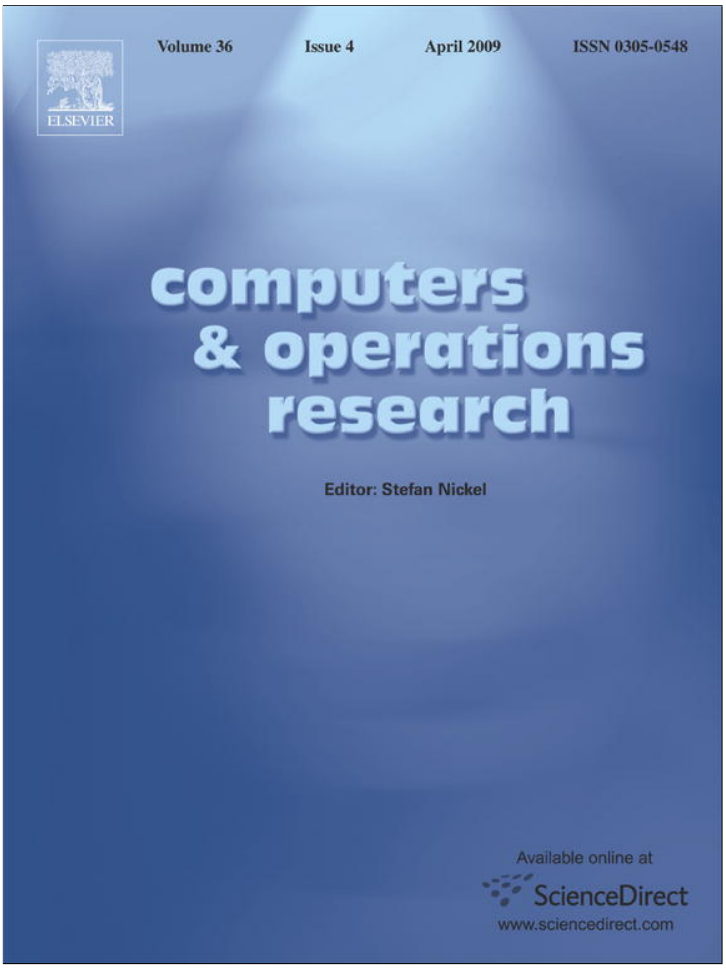

Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 36 (2009) 1191–1203
computers & operations research
www.elsevier.com/locate/cor

Decremental state space relaxation strategies and initialization heuristics for solving the Orienteering Problem with Time Windows with dynamic programming

Giovanni Righini*, Matteo Salani[1]

Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Via Bramante 65, 26013 Crema (CR), Italy

Available online 25 January 2008

Abstract

We present an exact optimization algorithm for the Orienteering Problem with Time Windows (OPTW). The algorithm is based on bi-directional and bounded dynamic programming with decremental state space relaxation. We compare different strategies proposed in the literature to guide decremental state space relaxation: our experiments on instances derived from the literature show that there is no dominance between these strategies. We also propose a new heuristic technique to initialize the critical vertex set and we provide experimental evidence of its effectiveness.

Keywords: Combinatorial optimization; Traveling salesman problem; Shortest path problem; Dynamic programming

1. Introduction

The Orienteering Problem with Time Windows (OPTW) is a combinatorial problem falling into the realm of Traveling Salesman Problems (TSPs) with profits. It can be formulated as a special case of the Resource Constrained Elementary Shortest Path Problem (RCESPP), for which effective dynamic programming algorithms have been recently proposed by Boland et al. [1] and by Righini and Salani [2]. These algorithms exploit a new technique, called *decremental state space relaxation* (DSSR) [2] or *state space augmentation* [1]. The purpose of this paper is twofold: first, to apply dynamic programming (DP) with DSSR to solve the OPTW; second, to present some new ideas concerning the heuristic initialization of the critical vertex set in DSSR algorithms.

1.1. Literature review

The TSP requires the computation of a minimum cost Hamiltonian cycle on an undirected graph. A large number of extensions of this problem have been proposed in the literature so far: for a detailed review we refer the reader to the survey paper by Laporte [3]. Several of these extensions belong to the class of TSP with profits (see Feillet et al. [4]), where a profit is associated with each vertex and the goal is to find a cycle, not necessarily Hamiltonian,

* Corresponding author. Tel.: +39 373 898060; fax: +39 373 898010.
E-mail addresses: righini@dti.unimi.it (G. Righini), matteo.salani@epfl.ch (M. Salani).
[1] Currently at Ecole Polytechnique Fédérale de Lausanne, Switzerland.

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.01.003

starting from a given vertex and such that the overall collected profit is maximum. The TSP with profits is particularly relevant because it arises as a pricing subproblem when vehicle routing problems with additional constraints are solved via column generation. The Orienteering Problem (OP), also called selective TSP, is the problem of maximizing the collected profits subject to a constraint on the maximum allowed tour length. The OP was introduced by Tsiligrides [5] and surveyed by Golden et al. [6]. A first exact algorithm for the OP was proposed by Ramesh et al. [7]. More recently Fischetti et al. [8] presented a branch-and-cut algorithm able to solve 500 cities instances to optimality in some hours. Another branch-and-cut algorithm was presented by Gendreau et al. [9] for the OP with compulsory vertices.

In this paper we consider the OPTW: a profit, a time window and a service time are associated with each vertex and a traveling time is associated with each edge. The objective is to find a maximum profit tour such that each vertex is either visited inside its time window or skipped. We are aware of only two papers on this problem, both dealing with approximation and heuristics: Kantor and Rosenwein [10] presented a so-called "tree heuristic", which only (approximately) solved small instances; more recently Bar-Yehuda et al. [11] studied geometric versions of the OPTW, with customers located on a line or in the Euclidean plane, and without the limit on the maximum allowed tour duration. The multi-vehicle version of the OP, with and without time windows, has recently been studied by Boussier et al. [12].

The OPTW can be reformulated as a RCESPP, which in turn can be effectively solved by DP with DSSR, a method presented in Righini and Salani [2]. DSSR consists in solving through DP a relaxed problem, in which it is allowed to visit vertices (and to get the corresponding profit) more than once. This relaxation is iteratively tightened by forbidding multiple visits to a *critical vertex set* of increasing size. Computational experience shows that an optimal elementary path is obtained after defining as critical a rather small fraction of the vertices. The same idea, named state space augmentation, was independently developed and presented by Boland et al. [1], who also compared different strategies to define new critical vertices at each iteration.

1.2. Original contributions

In this paper we present an exact optimization algorithm for the OPTW based on DP with DSSR and we compare the strategies proposed by Boland et al. by testing them on the OPTW using Solomon's data-sets and other instances derived from a data-set of Cordeau et al. [13]. Our result is that there is no domination between the different strategies.

In addition, we give a further methodological contribution along this research stream: we propose a new heuristic technique for the initialization of the critical vertex set, and we show that it reduces the number of iterations and the amount of computing time needed by the DSSR algorithm to converge to an optimal solution. In our experiments the speed-up obtained in this way positively affected all the DSSR strategies and for some data-sets it was very significant, in excess of 60%.

1.3. Paper outline

In Section 2 we give a formal statement of the problem; in Section 3 we present a DP algorithm for the OPTW; in Section 4 we present the DSSR algorithm and we report on computational experiments; in Section 5 we present the ideas to initialize the critical vertex set and we report on related computational experiments.

2. Problem definition

The OPTW is defined as follows. We are given a complete undirected graph $\mathscr{G} = (\mathscr{N}, \mathscr{E})$, with a positive weight t_{ij} associated with each edge, representing the travel time between vertices i and j. For each vertex $i \in \mathscr{N}$ we have the following data: p_i is a positive profit that is collected when the vertex is visited, $[a_i, b_i]$ is a time window defining the feasible arrival time at the vertex and s_i is a non-negative service time, that is the amount of time which is spent to visit the vertex. Two special vertices, numbered 1 and n, where $n = |\mathscr{N}|$, are the endpoints of the path to be computed. We have $p_1 = p_n = 0$, $s_1 = s_n = 0$, $a_1 = a_n = 0$ and $b_1 = b_n = T$, where T is equal to the maximum feasible arrival time at vertex n, that is $T = \max_{i \in \mathscr{N} \setminus \{1,n\}} \{b_i + s_i + t_{in}\}$.

We indicate with τ_i the arrival time at vertex i. The OPTW requires the computation of an elementary path $\mathscr{P}$ defined as an ordered sequence of vertices, starting from node 1, ending at node n, maximizing $\sum_{i \in \mathscr{P}} p_i$ and such that $a_i \leqslant \tau_i \leqslant b_i, \ \forall i \in \mathscr{P}$. For each pair of vertices (i, j) consecutively visited along $\mathscr{P}$ we have $\tau_j = \max\{\tau_i + s_i + t_{ij}, a_j\}$.

3. DP for the OPTW

In DP algorithms for computing optimal paths, a *state* associated with vertex i represents a path starting at vertex 1 and ending at vertex i. The DP algorithm repeatedly extends each state to generate new states. The extension of a state corresponds to adding a vertex to a path. To limit the exponential increase of the number of states, suitable dominance criteria are applied to identify states whose extension cannot produce an optimal solution. Recent references on this topic are the papers by Feillet et al. [14], Irnich and Desaulniers [15] and Boland et al. [1].

Applying the same idea to the OPTW, one can design a basic DP algorithm, which generates and extends states represented by *labels*, that is tuples of the form (S, τ, P, i), where S is a binary vector representing the subset of vertices already visited, τ is the overall time elapsed, P is the overall profit collected, and i is the last reached vertex.

This is equivalent to say that the OPTW is a special case of the RCESPP, a general and fundamental $\mathcal{NP}$-hard network optimization problem, often encountered as a subproblem of more complex routing problems. In the OPTW we have two *resources*: one represents time and its consumption is indicated by τ; the other is a dummy resource, whose consumption is indicated by S: there is one resource unit available at each vertex and it is consumed when the vertex is visited. This method was introduced by Beasley and Christofides [16] to enforce the elementary path constraint.

The RCESPP has recently been attacked by new DP algorithms developed by Dumitrescu and Boland [17] and Righini and Salani [18]. Hereafter we apply to the OPTW the bi-directional and bounded DP method presented in [18], which we briefly recall to make this paper self-contained. In bi-directional DP the extension of states is done both forward from vertex 1 and backward from vertex n. Intuitively the idea is to develop two smaller sets of states instead of one larger set. Bi-directional search is coupled with bounding, that is the extension of the paths is stopped at a certain state, and when there is the guarantee that if the state belongs to an optimal path, then the remaining part of that path has been or will be generated in the other direction. To decide when a path can be stopped, a critical resource is identified and no path is allowed to exceed a consumption of the critical resource equal to half the overall quantity of available resource. This technique requires to match forward and backward paths to yield complete solutions.

Hereafter we give the details on extension rules, dominance tests and matching procedures of forward and backward paths. Extension rules and dominance tests are those commonly used in all DP algorithms to compute shortest paths with resource constraints, as those presented in the papers cited here above; matching procedures of forward and backward paths are taken from Righini and Salani [18].

3.1. Extension rules

When a label (S, τ, P, i) associated with vertex i is extended to vertex j, it generates a new label (S', τ', P', j) according to the following rules.

The prize P, initialized to 0 at vertex 1, is updated according to the formula

$$P' = P + p_i/2 + p_j/2.$$

This definition allows to preserve symmetry in the bi-directional search algorithm illustrated in the remainder.

The vector S is initialized to 0 and the update rule is

$$S'_k = \begin{cases} S_k + 1, & k = j, \\ S_k, & k \neq j. \end{cases}$$

A state (S, τ, P, i) can be extended to vertex j only if $S_j = 0$.

The consumption of time resource τ is updated according to the direction of the extension. We define the time window $[a_i^{\mathrm{bw}}, b_i^{\mathrm{bw}}]$ representing the backward time window of vertex i: it is obtained by adding the service time s_i to the forward time window $[a_i, b_i]$ for each $i \notin \{1, n\}$.

For forward extensions we have

$$\tau' = \max\{\tau + s_i + t_{ij}, a_j^{\mathrm{fw}}\}$$

and for backward extensions we have

$$\tau' = \max\{\tau + s_j + t_{ij}, T - b_i^{\mathrm{bw}}\}.$$

A forward state (S, τ, P, i) is feasible only if $\tau \leqslant b_i$; a backward state (S, τ, P, i) is feasible only if $\tau \leqslant T - a_i^{\mathrm{bw}}$.

3.2. Dominance tests

Dominance tests are always performed when states are extended, so that the algorithm records only non-dominated states. The dominance test is the following. Let $L_1 = (S_1, \tau_1, P_1, i)$ and $L_2 = (S_2, \tau_2, P_2, i)$ be the labels of two states associated with vertex i and both generated in the same direction; then L_1 dominates L_2 only if

$$\begin{cases} S_1 \leqslant S_2, \\ \tau_1 \leqslant \tau_2, \\ P_1 \geqslant P_2 \end{cases}$$

and at least one of the inequalities is strict.

3.3. Matching forward and backward states

Forward and backward states are matched together to form complete paths from vertex 1 to vertex n. When matching a forward path $(S^{\mathrm{fw}}, \tau^{\mathrm{fw}}, P^{\mathrm{fw}}, i)$ with a backward path $(S^{\mathrm{bw}}, \tau^{\mathrm{bw}}, P^{\mathrm{bw}}, j)$ the feasibility conditions are the following:

$$\begin{cases} S_k^{\mathrm{fw}} + S_k^{\mathrm{bw}} \leqslant 1 & \forall\, k \in \mathcal{N}, \\ \tau^{\mathrm{fw}} + s_i + t_{ij} + s_j + \tau^{\mathrm{bw}} \leqslant T \end{cases}$$

and the overall profit of the resulting path is $P = P^{\mathrm{fw}} + p_i/2 + p_j/2 + P^{\mathrm{bw}}$.

4. Decremental state space relaxation

State space relaxation (SSR) was introduced by Christofides et al. [19] to reduce the number of states to be explored; with SSR the search space explored by DP is projected onto a lower dimensional space so that only the minimum cost state is retained among all the corresponding states in the higher dimensional space. The main drawback of this method is that some original state corresponding to an infeasible solution may be projected onto a state corresponding to a feasible solution in the lower dimensional space. Therefore the search in the lower dimensional state space does not guarantee to find an optimal solution but rather a dual bound. In the case of OPTW we apply state space relaxation to the binary vector S, replacing it with the number of visited vertices

$$\sigma = \sum\nolimits_{i \in \mathcal{N}} S_i,$$

which is constrained to be less than or equal to n to prevent unboundedness. In this way we neglect the information on the vertices already visited and this results in an algorithm where cycles are no longer forbidden. The dominance test between two labels $L1$ and $L2$ now results as follows:

$$\begin{cases} \sigma_1 \leqslant \sigma_2, \\ \tau_1 \leqslant \tau_2, \\ P_1 \geqslant P_2. \end{cases}$$

In Righini and Salani [2] the authors introduced DSSR, with the idea of iteratively reducing the relaxation of the state space as needed, according to the structure of the optimal solution of the relaxed problem. Boland et al. [1] independently proposed the same idea, calling it *state space augmenting* algorithm. In both cases the idea is to start with a relaxation of the whole state space induced by the set of binary variables S and to tighten this relaxation at each iteration as long as the DP algorithm returns an optimal path with cycles. Let us define Θ as the *critical vertex set*, that is the set of vertices for which the elementary path constraint is enforced, forbidding multiple visits. The set Θ is initially empty, that is the path can visit all vertices more than once. If the optimal solution of this relaxed problem is feasible, then it is also optimal for the original problem; otherwise some vertices are identified as critical, they are inserted into Θ, thus augmenting the search space of the DP algorithm. The loop is repeated until the optimal solution of the relaxed problem turns out to be elementary. Every time Θ is enlarged, the number of dummy resources (i.e. the number of components of vector S) increases. In turn this increases the number of states that DP must consider and requires additional memory space and computing time.

In their paper [1] Boland et al. compared different ways of inserting new vertices into the critical vertex set. The main strategies they took into account are the following:

- HMO: insert one vertex at a time, selecting the vertex visited the largest number of times. In case of *ex aequo*, choose one at random;
- HMO-All: insert all vertices visited the maximum number of times;
- MO-All: insert all vertices visited more than once in an optimal path; this is also the strategy used by Righini and Salani [2];
- M-All: insert all vertices visited more than once in any Pareto-optimal path; we did not consider this strategy because our bi-directional algorithm does not generate all the Pareto-optimal paths at the final vertex.

In this paper we use the same notation introduced in [1] and we compare the first three strategies above.

4.1. Primal solutions

At each iteration of the DSSR algorithm we solve a relaxation of the original problem, which yields an upper bound to the optimal solution. Given a feasible solution of the relaxed problem, we compute a feasible solution for the original problem by skipping all the vertices that are visited more than once. The feasibility with respect to the time window constraints is guaranteed owing to the triangular inequality: skipping vertices can only reduce the traveling time. More sophisticated primal solutions can be computed from the solution of the relaxed problem but this goes beyond the scope of this work. This heuristic gives us a benchmark to evaluate an optimality gap at each iteration of the DSSR algorithm.

4.2. Data-sets

We tested our algorithms on two classes of instances obtained from the well-known Solomon's data-set of VRPTW instances and from instances proposed by Cordeau et al. [13] for the Multi-Depot Periodic Vehicle Routing Problem (MDPVRP). The first data-set, composed by 29 instances, has been made by considering the first 100 vertices of Solomon's instances. Depending on the displacement of the customers, this data-set is divided into *random*, *clustered* and *random–clustered* categories. Instances belonging to the same category have the customers located in the same way and with the same demands; they differ only for the time windows. The other data-set has been derived from Cordeau's 20 instances data-set, considering all customers active in the same day. We considered the delivery demand associated with vertex i in the original data-set as the prize p_i for that vertex. Cordeau's instances are definitely harder than those of Solomon: in this data-set there are up to 288 customers and the time windows are larger. Optimal solutions of the Solomon's instances contain 9–13 customers, while the largest optimal solutions we could solve in Cordeau's data-set contain 31 vertices.

Other authors solved the OP (or equivalently the resource constrained shortest path problem), testing their algorithms on particular data-sets. For instance, Boland et al. [1] evaluated their algorithms for the OP with capacities and no time windows on data-sets made of randomly generated graphs with varying size, density and number of negative cost arcs. We did not make additional experiments using these particular data-sets for two main reasons: first, because the problem considered is different and hence a meaningful comparison with the results reported in [1] would not have been possible even if we had used the same instances; second, because the main purpose of this paper is not to compete with other exact approaches for the solution of the OPTW, but rather to illustrate the effectiveness of some algorithmic ideas, and this can be clearly appreciated from the results obtained with the well-known data-sets we have used, as reported in the remainder.

4.3. Experimental results

All tests were performed on a PC equipped with a Pentium IV 1.6 GHz processor with 512 MB RAM. The algorithms were coded in ANSI-C and compiled with *gcc* 3.0.4 and were run with a time limit of 2 h.

Tables 1 and 2 report on the experimental comparison between the three DSSR algorithms and the basic DP algorithm described in Section 3. The first three columns report the instance name, the optimal solution, and the number of vertices

Table 1
Solomons's instances–100 vertices

Instance name	Optimum		DSSR HMO				DSSR HMO-All				DSSR MO-All				Basic D.P.
	Value	Vert.	It.	Θ	Time	(%)	It.	Θ	Time	(%)	It.	Θ	Time	(%)	Time
c101_100	320	10	**1**	**0**	0.07	0.0	**1**	**0**	**0.06**	0.0	**1**	**0**	**0.06**	0.0	0.14
c102_100	360	11	4	**3**	5.33	0.0	**3**	**3**	**3.81**	0.0	**3**	**3**	4.49	0.0	–
c103_100	400	11	9	**8**	**1081.04**	0.0	8	9	1393.38	0.0	**6**	9	1101.74	0.0	–
c104_100	420	11	9	**8**	2141.38	0.0	7	**8**	**1856.39**	0.0	**6**	9	2166.79	0.0	–
c105_100	340	10	**1**	**0**	**0.12**	0.0	**1**	**0**	0.13	0.0	**1**	**0**	**0.12**	0.0	0.3
c106_100	340	10	**1**	**0**	**0.14**	0.0	**1**	**0**	0.15	0.0	**1**	**0**	0.15	0.0	0.39
c107_100	370	11	**1**	**0**	**0.20**	0.0	**1**	**0**	0.20	0.0	**1**	**0**	0.20	0.0	0.51
c108_100	370	11	**4**	**3**	1.43	0.0	**4**	4	1.47	0.0	**4**	4	1.46	0.0	**0.93**
c109_100	380	11	12	**11**	14.01	0.0	8	13	10.65	0.0	**8**	13	**10.57**	0.0	11.67
r101_100	198	9	**1**	**0**	0.04	0.0	**1**	**0**	**0.03**	0.0	**1**	**0**	0.04	0.0	0.08
r102_100	286	11	7	**6**	**233.20**	0.0	7	7	260.04	0.0	**5**	8	310.79	0.0	–
r103_100	293	11	10	**9**	**5498.81**	0.0	–	–	–	0.3	**7**	10	5729.01	0.0	–
r104_100	303	13	–	–	–	**2.5**	–	–	–	**2.5**	–	–	–	**2.5**	–
r105_100	247	11	3	**2**	0.35	0.0	**2**	**3**	**0.23**	0.0	**2**	**3**	**0.23**	0.0	0.24
r106_100	293	11	8	**7**	579.44	0.0	8	8	634.89	0.0	**5**	8	**334.49**	0.0	–
r107_100	299	13	9	**8**	**2979.94**	0.0	9	10	3483.73	0.0	**6**	11	3514.80	0.0	–
r108_100	308	13	–	–	–	16.8	–	–	–	**2.5**	–	–	–	**2.5**	–
r109_100	277	12	11	10	6.87	0.0	6	9	3.63	0.0	**5**	**9**	3.09	0.0	**1.67**
r110_100	284	13	10	**9**	78.52	0.0	7	10	72.27	0.0	**4**	**9**	**30.83**	0.0	79.05
r111_100	297	12	11	**10**	1932.55	0.0	9	10	1807.86	0.0	**7**	12	**1408.80**	0.0	–
r112_100	298	12	12	**11**	2624.42	0.0	9	**11**	**2508.17**	0.0	**7**	14	3177.02	0.0	–
rc101_100	219	9	4	**3**	0.31	0.0	**3**	**3**	0.24	0.0	**3**	**3**	0.23	0.0	**0.14**
rc102_100	266	10	8	**7**	**6.11**	0.0	8	9	8.68	0.0	**6**	11	9.88	0.0	12.75
rc103_100	266	10	14	**13**	**88.12**	0.0	12	14	99.25	0.0	**9**	18	111.44	0.0	401.34
rc104_100	301	11	12	**11**	304.42	0.0	11	11	268.63	0.0	**7**	16	**264.84**	0.0	–
rc105_100	244	11	8	**7**	2.86	0.0	8	9	3.08	0.0	**7**	10	2.95	0.0	**1.93**
rc106_100	252	11	11	**10**	3.64	0.0	7	10	2.24	0.0	**6**	13	2.08	0.0	**0.90**
rc107_100	277	10	14	**13**	50.76	0.0	14	13	50.82	0.0	**10**	19	**49.19**	0.0	59.16
rc108_100	298	11	10	**9**	77.77	0.0	9	10	71.10	0.0	**7**	14	**68.95**	0.0	959.45
Average			7.59	**6.59**	**1107.31**	0.66 (27)	6.31	7.08	1177.28	0.18 (26)	**5.00**	8.37	1127.73	**0.17** (27)	3032.05 (17)

Table 2
Cordeau's instances

Instance name	Optimum		DSSR HMO				DSSR HMO-All				DSSR MO-All				Basic D.P.
	Value	Vert.	It.	Θ	Time	(%)	It.	Θ	Time	(%)	It.	Θ	Time	(%)	Time
pr01_48	308	21	13	**12**	3.79	0.0	10	12	2.93	0.0	**4**	15	1.19	0.0	**0.70**
pr02_96	404	24	22	**21**	101.78	0.0	13	23	68.75	0.0	**6**	24	37.52	0.0	**30.70**
pr03_144	394	22	26	**25**	442.45	0.0	19	27	318.79	0.0	**8**	38	**151.73**	0.0	265.72
pr04_192	489	24	38	**37**	3152.74	0.0	20	39	1639.49	0.0	**7**	40	**648.82**	0.0	1084.80
pr05_240	595	31	–	–	–	12.2	–	–	–	9.3	**7**	**40**	**6815.82**	0.0	–
pr06_288	(501)	(26)	–	–	–	**29.7**	–	–	–	47.9	–	–	–	45.1	–
pr07_72	298	17	21	**20**	12.13	0.0	12	22	6.85	0.0	**6**	21	3.65	0.0	**1.51**
pr08_144	463	25	25	**24**	338.25	0.0	11	25	131.94	0.0	**6**	31	**90.71**	0.0	128.60
pr09_216	493	29	–	–	–	3.3	17	**26**	3988.45	0.0	**8**	36	**3270.88**	0.0	–
pr10_288	(584)	(32)	–	–	–	37.3	–	–	–	11.0	–	–	–	**1.0**	–
Average			24.17	**23.17**	3285.11	8.25 (6)	14.57	24.86	2775.72	6.82 (7)	**6.50**	30.63	**2542.03**	**4.61** (8)	3031.20 (6)

in the optimal solution. If the optimal value is not known, the best known solution is reported within parentheses. Next, for the three strategies, we report the number of iterations required, the number of vertices added to the critical set Θ, the computing time in seconds, and the percentage gap between the best known solution and the lower bound obtained in the last iteration within the time limit.

The last column reports the computing time required by the basic DP algorithm. Empty cells in the "Time" column mean that the corresponding instances were not solved to optimality within the time limit. The last two rows report the average values and, within parentheses, the number of instances solved within the time limit. The average values for iterations and number of critical vertices have been computed over the number of solved instances. On the other hand the average values for computing times and gaps have been computed over the whole set of instances setting the computing time of the unsolved instances equal to the time limit.

The basic DP algorithm solved only 17 of the 29 Solomon's instances with 100 vertices, while DSSR allowed to solve 27 of them; none of the proposed algorithms solved instances R104_100 and R108_100 within 2 h, but the smallest gaps for the unsolved instances were obtained by DSSR MO-All. Cordeau's instances are harder to solve: all algorithms failed to provide reasonable solutions for instances $pr11$ to $pr20$ which are not reported here. On Cordeau's data-set DSSR MO-All dominated the other DSSR strategies with the only exception of instance $pr06$, which is the most difficult one reported here. In this case the HMO policy achieved the best result with a gap as large as 29.7%.

From the examination of the tables we can observe some regularity in the effects produced by the different DSSR methods. The HMO policy always provides the smallest critical vertex set, as it is a conservative approach: it inserts only one critical vertex at each iteration, thus reducing the redundancy of the critical set. The drawback is that it always needs a number of iterations equal to the size of the final critical set plus one. The MO-All strategy behaves in a complementary way: it constantly requires the smallest number of iterations but it tends to add useless vertices to the critical set, so that the size of the critical set is larger, in average, than that obtained with the HMO policy. For Cordeau's data-set this difference is particularly evident: the HMO strategy required, in average, 24.17 iterations compared to 6.50 required by the MO-All strategy. There is a trade-off between the increase in the number of iterations and the increase in the cardinality of the critical set, and in general there is no clear domination between policies HMO and MO-All. We observe that HMO-All was almost always dominated either by MO-All or by HMO.

In their experiments, made on different randomly generated data-sets, Boland et al. [1] observed that the most conservative strategy HMO always dominated the others. This was not the case in our experiments: in particular HMO was never the best strategy on any instance of Cordeau's data-set.

In the next section we propose some new ideas to initialize the critical vertex set in order to reduce the number of iterations and the computing time needed by DSSR algorithms.

5. Initialization of the critical vertex set

We observed that the DSSR HMO strategy reduces the number of critical vertices and increases the number of iterations needed, while the DSSR MO-All strategy reduces the number of iterations against an increase of the number of critical vertices. We investigated how to initialize the set Θ in a preprocessing phase, to identify a subset of vertices that have a high probability to belong to the final critical vertex set. Let us define f_{ij} to be a measure of the "cycling attractiveness" of a vertex i with respect to a vertex j as the ratio of the prize p_i over the duration of the cycle i–j–i:

$$f_{ij} = p_i/(s_i + t_{ij} + s_j + t_{ji}).$$

Now we can define an ordering of the vertices based on the following criteria:

- Highest cycling attractiveness (HCA): order by $\max_{j\in\mathcal{N}\setminus\{i\}}\{f_{ij}\}$.
- Total cycling attractiveness (TCA): order by

$$\sum_{j\in\mathcal{N}\setminus\{i\}} f_{ij}.$$

- Weighted highest cycling attractiveness (WHCA): order by $\max_{j\in\mathcal{N}\setminus\{i\}}\{f_{ij}(b_i - a_i)\}$.

- Weighted total cycling attractiveness (WTCA): order by

$$\sum_{j \in \mathcal{N} \setminus \{i\}} f_{ij}(b_i - a_i).$$

5.1. Experimental results

Tables 3–6 report on the average computational results obtained applying the four initialization strategies to Solomon's instances with 100 vertices (Tables 3 and 4) and Cordeau's instances (Tables 5 and 6). For Solomon's data-set we reported averages for each different distribution of customers (C, R, and RC classes). From the examination of the number of vertices in optimal solutions we chose to initialize the critical vertex set with five and 10 vertices. The last row reports also the percentage speed-up on the computing time. We performed our experiments using HMO and MO-All policies, because in the previous tests HMO-All was dominated by one of them in most cases.

We can observe that an initialization of the critical vertex set is useful, because it reduces the overall computing time as well as the size of the final critical set. For Solomon's instances and the HMO policy (Table 3) we could not observe a reduction on the size of the critical set, because the most conservative approach usually finds a minimum critical set. Therefore even with a smart initialization we can expect only a reduction of the number of iterations. This consideration arises clearly analyzing the average values: except the TCA method initialized with five vertices all other methods performed worse in term of average computational time and average size of the critical vertex set while the number of iterations decreased.

For Solomon's data-set and the MO-All policy (Table 4) we could observe a reduction on both the computing time and the size of the critical set. As pointed out in the previous section this policy tends to include in the critical set some vertices which are not needed to compute an elementary path. This undesired behavior is mitigated when the initialization is used.

A general consideration we can derive from the tables concerning Solomon's data-set is that an initialization with five critical vertices can be useful to reduce the computational time. The proposed methods were not able to discover a good initial critical set with 10 vertices.

For Courdeau's instances, where the average length of the optimal path is usually twice with respect to Solomon's instances, the initialization with 10 vertices gave better results. For all instances solved with the HMO policy we could observe a relevant reduction of the number of iterations and of the computational time.

For the MO-All policy, which outperformed HMO without initialization, we observed only negligible improvements in some cases and in general we can conclude that the initialization with 10 vertices performed better.

5.2. Mixed strategy

In general the initialization criteria give different results and none of them is reliable to reveal the necessary vertices to be put in the critical set. Therefore we devised a mixed strategy (MS_m) to initialize the critical vertex set: let us define HCA_m, TCA_m, $WHCA_m$ and $WTCA_m$ to be the sets obtained according to the above criteria considering only the former m vertices in the correspondent ordering. Now we use as an initial critical vertex set the one obtained from the intersection of these four sets: $\Theta_m = HCA_m \cap TCA_m \cap WHCA_m \cap WTCA_m$. By a suitable choice of the value of m the set Θ_m can be initialized with the aim of reducing the number of iterations (high m) or reducing the probability of inserting unnecessary vertices (low m) into it.

5.3. Experimental results

Tables 7 and 8 report on the computational results obtained by MS_m on Solomon's and Cordeau's data-sets with strategies HMO and MO-All. From the examination of the number of vertices in the optimal solutions (reported in Tables 2 and 3) we chose $m = 10$ and 20. The format of Tables 7 and 8 is the same of Tables 1 and 2 but an additional column reports on the number of vertices inserted into the critical set in the initialization phase. The last row reports also the percentage speed-up achieved.

The critical set initialization with the mixed strategy definitely improved the performance of both HMO and MO-All algorithms. Remarkably all the instances were solved within the time limit. Both algorithms needed fewer iterations to converge to an elementary solution.

Table 3
Solomon's instances—100 vertices—DSSR HMO

Instance set	DSSR w/o initialization				DSSR tca 5				DSSR tca 10				DSSR hca 5				DSSR hca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)
C Class (9)	4.7	3.7	360.4	0.0 (9)	2.7	6.7	349.9	0.0 (9)	1.8	10.8	1089.3	0.0 (9)	3.4	7.4	688.2	0.0 (9)	2.3	11.3	555.8	0.0 (9)
R Class (12)	8.2	7.2	2361.2	1.6 (10)	4.4	8.4	2156.7	0.4 (10)	2.9	11.9	2472.2	1.6 (10)	4.7	8.7	2212.7	0.4 (10)	3.6	12.6	2364.3	1.6 (10)
RC Class (8)	10.1	9.1	66.7	0.0 (8)	6.6	10.6	58.6	0.0 (8)	4.3	13.3	65.3	0.0 (8)	6.3	10.3	37.1	0.0 (8)	4.5	13.5	34.5	0.0 (8)
Average (29)	7.6	6.6	1107.3	0.7 (27)	4.5	8.5	1017.2	0.2 (27)	2.9	11.9	1379.1	0.7 (27)	4.7	8.7	1139.4	0.2 (27)	3.4	12.4	1160.3	0.7 (27)
Speed up (%)							8.1				−24.5				−2.9				−4.8	
Instance set	DSSR w/o initialization				DSSR wtca 5				DSSR wtca 10				DSSR whca 5				DSSR whca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)
C Class (9)	4.7	3.7	360.4	0.0 (9)	2.7	6.7	624.4	0.0 (9)	1.6	10.6	1158.7	0.0 (8)	2.8	6.8	611.3	0.0 (9)	1.6	10.6	1058.7	0.0 (8)
R Class (12)	8.2	7.2	2361.2	1.6 (10)	4.8	8.8	2268.6	1.6 (10)	3.3	12.3	2533.8	1.6 (10)	4.7	8.7	2260.1	0.4 (10)	3.2	12.2	2516.6	1.6 (10)
RC Class (8)	10.1	9.1	66.7	0.0 (8)	6.8	10.8	74.1	0.0 (8)	4.5	13.5	121.3	0.0 (8)	6.4	10.4	54.2	0.0 (8)	4.5	13.5	62.6	0.0 (8)
Average (29)	7.6	6.6	1107.3	0.7 (27)	4.7	8.7	1152.9	0.7 (27)	3.2	12.2	1441.5	0.7 (26)	4.6	8.6	1139.9	0.2 (27)	3.1	12.1	1387.2	0.7 (26)
Speed up(%)							−4.1				−30.2				−2.9				−25.3	

Table 4
Solomon's instances—100 vertices—DSSR MO-All

Instance set	DSSR w/o initialization				DSSR tca 5				DSSR tca 10				DSSR hca 5				DSSR hca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)
C Class (9)	3.4	4.2	365.1	0.0 (9)	2.3	7.0	405.3	0.0 (9)	1.7	10.8	1087.6	0.0 (9)	2.8	7.8	610.7	0.0 (9)	1.9	11.4	456.5	0.0 (9)
R Class (12)	4.9	8.4	2409.1	0.4 (10)	3.0	8.8	1988.6	0.4 (10)	2.4	12.1	2316.5	0.4 (10)	3.0	9.0	1857.3	0.4 10)	2.6	12.7	2179.4	0.4 (10)
RC Class (8)	6.9	13.0	63.7	0.0 (8)	4.9	12.4	66.9	0.0 (8)	3.4	14.6	85.1	0.0 (8)	4.9	13.1	48.3	0.0 (8)	3.8	15.1	40.6	0.0 (8)
Average (29)	5.0	8.4	1127.7	0.2 (27)	3.3	9.3	967.1	0.2 (27)	2.4	12.4	1319.5	0.2 (27)	3.5	9.8	971.4	0.2 (27)	2.7	13.0	1054.7	0.2 (27)
Speed up (%)							14.2				−17.0				13.9				6.5	
Instance set	DSSR w/o initialization				DSSR wtca 5				DSSR wtca 10				DSSR whca 5				DSSR whca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time(s)	(%)	It.	Θ	Time (s)	(%)
C Class (9)	3.4	4.2	365.1	0.0 (9)	2.3	7.0	629.8	0.0 (9)	1.5	10.6	1158.9	0.0 (8)	2.3	7.0	444.1	0.0 (9)	1.5	10.6	1058.9	0.0 (8)
R Class (12)	4.9	8.4	2409.1	0.4 (10)	2.9	9.2	2044.1	0.4 (10)	2.3	12.5	2197.2	0.4 (10)	3.0	9.1	2084.0	0.4 (10)	2.2	12.5	2212.7	0.4 (10)
RC Class (8)	6.9	13.0	63.7	0.0 (8)	4.9	12.4	71.0	0.0 (8)	3.8	15.0	141.8	0.0 (8)	5.0	12.4	105.9	0.0 (8)	3.8	14.9	100.9	0.0 (8)
Average (29)	5.0	8.4	1127.7	0.2 (27)	3.3	9.4	1060.9	0.2 (27)	2.5	12.7	1307.9	0.2 (26)	3.4	9.4	1029.4	0.2 (27)	2.5	12.7	1272.1	0.2 (26)
Speed up (%)							5.9				−16.0				8.7				−12.8	

Table 5
Cordeau's instances—DSSR HMO

	DSSR w/o initialization				DSSR tca 5				DSSR tca 10				DSSR hca 5				DSSR hca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)
Average (10)	24.2	23.2	3285.1	8.3 (6)	20.6	24.6	3204.6	7.9 (7)	16.0	25.0	3241.0	4.2 (6)	19.9	23.9	3039.8	7.9 (7)	15.4	24.4	3019.5	7.9 (7)
Speed-up (%)							2.45				1.34				7.47				8.09	
	DSSR w/o initialization				DSSR wtca 5				DSSR wtca 10				DSSR whca 5				DSSR whca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)
Average (10)	24.2	23.2	3285.1	8.3 (6)	20.6	24.6	3153.1	7.9 (7)	17.4	26.4	3160.4	7.9 (7)	19.9	23.9	3035.4	7.9 (7)	16.1	25.1	2985.7	7.9 (7)
Speed-up (%)							4.02				3.80				7.60				9.11	

Table 6
Cordeau's instances—DSSR MO-All

	DSSR w/o initialization				DSSR tca 5				DSSR tca 10				DSSR hca 5				DSSR hca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)
Average (10)	6.5	30.6	2542.0	4.6 (8)	6.3	28.7	2544.3	5.8 (7)	5.9	31.8	2615.9	4.0 (8)	6.4	28.0	2491.9	5.8 (7)	5.6	29.0	2336.3	4.0 (8)
Speed-up (%)							−0.09				−2.90				1.97				8.09	
	DSSR w/o initialization				DSSR wtca 5				DSSR wtca 10				DSSR whca 5				DSSR whca 10			
	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)	It.	Θ	Time (s)	(%)
Average (10)	6.5	30.6	2542.0	4.6 (8)	6.0	29.7	2519.1	4.0 (7)	6.0	31.1	2605.4	5.0 (7)	6.3	28.1	2579.4	4.7 (7)	6.0	28.3	2518.9	4.2 (7)
Speed-up (%)							0.90				−2.49				−1.47				0.91	

Table 7
Solomon's instances—100 vertices

	DSSR HMO w/o initialization				MS_{10}					MS_{20}				
	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)
Average	7.59	6.59	1107.31	0.66 (27)	4.21	4.55	7.72	425.96	0.0 (29)	9.38	3.10	11.48	671.72	0.0 (29)
Speed-up (%)								61.53					39.33	
	DSSR MO-All w/o initialization				MS_{10}					MS_{20}				
	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)
Average	5.00	8.37	1127.73	0.17 (27)	4.21	3.28	8.66	432.81	0.0 (29)	9.38	2.72	12.14	708.57	0.0 (29)
Speed-up (%)								61.62					37.17	

Table 8
Cordeau's instances

	DSSR HMO w/o initialization				MS_{10}					MS_{20}				
	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)
Average	24.17	23.17	3285.11	8.25 (8)	3	21	23.7	3112.60	6.36 (7)	8	16	24	3135.00	9.73 (6)
Speed-up (%)								5.25					4.57	
	DSSR MO-All w/o initialization				MS_{10}					MS_{20}				
	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)	Init	It.	Θ	Time (s)	(%)
Average	6.50	30.63	2542.03	4.61 (8)	3	6.6	28.6	2517.07	5.76 (7)	8	5.4	25	2379.52	4.04 (8)
Speed-up (%)								0.98					6.39	

We also observed that in some difficult cases (most Solomon's rc instances and Cordeau's instances) the number of final critical vertices considered by MO-All algorithm with initialization ($m = 10$) was less than without initialization: this means that a smart choice of the initial critical vertices cannot only save computing time in the early iterations of the DSSR algorithm but it may also have beneficial effects on the final cardinality of Θ and hence on the computing time of the last iteration. This is even more remarkable when referred to HMO strategy, which usually produces the smallest critical sets as shown in Tables 1 and 2. We observed that in some few cases the number of critical vertices at the end of the HMO algorithm was smaller when the critical set had been initialized.

The initialization with our mixed strategy also yielded a substantial speed-up: for Solomon's data-set the reduction of the average computing time of the DSSR MO-All algorithm was 61.62% with MS_{10} and 37.17% with MS_{20} and the reduction of the average computing time for the DSSR HMO algorithm was 61.53% with MS_{10} and 39.33% with MS_{20}. The reported speed-up has been computed considering the time limit of 2 h for the unsolved instances. Therefore it is a lower bound on the real speed-up for those instances. For Cordeau's data-set the initialization of the critical vertex set produced a smaller average speed-up: 5.25% with DSSR HMO algorithm and $m = 10$, and 6.39% with DSSR MO-All algorithm and $m = 20$.

As a rule of thumb coming from our experiments, we could observe that the best results were obtained when the initial cardinality of the critical set was about half the expected number of vertices in an optimal solution. With the mixed strategy this results is obtained, in average, by setting the parameter m to the expected number of vertices in an optimal tour.

6. Conclusions

We have presented an exact optimization algorithm for the OPTW based on DSSR and we have compared three different strategies proposed by Boland et al. to iteratively increase the size of the critical vertex set. The outcome of our experiments is that there is no domination between the different strategies. In addition, we have proposed a new heuristic technique for the initialization of the critical vertex set, showing that it significantly reduces the number of iterations and the amount of computing time needed by the DSSR algorithm to converge to an optimal solution.

Acknowledgments

We acknowledge the useful comments of two anonymous referees. This work has been partially supported by the Italian Ministry for University and Research—project PRIN 2005 "Routing and packing problems in the optimization of transportation systems".

References

[1] Boland N, Dethridge J, Dumitrescu I. Accelerated label setting algorithms for the elementary resource constrained shortest path. Operations Research Letters 2006;34:58–68.
[2] Righini G, Salani M. New dynamic programming algorithms for the resource constrained elementary shortest path. Networks, in press, doi:10.1002/net.20212.
[3] Laporte G. The Traveling Salesman Problem: an overview of exact and approximate algorithms. European Journal of Operational Research 1992;59:231–47.
[4] Feillet D, Dejax P, Gendreau M. Traveling Salesman Problems with profits: an overview. Transportation Science 2005;39:188–205.
[5] Tsiligrides T. Heuristic methods applied to orienteering. Journal of the Operational Research Society 1984;35:797–809.
[6] Golden BL, Levy L, Vohra R. The Orienteering Problem. Naval Research Logistics 1987;34:307–18.
[7] Ramesh R, Yong Seok Y, Karwan MH. An optimal algorithm for the orienteering tour problem. ORSA Journal on Computing 1992;4: 155–65.
[8] Fischetti M, Salazar JJ, Toth P. Solving the Orienteering Problem through branch-and-cut. INFORMS Journal on Computing 1998;10: 133–48.
[9] Gendreau M, Laporte G, Semet F. A branch-and-cut algorithm for the undirected selective traveling salesman problem. Networks 1998;32: 263–73.
[10] Kantor MG, Rosenwein MB. The orienteering problem with time windows. Journal of the Operational Research Society 1992;43:629–35.
[11] Bar-Yehuda R, Even G, Shahar S. On approximating a geometric prize-collecting traveling salesman problem with time windows. Lecture notes in computer science, vol. 2832, Berlin, Springer, 2003, p. 55–66.
[12] Boussier S, Feillet D, Gendreau M. An exact algorithm for the team orienteering problem. 4OR 2007;5:211–30.

[13] Cordeau JF, Gendreau M, Laporte G. A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 1997;30: 105–19.
[14] Feillet D, Dejax P, Gendreau M, Gueguen C. An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems. Networks 2004;44:216–29.
[15] Irnich S, Desaulniers G. Shortest path problems with resource constraints. In: Desaulniers G, Desrosiers J, Solomon MM, editors. Column generation. US: Springer; 2005. p. 33–65.
[16] Beasley JE, Christofides N. An algorithm for the resource constrained shortest path problem. Networks 1989;19:379–94.
[17] Dumitrescu I, Boland N. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. Networks 2003;42:135–53.
[18] Righini G, Salani M. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optimization 2006;3:255–73.
[19] Christofides N, Mingozzi A, Toth P. State-space relaxation procedures for the computation of bounds to routing problems. Networks 1981;11: 145–64.