDCH is described as follows. Firstly, BIA is called to initialize the current solution from scratch. On following iterations a small part of the current solution is destroyed by removing a limited random number (1, 2 or 3) of random customers from tours, and a 2-opt procedure is used to reduce the travel cost of tours. A reconstruction phase is then processed using a Prioritized Best Insertion Algorithm (PBIA). The destruction and construction phases are iterated, and each time a customer remains unrouted after the construction phase its priority is increased by the value of its associated profit. In the PBIA, the subset of unrouted customers with the highest priority is considered for an insertion using a BIA call. When no more of these customers can be inserted, unrouted customers with lower priorities are considered, and so on. The idea behind this technique is to explore solutions composed of high profit customers. IDCH memorizes the best discovered solutions so far and stops after a fixed number of Destruction/Construction iterations without improvement of this solution. This number is set to n for the fast version of IDCH. This version is used to generate a new position for a particle when it is moved out of its current position. For the slower version used to initialize the PSO, this value is set to n^2. In the slow version, after n iterations without improvement a diversification process is applied. This involves destroying a large part of the solution while removing a number (bounded by n/m rather than by 3) of customers from tours then applying 2-opt to each tour to optimize the travel cost, and finally performing the reconstruction phase.

2.4. *Improvement of positions through local search*

In our PSO, whenever a new position, i.e. a new permutation, is found, it has a pm probability of being improved using a local search technique (LS). This LS contains 3 neighborhoods which were proved to be efficient for TOP [9]:

- *shift operator*: evaluate each permutation obtained by moving each customer i from its original position to any other position in the permutation.
- *swap operator*: evaluate each permutation obtained by exchanging every two customers i and j in the permutation.
- *destruction/repair operator*: evaluate the possibility of removing a random number (between 1 and $\frac{n}{m}$) of customers from an identified solution and then rebuilding the solution using BIA procedure described in the previous section.

The procedure is as follows. One neighborhood is randomly chosen to be applied to the particle position. As soon as an improvement is found, it is applied and the LS procedure is restarted from the new improved position. The LS is stopped when all neighborhoods are fully applied without there being any improvement. In addition, we enhanced the randomness of shift and swap operators. That is to say the possibilities of moving or exchanging customers in those operators are evaluated in random order.

2.5. Genetic crossover operator to update position

In combinatorial optimization, the particle position update of PSO can be interpreted as a recombination of three positions/solutions according to inertia, cognitive and social parameters. There are various ways of defining this kind of recombination operator [4]. In our approach, the recombination operator is similar to a genetic crossover whose core component is an extraction of l customers from a permutation π. To make sure that a customer can be extracted at most once from sequential calls of the core component, a set M is used to mark extracted customers from previous calls. The extracted subsequence is denoted π^l_M and the procedure is described as follows:

- Step 1 : generate a random location r in π and initialize π^l_M to empty.
- Step 2 : browse customers from $\pi[r]$ to $\pi[n]$ and add them to the end of π^l_M if they are not in M. If $|\pi^l_M|$ reaches l then go to Step 4, otherwise go to Step 3.
- Step 3 : browse customers from $\pi[r]$ down to $\pi[1]$ and add them to the beginning of π^l_M if they are not in M. If $|\pi^l_M|$ reaches l then go to Step 4.
- Step 4 : add customers from π^l_M to M.

With the core component, the position update procedure of particle x from the swarm S with respect to the three PSO parameters w, c_1 and c_2 is as follows:

- Phase 1 : apply sequentially but in a random order the core component to extract subsequences from $S[x].pos$, $S[x].lbest$ and $S[best].lbest$ with a common set M of customers to be skipped. M is initialized to the empty set and the desired numbers of customers to be extracted from $S[x].pos$, $S[x].lbest$ and $S[best].lbest$ are respectively $w \cdot n$, $(1-w) \cdot n \cdot \frac{c_1.r_1}{(c_1.r_1+c_2.r_2)}$ and $(1-w) \cdot n \cdot \frac{c_2.r_2}{(c_1.r_1+c_2.r_2)}$. Here r_1 and r_2 are real numbers whose values are randomly generated in the interval $[0,1]$ with a uniform distribution. Real numbers obtained from those computations are truncated to integral values.
- Phase 2 : link three extracted subsequences in a random order to update $S[x].pos$.

To illustrate the update procedure, we consider an arbitrary instance of TOP with ten customers and an arbitrary particle x with $S[x].pos$ =(4, 5, 2, 6, 10, 1, 7, 8, 9, 3), $S[x].lbest$ = (4, 2, 3, 8, 5, 6, 9, 10, 7, 1) and $S[best].lbest$ = (1, 2, 4, 9, 8, 10, 7, 6, 3, 5). PSO parameters are $w = 0.3$, $c_1 = 0.5$ and $c_2 = 0.3$. Random variables r_1 and r_2 generated are respectively 0.5 and 0.5. Then the desired numbers of customers to be extracted for $S[x].pos$, $S[x].lbest$ and $S[best].lbest$ are respectively 3 ($= \lfloor 0.3 * 10 \rfloor$), 4 ($= \lfloor (1-0.3) * 10 * 0.5 * 0.5/(0.5 * 0.5 + 0.3 * 0.5) \rfloor$) and 3 ($= 10 - 3 - 4$). Random extraction order in Phase 1 is ($S[x].lbest$, ...) —

Random extraction order in Phase 1 is ($S[x].pos$, $S[x].lbest$, $S[best].lbest$) and random linking order in Phase 2 is ($S[x].lbest$, $S[x].pos$, $S[best].lbest$). Figure 2 gives an example of the update procedure that indicates the new position for the particle x of (8, 5, 6, 9, 10, 1, 7, 2, 4, 3).

Our particle position update procedure therefore works with the standard PSO parameters w, c_1 and c_2, the only restriction being that w has to be in the interval $[0, 1[$. Our PSO approach can be classified as PSO with position only, given that no velocity vector is used [26]. It is noteworthy to mention that the core component was created to adapt to a linear permutation order, but it can easily be adapted to a circular order by changing Step 3.

2.6. Swarm local best update

In some situations, PSO can be trapped in a local optimum, especially when all the local best positions of particles in the swarm are identical. In our approach, the fact that a particle can be randomly moved out of its current position reduces this premature convergence. However, the effect of this reduction is only partial because the probability to move a particle out of its current position is set to a small value. This setting is due to two main reasons: firstly, a frequent use of the IDCH heuristic to generate new positions is time-consuming and secondly, a frequent use of perturbing operations is undesired in a PSO algorithm [42].

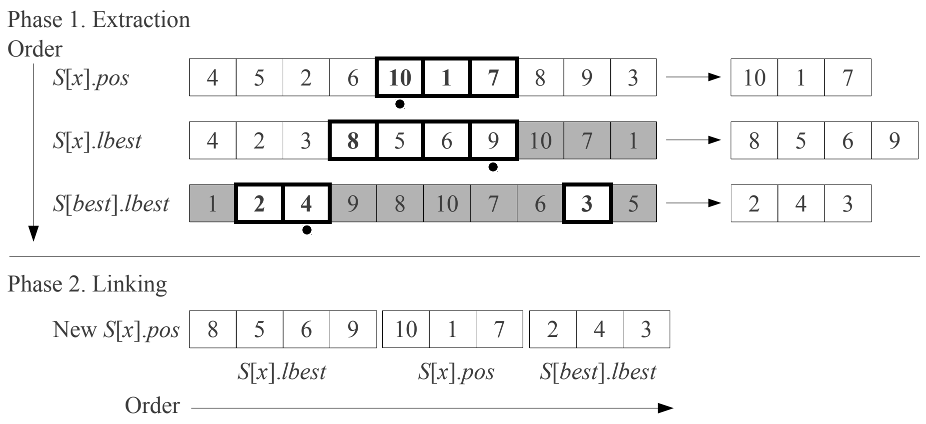


Figure 2: An example of position update for an arbitrary instance of ten customers. Black dots represent random generated locations r and shaded boxes represent marked customers from M during Phase 1.

So then to strengthen the diversification process, whenever a new position is found by a particle x in the swarm S, instead of updating $S[x].lbest$, the algorithm will search for an appropriate particle y in the swarm using a similarity measure and update $S[y].lbest$. The similarity measure is based on two criteria: the total collected profit and the travel cost/time of the identified solution. Two positions are said to be *similar* or *identical* if the evaluation procedure on these positions returns the same profit and a difference in travel cost/time that is lower than a value δ. Our update rules are based on Sha and Hsu [31] but simplified as follows. For convenience, the particle having the worst local best position of the swarm is denoted as $S[worst]$.

- Rule 1 : the update procedure is applied if and only if the performance of new position $S[x].pos$ is better than the worst local best $S[worst].lbest$.
- Rule 2 : if there exists a particle y in S such that $S[y].lbest$ is similar to $S[x].pos$, then replace $S[y].lbest$ with $S[x].pos$.
- Rule 3 : if no such particle y according to Rule 2 exists, replace $S[worst].lbest$ with $S[x].pos$. Each successful application of this rule indicates that a new local best has been *discovered* by the swarm.

The implementation of these rules was made efficient through the use of a binary search tree to sort particles by the performance of their local best positions using the two criteria. In the next section, the performance of our PSO on the standard benchmark for TOP is discussed.

3. Numerical results on the standard benchmark

PSOiA is coded in C++ using the Standard Template Library (STL) for data structures. The program is compiled with GNU GCC in a Linux environment, and all experiments were conducted on an AMD Opteron 2.60 GHz. In order to compare the performance of our approach with those of the existing algorithms in the literature, we use 387 instances from the standard benchmark for TOP [13]. These instances comprise 7 sets. Inside each set the original number of customers and customer positions are constant, however the maximum tour duration L varies. Therefore the number of accessible customers are different for each instance. The number of vehicles m also varies between 2 and 4.

3.1. Protocol and performance metrics

Our approach was tested using the same protocol as in [21, 25, 32]. For each instance of the benchmark, the algorithms were executed 10 times. The average and maximal scores as well as the average and maximal computational times were recorded. In order to evaluate separately the performance of different configurations or methods, the best known result in the literature for each instance, denoted by Z_{best}, is used as the reference score of the instance. These best results for all instances of the benchmark are collected from [2, 9, 16, 21, 32, 33] and also from our PSO algorithms, but not from Chao et al. [14] because the authors used a different rounding precision and some of their results exceeded the upper bounds given in [10].

For an algorithm tested on an instance, obtained solutions of 10 runs are recorded and we use Z_{max} and Z_{avg} to denote respectively the maximal and average scores of these runs. Then the relative percentage error (RPE) and the average relative percentage error (ARPE) are used to evaluate the performance of the algorithm. RPE is defined as the relative error between Z_{best} and Z_{max}. It was used in [25, 32] to show the performance of the algorithm over 10 runs.

$$RPE = \frac{Z_{best} - Z_{max}}{Z_{best}} \cdot 100 \tag{5}$$

ARPE is defined as the relative error between Z_{best} and Z_{avg}. It was used in [25] to show the robustness of the algorithm over 10 runs. In other words, a small value of ARPE indicates a higher chance of getting a good score (or a small RPE) for a limited number of runs of the algorithm on the instance. The instances, for which there is no accessible customer (or $Z_{best} = 0$) are discarded from the comparison. The number of instances is then reduced to 353.

$$ARPE = \frac{Z_{best} - Z_{avg}}{Z_{best}} \cdot 100 \tag{6}$$

For a set of instances, the respective average values of RPE and ARPE of the instances are computed to show the performance and robustness of the algorithm. For a benchmark composed of different sets, the average value of the latter ones on all the sets is computed to show the overall performance and robustness of the algorithm on the benchmark. As a complement measure for a benchmark, NBest is used to denote the number of instances in which Z_{best} are reached.

3.2. Parameter setting

Values of some parameters are directly taken from the previous studies of [9, 16]. Therefore, we did not do further experiments on those parameters:

- N, the population size, is set to 40.
- N_{IDCH}, the number of local best positions initialized with the slow version of IDCH, is set to 5.
- pm, the local search rate, is set to $1 - \frac{iter}{itermax}$.
- δ, the similarity measurement of particles, is set to 0.01.
- c_1, c_2, the cognitive and social factors of the particles, are set to 0.5 ($c_1 = c_2 = 0.5$).
- w, the inertia parameter, decreases gradually as the algorithm proceeds. It is initialized to 0.9 and multiplied by 0.9 after each iteration of the PSO.
- α, the control parameter of intuitive criteria of the BIA heuristic, is generated as follows. Two random numbers r_1 and r_2 are first generated in $[0, 1]$ with a uniform distribution, then $\alpha = 1 + 2 \cdot \frac{r_1}{r_1+r_2}$ is computed.

The most important parameter which could be up for discussion is the stopping condition k. We tested PSOiA on the 353 instances of the benchmark using varied values of k from 10 to 100 with steps of 10. In order to maximally exploit in these tests the crossover operator and the evaluation process, we set the

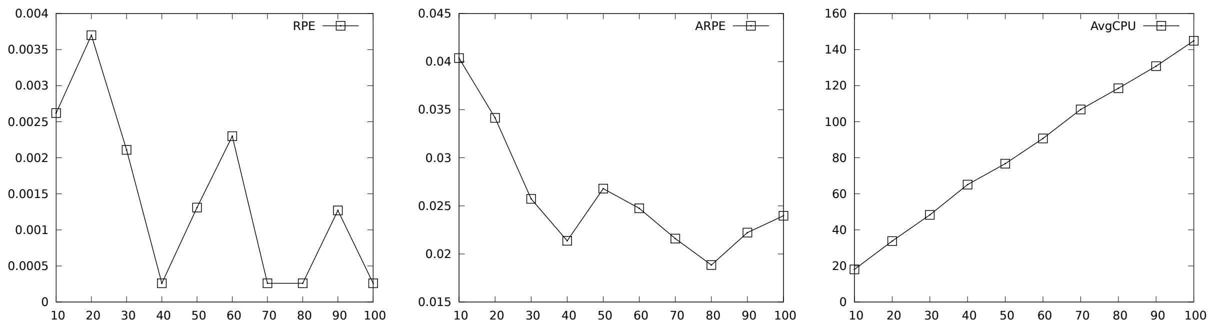


Figure 3: Performance of PSOiA in terms of the stopping condition k.

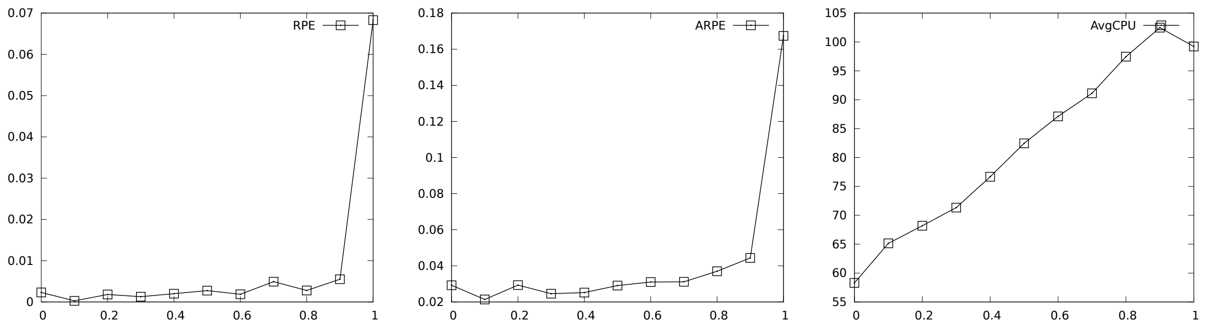


Figure 4: Performance of PSOiA in terms of the probability ph of a particle to be moved out of its current position.

probability ph of a particle to be moved out of its current position equal to 0.1. We will return to the ph parameter later (once k is fixed) to check whether it is over-tuned.

Figures 3 illustrate the evolution of RPE, ARPE and the average computational time in terms of k. One may notice that from $k = 40$, the algorithm starts to provide the best RPE and interesting values of ARPE. On the other hand, the computational time linearly increases in terms of k, hence the value $k = 40$ were selected to present our final results of PSOiA.

Next, we set k to 40 and varied the value of ph from 0 to 1 with a step equal to 0.1. Figures 4 show the evolution of RPE, ARPE and the average computational time in terms of ph. In these tests, the computational time linearly increases in terms of ph (with a small exception for $ph = 1.0$) and value 0.1 is the right choice for the parameter.

3.3. Comparison with the literature

The results of PSOiA ($k = 40$) on instances of Chao's benchmark are then compared with the state-of-the-art algorithms in the literature:

- SVNS proposed by Archetti et al. [2], tested on an Intel Pentium 4 2.80 GHz,
- MA proposed by Bouly et al. [9], tested on an Intel Core 2 Duo 2.67 GHz,
- SPR proposed by Souffriau et al. [32], tested on an Intel Xeon 2.50 GHz,
- PSOMA (with $w = 0.07$, the best configuration) described in [16] as the preliminary study of this work, tested on an AMD Opteron 2.60 GHz.

On the comparison between computers in use, machine performances of PSOiA, PSOMA, MA/MA10 [9] and SPR [32] are almost the same: recent dual-core processors with clock frequency varying from 2.50 GHz

Method	Year	RPE average for each data set					NBest
		4	5	6	7	avg	
SVNS	2007	0.0680	0.0267	0	0.0627	0.0394	134
ACO	2008	0.3123	0.0355	0	0.0064	0.0885	128
MA	2010	0.0548	0.0612	0	0.0571	0.0433	129
SPR	2010	0.1157	0.0465	0	0.0454	0.0519	126
DPSO	2011	2.0911	0.7828	0.3375	1.7618	1.2433	39
MA10	2011	0.0304	0.0612	0	0.0127	0.0261	146
PSOMA	2011	0.0262	0.0151	0	0.0211	0.0156	146
PSOiA	2012	0.0019	0	0	0	0.0005	156

Table 1: Performance comparison based on RPE average for each data set of the relevant instances.

Method	Year	ARPE average for each data set				
		4	5	6	7	avg
SVNS	2007	n/a	n/a	n/a	n/a	n/a
ACO	2008	1.8663	0.8228	1.1754	0.5118	1.0941
MA	2010	n/a	n/a	n/a	n/a	n/a
SPR	2010	n/a	n/a	n/a	n/a	n/a
DPSO	2011	5.1956	3.7100	2.0073	4.1986	3.7779
MA10	2011	0.2068	0.0953	0.0169	0.1056	0.1061
PSOMA	2011	0.2851	0.0904	0	0.1790	0.1386
PSOiA	2012	0.1105	0.0336	0	0.0305	0.0436

Table 2: Robustness comparison based on ARPE average for each data set of the relevant instances.

to 2.67 GHz. SVNS [2] used a computer with higher clock frequency (2.8 GHz) but that was a Pentium 4. It is supposed to have a lower performance than the others.

In [32], the authors of SPR algorithm talk about the 157 relevant instances of sets 4, 5, 6 and 7 and show only their results on these instances. Therefore, we will provide the comparison focused on these 157 instances. We also noted that results of SVNS were taken from the website of the first author of [2]. These results were updated in 2008 and the rounding convention problem reported in [9, 21] was corrected. It also appears that these results are better than the ones published in the journal article [2]. Additionally, a different testing protocol which considered only 3 runs for each instance of the benchmark had been used for SVNS and MA. So in [16], the source code of MA [9] was received from the authors and turned to match the new testing protocol: 10 executions per instance. Results of this new test for MA is denoted by MA10.

Our results are also compared with the other swarm intelligence algorithms available in the literature:

- Sequential version of the Ant Colony Optimization (ACO) proposed by Ke et al. [21], tested on an Intel CPU 3.0 GHz,
- Discrete Particle Swarm Optimization (DPSO) proposed by Muthuswamy and Lam [25], tested on an Intel Core Duo 1.83 GHz.

Table 1 reports RPE averages for each data set of all methods. From this table, we observe that PSOMA (with very basic PSO components) already outperforms the other methods in the literature. This motivates our choice of testing the new optimal split procedure on PSO scheme instead of MA one. Regarding PSOiA, the results are almost perfect with zero RPE for sets 5, 6, 7 and only one instance was missed for set 4 with a very small value of RPE. Table 2 reports ARPE averages for each data set of the standard benchmark.

Method	Average CPU time in seconds for each data set						
	1	2	3	4	5	6	7
SVNS	7.78	0.03	10.19	457.89	158.93	147.88	309.87
ACO	5.77	3.16	6.50	37.09	17.36	16.11	30.35
MA	1.31	0.13	1.56	125.26	23.96	15.53	90.30
SPR	n/a	n/a	n/a	36.74	11.99	8.96	27.28
DPSO	n/a	n/a	n/a	n/a	n/a	n/a	n/a
MA10	1.95	0.24	2.06	182.36	35.33	39.07	112.75
PSOMA	0.18	0.01	0.49	83.89	14.72	7.59	49.09
PSOiA	2.15	0.41	3.18	218.58	49.5	47.08	97.47

Table 3: Average CPU time for each data set of the standard benchmark.

Method	Maximal CPU time in seconds for each data set						
	1	2	3	4	5	6	7
SVNS	22	1	19	1118	394	310	911
ACO	n/a	n/a	n/a	n/a	n/a	n/a	n/a
MA	4.11	0.531	3.963	357.053	80.19	64.292	268.005
SPR	n/a	n/a	n/a	n/a	n/a	n/a	n/a
DPSO	n/a	n/a	n/a	n/a	n/a	n/a	n/a
MA10	8.59	1.16	6.34	635.75	113.58	96.89	443.59
PSOMA	4.35	0.03	4.88	466.65	78.12	48.77	350.86
PSOiA	10.61	2.20	10.93	1274.52	170.09	115.93	420.50

Table 4: Maximal CPU time for each data set of the standard benchmark.

Method	Year	Number of instances (%)				
		4	5	6	7	avg
SVNS	2007	n/a	n/a	n/a	n/a	n/a
ACO	2008	7	18	0	21	11
MA	2010	n/a	n/a	n/a	n/a	n/a
SPR	2010	n/a	n/a	n/a	n/a	n/a
DPSO	2011	0	2	0	0	1
MA10	2011	33	84	93	47	61
PSOMA	2011	26	76	100	33	55
PSOiA	2012	52	87	100	74	72

Table 5: Stability comparison based on the number of instances having zero APRE.

From that table, we observe that PSOMA is less robust than MA10 on data sets 4 and 7. However, it is more robust than MA10 on data sets 5 and 6. Finally, PSOiA is the most robust method. The ARPE average on all data sets of PSOiA is 0.0436% which almost equivalent to the RPE averages on all data sets of the state-of-the-art algorithms (SVNS, SPR and MA) reported in the literature (ranging from 0.0394% to 0.0519%) as shown in Table 1.

Tables 3 and 4 report the average and maximal CPU times respectively for each data set of all methods. From this table, we notice that ACO and SPR are fast methods. However, their performances are not as good as SVNS, MA/MA10, PSOMA and PSOiA as seen in Tables 1 and 2. SVNS is slower than the others. PSOMA is quite faster than MA10 but as mentioned above, MA10 is more robust than PSOMA. Computational efforts required for PSOiA and MA10 are almost the same. Based on a remark of [35] and our own verification, it is worthy to mention that the maximal CPU times of ACO method reported in [21] are in fact the maximal average CPU times, i.e. for each instance the average CPU time is computed, then the maximal value of these CPU times is reported for a whole set. Therefore, the maximal CPU times of ACO method are marked as n/a (not available) in Table 4.

Table 5 reports the number of instances (in percent) for which the value of ARPE is zero, which means that the results of all runs are identical or that the algorithm is stable. From this table, one may notice that PSOiA is stable in most cases (72%). Additionally, the performance analysis of SVNS, MA, SPR, PSOMA and PSOiA indicates that the results from data sets 4 and 7 are generally less stable than those from data sets 5 and 6. This can be explained by the differences between the features of those instances. Data sets 4 and 7 contain up to 100 customers for which both profits and positions are randomly distributed. On the other hand, data sets 5 and 6 have at most 64 customers arranged in a grid such that large profits are assigned to customers located far away from the depots.

Finally, detailed results of MA10, PSOMA and PSOiA for the 157 instances are reported in Tables 7, 8, 9 and 10. For each instance, columns CPU_{avg} report the average computational time in seconds of the ten runs. Complete results of the 353 tested instances are available at http://www.hds.utc.fr/~moukrim. According to these results, $p4.4.n$ is the only instance of the whole benchmark from which PSOiA was not able to find the best known solution. One unit of profit was missed for this instance. Furthermore, a strict improvement was detected for instance $p4.2.q$ with a new score of 1268 instead of 1267.

4. A set of larger instances for the team orienteering problem

From the previous section, we observe that PSOiA achieves a value of RPE of 0.0005%. Therefore, it would be very difficult to develop better heuristics for the current standard benchmark instances. In order to promote algorithmic developments for TOP, we introduce a new set of benchmark instances with a larger number of customers. Our new instances are based on the OP instances of Fischetti et al. [18] with the transformation of Chao et al. [13]. This transformation consists of designing the travel length limit of vehicles for TOP as $L^{TOP} = \frac{L^{OP}}{m}$. In this formulation, m is the number of vehicles of the new TOP instance and L^{OP} is the travel length limit of the vehicle of the former OP instance.

We used instances from the two classes described in [18] to generate TOP instances. According to the authors, the first class was derived from instances of the Capacitated Vehicle Routing Problem (CVRP) [15, 29] in which customer demands were transformed into profits and varied values of L^{OP} were considered. The second class was derived from instances of the Traveling Salesman Problem (TSP) [29] in which customer profits were generated in different ways: equal to 1 for each customer (gen1); using pseudo-random function so that the output values are in $[1, 100]$ (gen2); using distance-profit function such that large profits are assigned to nodes far away from the depots (gen3).

In total, 333 new instances were used in our test. It can be seen that PSOiA is very stable for a large part of those instances, especially the ones from the CVRP benchmark. So Table 11 reports the results of PSOiA for which the value of ARPE is non-zero. A complete specification, including the number of accessible customers n, the number of vehicles m, the travel length limit L and the way to generate the profits for the customers gen, is also given for each instance. The values in the last row corresponding to Z_{avg} and CPU_{avg} columns respectively indicate the ARPE and the average computational time on the set of instances. All tested instances and the other results are available on the previously mentioned website.

Generation	Number of instances with ARPE zero (%)	CPU_{avg}
gen1	81%	4123.84
gen2	73%	4764.23
gen3	75%	5357.32
	76%	4748.47

Table 6: Influence of profit generations on the stability of PSOiA

In addition, we also analyzed the computational behavior of PSOiA on the new instances according to the various generations of the profits (namely gen1, gen2 and gen3). In this analysis, for each TSP instance, three variants of TOP instances are available. This implies the same sample size of 93 instances per generation and provides a fair comparison. Table 6 reports the number of instances (in percent) for which the ARPE is zero and the average computational time CPU_{avg} for each generation. From this table, one may notice that PSOiA is more stable and requires less computational effort on generation $gen1$ (equal profits) than on generations $gen2$ (random profits) and $gen3$ (large profits distributed to customers located far away). Finally, it should be noted that the sample size to analyze the stability of PSOiA according to the positions of the customers is not statistically large enough to reveal the detail.

5. Conclusion

This paper presented an effective Particle Swarm Optimization approach for the Team Orienteering Problem. The approach uses giant tours to indirectly encode particle positions. A new fast evaluation process based on an interval graph model was proposed. This process enabled more iterations for the PSO without increasing the global computational time. Numerical results on the standard benchmark for TOP demonstrate the competitiveness of the algorithm. Our approach outperforms the prior methods both in terms of computational time and solution quality. Hence it improved considerably solving methods for TOP, a new strict improvement on one instance was detected and the newly attained relative error for all instances being 0.0005%. This success is due to the new accelerated split procedure, the good design of the recombination operator to update particle positions, the introduction of extra positions to the swarm, as well as the appropriate management of dynamic parameters. In summary, the results presented in this paper are encouraging for the application of Particle Swarm Optimization to solve combinatorial problems, as already indicated in [4] and for the application/acceleration of optimal split procedures in dealing with vehicle routing problems, as already indicated in [17].

Acknowledgements

This work was partially supported by the GDR in Operations Research of the French National Center for Scientific Research (CNRS), ARFRO's project "multi-objective and multi-modal path planning on large graphs". Finally, the authors would like to thank two anonymous reviewers for their helpful comments and suggestions that improved the quality of the paper.

Table 7: Results for set 4 of the benchmark.

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p4.2.a	206	206	206	13.71	206	206	0.21	206	206	5.88
p4.2.b	341	341	341	51.8	341	341	0.65	341	341	39.21
p4.2.c	452	452	452	83.16	452	452	2.05	452	452	67.12

continued on next page

Table 7 – continued

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p4.2.d	531	531	530.7	143.67	531	530.8	29.77	531	531	124.29
p4.2.e	618	618	616.8	205.89	618	618	25.23	618	618	197.94
p4.2.f	687	687	679.6	204.92	687	681.4	109.12	687	687	322.64
p4.2.g	757	757	756	190.49	757	755.5	97.95	757	757	206.54
p4.2.h	835	835	828.1	245.31	835	826.1	126.94	835	833.6	257.24
p4.2.i	918	918	918	366.45	918	913.6	150.45	918	918	368.2
p4.2.j	965	965	962.6	300.78	965	963.1	167.88	965	965	258.36
p4.2.k	1022	1022	1020.7	370.53	1022	1020.2	180.83	1022	1021	350.07
p4.2.l	1074	1071	1070.8	281.38	1071	1066.9	160.98	1074	1072.3	357.41
p4.2.m	1132	1132	1129.4	303.22	1132	1129.9	201.95	1132	1130.6	321.08
p4.2.n	1174	1174	1172.7	374.27	1174	1170.3	158.34	1174	1172	427.5
p4.2.o	1218	1218	1216	306.18	1218	1211.7	154.13	1218	1210.9	415.43
p4.2.p	1242	1242	1241.2	311.23	1241	1238.4	198.04	1242	1239.5	347.47
p4.2.q	1268	1267	1264.3	313.03	1267	1264.6	192.02	1268	1266.2	588.22
p4.2.r	1292	1292	1288.7	352.5	1292	1287.8	178.19	1292	1289.9	470.01
p4.2.s	1304	1304	1301.8	297.88	1304	1302.1	190.02	1304	1303.8	486.19
p4.2.t	1306	1306	1306	292.66	1306	1306	167.27	1306	1306	408.65
p4.3.c	193	193	193	6.07	193	193	0.05	193	193	1.89
p4.3.d	335	335	335	29.92	335	335	1.08	335	335	16.6
p4.3.e	468	468	468	39.23	468	468	2.6	468	468	36.41
p4.3.f	579	579	579	107.55	579	579	6.3	579	579	72.88
p4.3.g	653	653	653	117.44	653	651.4	32.49	653	653	70.44
p4.3.h	729	728	724.7	138.95	729	724.7	60.24	729	729	194.18
p4.3.i	809	809	809	197.7	809	808.6	41.17	809	809	247.26
p4.3.j	861	861	859.5	175.98	861	857.6	85.71	861	860.9	229.11
p4.3.k	919	919	918.2	218.73	919	916.7	94.55	919	919	275.31
p4.3.l	979	979	975.2	206.99	979	977.4	89.28	979	976.9	258.33
p4.3.m	1063	1063	1057.9	231.16	1063	1058.4	103.42	1063	1062	281.42
p4.3.n	1121	1121	1115.9	197.23	1121	1115.6	104.57	1121	1118.4	309.03
p4.3.o	1172	1172	1169	306.3	1172	1169.7	145.44	1172	1172	371.72
p4.3.p	1222	1222	1219.4	301.81	1222	1222	123.77	1222	1222	284.61
p4.3.q	1253	1253	1250	220.44	1253	1250.2	112.58	1253	1252.2	448.69
p4.3.r	1273	1273	1270.2	218.38	1273	1269.5	115.4	1273	1269.4	288.72
p4.3.s	1295	1295	1293.8	255.25	1295	1291.5	115.6	1295	1289.5	278
p4.3.t	1305	1305	1303.7	210.95	1304	1301.1	124.38	1305	1304.3	305.85
p4.4.e	183	183	183	0.45	183	183	0.02	183	183	0.65
p4.4.f	324	324	324	16.5	324	324	0.2	324	324	8.61
p4.4.g	461	461	461	35.13	461	461	0.74	461	461	24.19
p4.4.h	571	571	571	52.23	571	567.1	5.56	571	571	36.74
p4.4.i	657	657	657	72.75	657	657	1.71	657	657	65.48
p4.4.j	732	732	732	95.06	732	731.2	9.2	732	732	81.35
p4.4.k	821	821	820.2	110.1	821	820.8	20.03	821	821	119.45
p4.4.l	880	880	879.1	113.9	880	879.1	54.34	880	879.5	101.6
p4.4.m	919	916	912.7	129.08	919	915.6	67.59	919	916.6	223.19
p4.4.n	977	969	965.5	197.89	969	964.2	72.07	976	967	257.14
p4.4.o	1061	1061	1057.7	185.94	1061	1051.6	91.63	1061	1060	208.36
p4.4.p	1124	1124	1119.8	226.38	1124	1115.9	102.16	1124	1122.7	193.14

continued on next page

Table 7 – continued

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p4.4.q	1161	1161	1161	195.51	1161	1159.4	94.86	1161	1161	252.98
p4.4.r	1216	1216	1210.2	226.09	1216	1201	94.76	1216	1206.2	260.06
p4.4.s	1260	1260	1256.9	191.25	1259	1257.1	131.53	1260	1257.5	256.15
p4.4.t	1285	1285	1283	175.01	1285	1284.5	100.62	1285	1282.3	161.33

Table 8: Results for set 5 of the benchmark.

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p5.2.h	410	410	410	50.52	410	410	1.34	410	410	51.98
p5.2.j	580	580	580	41.63	580	580	1.45	580	580	54.37
p5.2.k	670	670	670	41.12	670	670	1.63	670	670	60.78
p5.2.l	800	800	800	65.16	800	800	37.6	800	800	88.15
p5.2.m	860	860	860	62.88	860	860	33.52	860	860	90.5
p5.2.n	925	925	925	53.75	925	925	25.21	925	925	72.46
p5.2.o	1020	1020	1020	47.56	1020	1020	29.7	1020	1020	65.93
p5.2.p	1150	1150	1150	96.04	1150	1150	59.14	1150	1150	109.63
p5.2.q	1195	1195	1195	53.26	1195	1195	36.71	1195	1195	116.62
p5.2.r	1260	1260	1260	68.66	1260	1260	37.6	1260	1260	92.78
p5.2.s	1340	1330	1325	63.44	1340	1329.5	37.04	1340	1340	85.78
p5.2.t	1400	1400	1397	52.27	1400	1400	37.39	1400	1400	111.2
p5.2.u	1460	1460	1460	68.79	1460	1460	44.59	1460	1460	110.39
p5.2.v	1505	1505	1503.5	65.45	1505	1504.5	43.79	1505	1505	112.4
p5.2.w	1565	1560	1560	50.22	1560	1560	47.45	1565	1562.5	124.13
p5.2.x	1610	1610	1610	57.27	1610	1610	50.02	1610	1610	124.68
p5.2.y	1645	1645	1645	66.25	1645	1645	37.98	1645	1645	112.34
p5.2.z	1680	1680	1680	64.77	1680	1679	41.75	1680	1680	122.55
p5.3.k	495	495	495	30.34	495	495	1.36	495	495	33.27
p5.3.l	595	595	595	39.81	595	595	1.25	595	595	53.91
p5.3.n	755	755	755	41.9	755	755	1.87	755	755	48.68
p5.3.o	870	870	870	34.7	870	870	2.13	870	870	48.93
p5.3.q	1070	1070	1070	49.38	1070	1070	23.28	1070	1070	51.54
p5.3.r	1125	1125	1125	43.97	1125	1125	22.68	1125	1125	46.7
p5.3.s	1190	1190	1189	41.16	1190	1189	26.6	1190	1190	59.48
p5.3.t	1260	1260	1260	54.36	1260	1260	32.65	1260	1260	69.12
p5.3.u	1345	1345	1345	51.74	1345	1345	26.21	1345	1345	57.97
p5.3.v	1425	1425	1425	48.05	1425	1425	29	1425	1425	56.21
p5.3.w	1485	1485	1481.5	44.83	1485	1477	36.32	1485	1484.5	76.2
p5.3.x	1555	1555	1547.5	50.42	1555	1546.5	36.79	1555	1552	76.78
p5.3.y	1595	1590	1590	54.95	1595	1590.5	30.46	1595	1591	70.54
p5.3.z	1635	1635	1635	61.91	1635	1635	32.64	1635	1635	84.24
p5.4.m	555	555	555	22.29	555	555	1.46	555	555	20.76
p5.4.o	690	690	690	33.08	690	690	1.69	690	690	42.91
p5.4.p	765	760	760	46.44	765	760.5	27.36	765	760.5	48.35
p5.4.q	860	860	860	50.91	860	860	1.53	860	860	61.58

continued on next page

Table 8 – continued

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p5.4.r	960	960	960	64.85	960	960	1.39	960	960	76.4
p5.4.s	1030	1030	1029.5	39.19	1030	1029.5	18	1030	1030	30.82
p5.4.t	1160	1160	1160	48.1	1160	1160	2.09	1160	1160	47.63
p5.4.u	1300	1300	1300	66.82	1300	1300	2.1	1300	1300	67.26
p5.4.v	1320	1320	1320	37.87	1320	1320	1.77	1320	1320	97.97
p5.4.w	1390	1380	1380	29.24	1385	1381	19.37	1390	1386	40.5
p5.4.x	1450	1450	1450	47.34	1450	1448	25.75	1450	1450	63.41
p5.4.y	1520	1520	1520	46.84	1520	1520	32.06	1520	1520	102.12
p5.4.z	1620	1620	1620	50.43	1620	1620	37.84	1620	1620	86.55

Table 9: Results for set 6 of the benchmark.

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p6.2.d	192	192	192	11.37	192	192	0.14	192	192	5.76
p6.2.j	948	948	945.6	54.68	948	948	27.8	948	948	54.1
p6.2.l	1116	1116	1116	50.3	1116	1116	34.75	1116	1116	66.31
p6.2.m	1188	1188	1188	38.06	1188	1188	28.62	1188	1188	58.34
p6.2.n	1260	1260	1260	34.66	1260	1260	21.82	1260	1260	62.26
p6.3.g	282	282	282	8.3	282	282	0.17	282	282	5.85
p6.3.h	444	444	444	19.78	444	444	0.71	444	444	12.34
p6.3.i	642	642	642	33.84	642	642	1.3	642	642	31.16
p6.3.k	894	894	894	34.05	894	894	2.05	894	894	61.6
p6.3.l	1002	1002	1002	39.3	1002	1002	4.42	1002	1002	50.91
p6.3.m	1080	1080	1080	30.86	1080	1080	21.27	1080	1080	56.64
p6.3.n	1170	1170	1170	30.85	1170	1170	1.61	1170	1170	52.52
p6.4.j	366	366	366	7.28	366	366	0.18	366	366	5.23
p6.4.k	528	528	528	13.51	528	528	0.6	528	528	8.56
p6.4.l	696	696	696	18.96	696	696	1.12	696	696	27.37

Table 10: Results for set 7 of the benchmark.

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p7.2.d	190	190	190	12.81	190	190	0.11	190	190	4.36
p7.2.e	290	290	290	30.15	290	290	0.57	290	290	18.56
p7.2.f	387	387	386.4	59.79	387	386.4	7.86	387	387	40.48
p7.2.g	459	459	459	106.42	459	459	49.73	459	459	126.9
p7.2.h	521	521	520.6	113.88	521	521	32.42	521	521	188.66
p7.2.i	580	580	579.4	151.09	580	579.1	82.32	580	579.9	181.96
p7.2.j	646	646	646	159.48	646	645.3	54.9	646	646	134.9
p7.2.k	705	705	703.7	182.99	705	704.2	70.42	705	704.7	170.92
p7.2.l	767	767	767	205.35	767	767	73.7	767	767	210.89

continued on next page

Table 10 – continued

Instance	Z_{best}	MA10			PSOMA			PSOiA		
		Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}	Z_{max}	Z_{avg}	CPU_{avg}
p7.2.m	827	827	827	203.14	827	827	104.68	827	827	177.68
p7.2.n	888	888	887.9	261.33	888	887.6	142.88	888	888	186.65
p7.2.o	945	945	945	221.07	945	945	109.36	945	945	208.93
p7.2.p	1002	1002	1001.6	274.36	1002	999.2	103.42	1002	1001.8	241.83
p7.2.q	1044	1044	1043.8	247.36	1044	1039.2	130.83	1044	1043.7	181.26
p7.2.r	1094	1094	1094	232.16	1094	1091.3	126.52	1094	1094	182.32
p7.2.s	1136	1136	1136	258.83	1136	1134.5	127.36	1136	1136	228.1
p7.2.t	1179	1179	1176.3	280.06	1179	1174.1	157.24	1179	1179	277.18
p7.3.h	425	425	424.8	43.6	425	424.2	3.62	425	425	27.88
p7.3.i	487	487	487	70.87	487	487	5.96	487	487	45.65
p7.3.j	564	564	563.6	98.35	564	562.7	30.49	564	564	54.98
p7.3.k	633	633	632.7	130.31	633	632	56.7	633	633	88.79
p7.3.l	684	684	682.1	112.98	683	682	50.14	684	684	100.72
p7.3.m	762	762	762	158.38	762	760.9	64.26	762	762	127.04
p7.3.n	820	820	820	164.08	820	817.6	114.61	820	820	175.64
p7.3.o	874	874	871	173.12	874	872.5	84.87	874	874	196.91
p7.3.p	929	929	926	165.55	927	924.8	73.3	929	928	162.61
p7.3.q	987	987	987	205.16	987	980.6	107.93	987	987	168.7
p7.3.r	1026	1026	1022.4	218.64	1026	1021.7	101.17	1026	1022.6	203.02
p7.3.s	1081	1081	1079.7	245.28	1081	1079.5	123.09	1081	1081	242
p7.3.t	1120	1120	1118.7	266.55	1120	1118.2	138.14	1120	1118.4	151.73
p7.4.g	217	217	217	10.06	217	217	0.08	217	217	1.72
p7.4.h	285	285	285	13.28	285	285	0.16	285	285	4.44
p7.4.i	366	366	366	29.69	366	366	0.46	366	366	12.68
p7.4.k	520	520	518.4	61.08	520	518.2	14.42	520	518.2	39.94
p7.4.l	590	590	587.9	80.05	590	588.4	19.85	590	590	53.8
p7.4.m	646	646	646	109.07	646	646	33.59	646	646	100.05
p7.4.n	730	726	726	113.62	726	725.9	37.48	730	728.8	110.63
p7.4.o	781	781	779.4	133.82	781	779.3	54.97	781	780.4	93.27
p7.4.p	846	846	843.7	167.49	846	841.4	62.24	846	846	124.32
p7.4.q	909	909	907	165.07	909	907	77.59	909	908.7	134.35
p7.4.r	970	970	970	147.92	970	970	65.89	970	970	143.04
p7.4.s	1022	1022	1020.8	170.42	1022	1020.7	62.23	1022	1022	150.72
p7.4.t	1077	1077	1077	180.48	1077	1077	88.03	1077	1077	128.13

Table 11: Results of the new instances.

Instance	n	m	L	gen	PSOiA		
					Z_{max}	Z_{avg}	CPU_{avg}
cmt101c_m3	100	3	126.33	$gen2$	1300	1299	111.109
cmt151b_m3	150	3	116.67	$gen2$	1385	1373.8	754.007
cmt151c_m2	150	2	262.5	$gen2$	1963	1962	1799.641
cmt151c_m3	150	3	175	$gen2$	1916	1909.1	1376.236
cmt151c_m4	150	4	131.25	$gen2$	1880	1875.6	881.114
cmt200b_m2	199	2	191	$gen2$	2096	2088.2	4180.987

continued on next page

Table 11 – continued

Instance	n	m	L	gen	PSOiA		
					Z_{max}	Z_{avg}	CPU_{avg}
cmt200b_m3	199	3	127.33	$gen2$	2019	2005	2711.659
cmt200b_m4	198	4	95.5	$gen2$	1894	1889.7	1515.185
cmt200c_m2	199	2	286.5	$gen2$	2818	2810.1	7320.261
cmt200c_m3	199	3	191	$gen2$	2766	2751.2	4217.286
cmt200c_m4	199	4	143.25	$gen2$	2712	2700.6	3004.103
eil101b_m3	100	3	105	$gen2$	916	913.8	134.388
eil101c_m2	100	2	236	$gen2$	1305	1304.8	452.79
eil101c_m3	100	3	157.33	$gen2$	1251	1244.1	227.613
gil262a_m2	241	2	297.5	$gen2$	4078	4056.4	5907.285
gil262a_m4	112	4	148.75	$gen2$	3175	3174.2	271.83
gil262b_m2	249	2	594.5	$gen2$	8081	8061.1	7473.182
gil262b_m3	249	3	396.33	$gen2$	7585	7574.9	7276.798
gil262b_m4	241	4	297.25	$gen2$	6781	6742	4878.641
gil262c_m2	249	2	892	$gen2$	11030	11020	27500.87
gil262c_m3	249	3	594.67	$gen2$	10757	10714.6	14553.762
gil262c_m4	249	4	446	$gen2$	10281	10259.4	8472.009
bier127_gen1_m2	126	2	29570.5	$gen1$	106	104.8	1153.874
bier127_gen1_m3	126	3	19713.7	$gen1$	103	102.4	591.889
bier127_gen2_m2	126	2	29570.5	$gen2$	5464	5446.8	1132.566
bier127_gen2_m3	126	3	19713.7	$gen2$	5393	5376.2	648.084
bier127_gen2_m4	120	4	14785.2	$gen2$	5122	5119.2	657.569
bier127_gen3_m2	126	2	29570.5	$gen3$	2885	2884.3	1301.269
bier127_gen3_m3	126	3	19713.7	$gen3$	2706	2703.8	711.736
bier127_gen3_m4	120	4	14785.2	$gen3$	2402	2384.6	680.789
gil262_gen1_m3	210	3	396.333	$gen1$	101	100.9	1769.314
gil262_gen1_m4	102	4	297.25	$gen1$	78	77.1	155.755
gil262_gen2_m2	261	2	594.5	$gen2$	7498	7457.8	7356.652
gil262_gen2_m3	210	3	396.333	$gen2$	5615	5608.2	3304.551
gil262_gen3_m2	261	2	594.5	$gen3$	7183	7182.8	9129.303
gil262_gen3_m4	102	4	297.25	$gen3$	2507	2499.8	276.424
gr229_gen1_m4	227	4	441.25	$gen1$	223	220.8	11922.016
gr229_gen2_m3	228	3	588.333	$gen2$	11566	11551.3	14197.206
gr229_gen2_m4	227	4	441.25	$gen2$	11355	11255.5	18799.5
gr229_gen3_m3	228	3	588.333	$gen3$	8056	8051.6	14090.055
gr229_gen3_m4	227	4	441.25	$gen3$	7621	7600	11399.708
kroA150_gen2_m2	149	2	6631	$gen2$	4335	4334.4	892.981
kroA150_gen3_m3	127	3	4420.67	$gen3$	2726	2719.6	538.011
kroA200_gen1_m4	132	4	3671	$gen1$	81	80.4	560.285
kroB200_gen1_m2	199	2	7359.5	$gen1$	111	110.4	2344.534
kroB200_gen2_m2	199	2	7359.5	$gen2$	6185	6182.2	3467.258
kroB200_gen2_m4	128	4	3679.75	$gen2$	4944	4942.2	640.664
kroB200_gen3_m2	199	2	7359.5	$gen3$	4765	4757.8	6306.618
kroB200_gen3_m3	157	3	4906.33	$gen3$	3028	3016	1713.877
lin318_gen1_m2	317	2	10522.5	$gen1$	180	170.1	20667.243
lin318_gen1_m3	256	3	7015	$gen1$	149	148.6	9014.643
lin318_gen2_m2	317	2	10522.5	$gen2$	9544	9533.8	23804.82
lin318_gen2_m3	256	3	7015	$gen2$	7786	7782.1	9773.63

continued on next page

Table 11 – continued

Instance	n	m	L	gen	PSOiA		
					Z_{max}	Z_{avg}	CPU_{avg}
lin318_gen3_m2	317	2	10522.5	$gen3$	7936	7905.6	44029
lin318_gen3_m4	154	4	5261.25	$gen3$	3797	3796.4	1446.258
pr136_gen1_m2	131	2	24193	$gen1$	63	62.7	451.134
pr136_gen2_m2	131	2	24193	$gen2$	3641	3631.8	601.312
pr264_gen1_m4	118	4	6142	$gen1$	107	106.6	503.071
pr264_gen2_m2	131	2	12284	$gen2$	6635	6634.2	2048.2
pr264_gen2_m3	131	3	8189.33	$gen2$	6420	6410.7	938.394
pr264_gen2_m4	118	4	6142	$gen2$	5584	5564.5	590.787
pr264_gen3_m3	131	3	8189.33	$gen3$	2772	2770	1037.505
pr299_gen1_m2	251	2	12048	$gen1$	139	138.5	4775.928
pr299_gen1_m3	162	3	8032	$gen1$	111	110.1	1303.726
pr299_gen1_m4	112	4	6024	$gen1$	84	83.6	383.479
pr299_gen2_m3	162	3	8032	$gen2$	6018	5966.7	1446.047
pr299_gen2_m4	112	4	6024	$gen2$	4457	4453	593.41
pr299_gen3_m2	251	2	12048	$gen3$	5729	5728.6	11872.546
pr299_gen3_m3	162	3	8032	$gen3$	3655	3611	2705.815
pr299_gen3_m4	112	4	6024	$gen3$	2268	2258	455.639
rat195_gen2_m2	190	2	581	$gen2$	5148	5145.6	2156.983
rat195_gen3_m3	122	3	387.333	$gen3$	2574	2571.2	721.818
ts225_gen2_m2	217	2	31661	$gen2$	5859	5858.5	2998.427
rd400_gen2_m2	399	2	3820.5	$gen2$	12993	12787.5	77049.22
rd400_gen2_m3	399	3	2547	$gen2$	12645	12372.1	53707.14
rd400_gen2_m4	399	4	1910.25	$gen2$	12032	11953.5	42001.58
rd400_gen1_m2	399	2	3820.5	$gen1$	230	227.8	56767.29
rd400_gen1_m3	399	3	2547	$gen1$	222	221.7	62476.08
rd400_gen1_m4	399	4	1910.25	$gen1$	213	210.6	34744.8
rd400_gen3_m2	399	2	3820.5	$gen3$	12428	12274.1	96178.7
rd400_gen3_m3	399	3	2547	$gen3$	11639	11629.5	68074.77
rd400_gen3_m4	399	4	1910.25	$gen3$	10417	10383.1	48462.77
ARPE/CPUavg						0.46	11031.04

References

[1] J. Aráoz, E. Fernández, and O. Meza. Solving the prize-collecting rural postman problem. *European Journal of Operational Research*, 196(3):886–896, 2009.

[2] C. Archetti, A. Hertz, and M. Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1), February 2007.

[3] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization. part I: background and development. *Natural Computing*, 6(4):467–484, 2007.

[4] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization. part II: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, 7(1):109–124, 2008.

[5] J. Beasley. Route-first cluster-second methods for vehicle routing. *Omega*, 11:403–408, 1983.

[6] J.-F. Bérubé, M. Gendreau, and J.-Y. Potvin. An exact epsilon-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits. *European Journal of Operational Research*, 194(1):39–50, 2009.

[7] L. Bonnefoy. L'optimisation par essaims particulaires appliquée au team orienteering problem. Preprint available at: http://ludovicbonnefoy.files.wordpress.com/2010/10/majecstic2010.pdf, 2010.

[8] H. Bouly, A. Moukrim, D. Chanteur, and L. Simon. Un algorithme de destruction/construction itératif pour la résolution d'un problème de tournées de véhicules spécifique. In *MOSIM'08*, 2008.

[9] H. Bouly, D.-C. Dang, and A. Moukrim. A memetic algorithm for the team orienteering problem. *4OR*, 8(1):49–70, 2010.

[10] S. Boussier, D. Feillet, and M. Gendreau. An exact algorithm for team orienteering problems. *4OR*, 5(3):211–230, 2007.

[11] S. Butt and T. Cavalier. A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research*, 21:101–111, 1994.

[12] S. E. Butt and D. M. Ryan. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26:427–441, 1999.

[13] I.-M. Chao, B. Golden, and E. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88: 464–474, 1996.

[14] I.-M. Chao, B. Golden, and E. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88:475–489, 1996.

[15] N. Christofides, A.Mingozzi, and P.Toth. *Combinatorial Optimization*, chapter The Vehicle Routing Problem, pages 315–338. Wiley, 1979.

[16] D.-C. Dang, R. N. Guibadj, and A. Moukrim. A pso-based memetic algorithm for the team orienteering problem. In *EvoApplications*, pages 471–480, 2011.

[17] C. Duhamel, P. Lacomme, and C. Prodhon. Efficient frameworks for greedy split and new depth first search split procedures for routing problems. *Computers & Operations Research*, 38(4):723–739, 2011.

[18] M. Fischetti, J. J. S. González, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

[19] B. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[20] K. Kameyama. Particle swarm optimization - a survey. *IEICE Transactions*, 92-D(7):1354–1361, 2009.

[21] L. Ke, C. Archetti, and Z. Feng. Ants can solve the team orienteering problem. *Computers & Industrial Engineering*, 54 (3):648–665, 2008.

[22] C. P. Keller. Algorithms to solve the orienteering problem: A comparison. *European Journal of Operational Research*, 41 (2):224–231, 1989.

[23] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[24] A. C. Leifer and M. B. Rosenwein. Strong linear programming relaxations for the orienteering problem. *European Journal of Operational Research*, 73(3):517–523, 1994.

[25] S. Muthuswamy and S. Lam. Discrete particle swarm optimization for the team orienteering problem. *Memetic Computing*, 3:287–303, 2011.

[26] M. Pant, R. Thangaraj, and A. Abraham. A new pso algorithm with crossover operator for global optimization problems. In *Innovations in Hybrid Intelligent Systems*, pages 215–222. Springer Berlin Heidelberg, 2008.

[27] M. Poggi de Aragão, H. Viana, and E. Uchoa. The team orienteering problem: Formulations and branch-cut and price. In *ATMOS*, pages 142–155, 2010.

[28] C. Prins, N. Labadi, and M. Reghioui. Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2):507–535, 2009.

[29] G. Reinelt. A traveling salesman problem library. *ORSA Journal on Computing*, 1991.

[30] R. Sadykov and F. Vanderbeck. Bin packing with conflicts: a generic branch-and-price algorithm. Preprint accepted for publication in INFORMS Journal on Computing, 2012.

[31] D. Y. Sha and C.-Y. Hsu. A hybird particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51(4):791–808, 2006.

[32] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden. A path relinking approach for the team orienteering problem. *Computers & Operations Research*, 37(11):1853–1859, 2010.

[33] H. Tang and E. Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Computer & Operations Research*, 32:1379–1407, 2005.

[34] R. E. Tarjan. Graph theory and gaussian elimination. Technical report, Stanford University, 1975.

[35] F. Tricoire, M. Romauch, K. F. Doerner, and R. F. Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37:351–367, 2010.

[36] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[37] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118 – 127, 2009.

[38] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. Metaheuristics for tourist trip planning. In *Metaheuristics in the Service Industry*, volume 624 of *Lecture Notes in Economics and Mathematical Systems*, pages 15–31. Springer Berlin Heidelberg, 2009.

[39] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[40] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. Technical report, CIRRELT, 2012.

[41] T. Yamada, S. Kataoka, and K. Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43(9):2864–2870, 2002.

[42] X. Zhao. A perturbed particle swarm algorithm for numerical optimization. *Applied Soft Computing*, 10(1):119–124, 2010.