Dynamic programming for the orienteering problem with time windows

Giovanni Righini, Matteo Salani

Dipartimento di Tecnologie dell'Informazione

Università degli Studi di Milano, Via Bramante 65, 26013 Crema (CR), Italy

fax.: +39 0373 898010, e-mail: {righini,salani}@dti.unimi.it

March 29, 2006

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 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. [3] and by Righini and Salani [19]. These algorithms exploit a new technique, called *decremental state space relaxation* [19] or *state space augmentation* [3]. The purpose of this paper is twofold: first, to apply dynamic programming with decremental state space relaxation (DSSR) to solve the OPTW; second, to present some new ideas concerning the heuristic initialization of the critical vertex set in DSSR algorithms.

Literature review. The Traveling Salesman Problem (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 [16]. Several of these extensions belong to the class of TSP with Profits (see Feillet et al. [7]), where a profit is associated with each vertex and the overall collected profit has to be maximized. 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; in the OP the tour starts from and returns to a specified depot. The OP was introduced by Tsiligrides [20] and surveyed by Golden et al. [12]. A first exact algorithm for the OP was proposed by Ramesh et al. [17]. More recently Fischetti et al. [9] 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. [10] for the OP with compulsory vertices.

In this paper we consider the Orienteering Problem with Time Windows (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 [15] presented a so-called "tree heuristic", which only (approximately) solved small instances; more recently Bar-Yehuda et al. [1] 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 OPTW can be reformulated as a Resource Constrained Elementary Shortest Path Problem (RCESPP), which in turn can be effectively solved by dynamic programming with decremental state space relaxation (DSSR), a method presented in Righini and Salani [19]. DSSR consists in solving through dynamic programming 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. [3], who also compared different strategies to define new critical vertices at each iteration.

Original contributions. In this paper we present an exact optimization algorithm for the OPTW based on dynamic programming 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. [5]. 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.

Paper outline. In Section 2 we give a formal statement of the problem; in Section 3 we present a dynamic programming 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 Orienteering Problem with Time Windows (OPTW) is defined as follows. We are given a complete undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, with a positive weight t_{ij} associated with each edge, representing the travel time between vertex i and vertex j. For each vertex $i \in \mathcal{N}$ 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 $N = |\mathcal{N}|$, 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 $T = \max_{i \in \mathcal{N} \setminus \{1,N\}} \{b_i + s_i + t_{iN}\}$.

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

3 Dynamic programming for the OPTW

In dynamic programming algorithms for computing optimal paths, a *state* associated with vertex i represents a path starting at vertex 1 and ending at vertex i. The dynamic programming 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. [8], Irnich and Desaulniers [14] and Boland et al. [3].

Applying the same idea to the OPTW, one can design a basic dynamic programming algorithm (DP), 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 Resource Constrained Elementary Shortest Path Problem (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 [2] to enforce the elementary path constraint.

The RCESPP has recently been attacked by new dynamic programming algorithms developed by Dumitrescu and Boland [6] and Righini and Salani [18]. Hereafter we apply to the OPTW the bi-directional and bounded dynamic programming method presented in [18], which we briefly recall to make this paper self-consistent. In bi-directional dynamic programming 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, 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. 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

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

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 $[a_i^{bw}, b_i^{bw}]$ 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

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

and for backward extensions we have

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

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

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

$$\begin{cases} S_1 \leq S_2 \\ \tau_1 \leq \tau_2 \\ P_1 \geq P_2 \end{cases}$$

and at least one of the inequalities is strict.

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 $(S^{fw}, \tau^{fw}, P^{fw}, i)$ with a backward path $(S^{bw}, \tau^{bw}, P^{bw}, j)$ the feasibility conditions are the following:

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

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

4 Decremental State Space Relaxation

State space relaxation (SSR) was introduced by Christofides et al. [4] to reduce the number of states to be explored; with SSR the search space explored by dynamic programming 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 $\sigma = \sum_{i \in \mathcal{N}} S_i$, 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:

$$\begin{cases} \sigma_1 \leq \sigma_2 \\ \tau_1 \leq \tau_2 \\ P_1 \geq P_2. \end{cases}$$

In Righini and Salani [19] the authors introduced decremental state space relaxation (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. [3] 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 dynamic programming 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 dynamic programming 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 dynamic programming must consider and requires additional memory space and computing time.

In their paper [3] 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 [19];
- 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 [3] and we compare the first three strategies above.

Data-sets. We tested our algorithms on three classes of instances obtained from the well-known Solomon's data-set of VRPTW instances and from instances proposed by Cordeau et al. [5] for the Multi Depot Periodic Vehicle Routing Problem (MDPVRP). The first two sets, composed by 29 instances, have been made by considering the first 50 and 100 vertices of Solomon's instances. Depending on the displacement of the customers, these data-sets are divided into *random*, *clustered* and *random-clustered* categories. Instances belonging to the same data-set have the customers located in the same way and with the same demands; the instances differ only for the time windows. The third set has been derived from Cordeau's 20 instances data-set, considering all customers active in the same day. We consider the delivery demand associated with vertex i in the original data-set as the prize p_i for that vertex.

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 two hours.

Experimental results. Tables 1, 2 and 3 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 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 mean that the corresponding instances were not solved within the time limit. The last two rows report the average values and 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.

All instances with 50 vertices were solved by the DSSR algorithms while the basic DP algorithm failed to solve 4 instances out of 29. 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 two hours, but for the unsolved instances DSSR HMO-All and DSSR MO-All obtained the smallest gaps. Cordeau's instances are harder to solve: all algorithms failed to provide reasonable solutions for instances $pr11$ to $pr20$ which are not reported here: after two hours of computing time we observed percentage gaps greater than 50%. For the other instances DSSR MO-All dominated the other DSSR strategies with the exception of instance $pr06$. For smaller and tightly constrained instances the basic DP algorithm was sometimes faster than the DSSR algorithms, but it failed to solve 4 out of 10 considered instances.

In their experiments, made on different randomly generated data-sets, Boland et al. [3] 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 Cordeau's instances. The comparison based on the number of iterations is favorable to DSSR MO-All, while the comparison based on the final number of critical vertices is favorable to DSSR HMO. We observe that DSSR HMO-All was almost always dominated either by DSSR MO-All or by DSSR HMO.

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 both DSSR HMO and DSSR MO-All 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 $\sum_{j \in \mathcal{N} \setminus \{i\}} f_{ij}(b_i - a_i)$.

In general these 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 into it unnecessary vertices (low m).

Experimental results. Tables 4 to 7 report on the computational results obtained by MS_m on Solomon's instances with 100 vertices (Tables 4 and 5) and Cordeau's instances (Tables 6 and 7) with strategies HMO (Tables 4 and 6) and MO-All (Tables 5 and 7). From the examination of the number of vertices in optimal solutions (reported in Tables 2 and 3) we chose $m = 10$ and $m = 20$. The format of Tables 4-7 is the same of Tables 1-3 but an additional column reports on the number of vertices inserted into the critical vertex set in the initialization phase. The last row reports also the percentage speed-up on the computing time. We performed our experiments using the MO-All and HMO algorithms because in the previous tests HMO-All was dominated by DSSR HMO or DSSR MO-All in most cases.

The critical vertex set initialization definitely improved the performance of both HMO and MO-All algorithms. Remarkably the nasty instances R104_100 and R108_100 were solved within the time limit. Both algorithms needed fewer iterations to converge to an elementary solution.

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 can not 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 1-3. We observed that in some cases (namely instances c103, pr03 and pr04) the number of critical vertices at the end of the HMO algorithm was smaller when the set had been initialized.

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

References

[1] R. Bar-Yehuda, G. Even, S. Shahar, "On approximating a geometric prize-collecting traveling salesman problem with time windows", *Lecture Notes in Computer Science* 2832 (2003), 55-66.

[2] J. E. Beasley, N. Christofides, "An algorithm for the resource constrained shortest path problem", *Networks* 19 (1989), 379-394.

[3] N. Boland, J. Dethridge, I. Dumitrescu, "Accelerated label setting algorithms for the elementary resource constrained shortest path", *Operations Research Letters* 34 (2006) 58-68.

[4] N. Christofides, A. Mingozzi, P. Toth, "State-space relaxation procedures for the computation of bounds to routing problems", *Networks* 11 (1981), 145-164.

[5] J.F. Cordeau, M. Gendreau, G. Laporte, "A tabu search heuristic for periodic and multi-depot", *Networks* 30 (1997), 105119.

[6] I. Dumitrescu, N. Boland, "Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem", *Networks* 42 (2003), 135-153.

[7] D. Feillet, P. Dejax, M. Gendreau, "Traveling Salesman Problems with profits: an overview", *Transportation Science* 39 (2005), 188-205.

[8] D. Feillet, P. Dejax, M: Gendreau, C. Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraints: application to some vehicle routing problems", *Networks* 44 (2004), 216-229.

[9] M. Fischetti, J.J. Salazar, P. Toth, "Solving the Orienteering Problem through Branch-and-Cut", *INFORMS Journal on Computing* 10 (1998), 133-148.

[10] M. Gendreau, G. Laporte, F. Semet, "A Branch-and-Cut algorithm for the undirected selective traveling salesman problem", *Networks* 32 (1998), 263-273.

[11] M. Gendreau, G. Laporte, F. Semet, "A tabu search heuristic for the undirected selective traveling salesman problem", *European Journal of Operational Research* 106 (1998), 539-545.

[12] B.L. Golden, L. Levy, R. Vohra, "The Orienteering Problem", *Naval Research Logistics* 34 (1987), 307-318.

[13] G. Gutin, A. Punnen, "The Traveling Salesman Problem and its Variations", Kluwer Academic Publishers, 2002, Boston

[14] S. Irnich, G. Desaulniers, "Shortest path problems with resource constraints", *Cahier du GERAD G-2004-11*, Université de Montréal, 2004

[15] M.G. Kantor, M.B. Rosenwein, "The Orienteering Problem with Time Windows", *Journal of the Operational Research Society* 43 (1992), 629-635.

[16] G. Laporte, "The Traveling Salesman Problem: An overview of exact and approximate algorithms", *European Journal of Operational Research* 59 (1992), 231-247.

[17] R. Ramesh, Y. Yong Seok, M.H. Karwan, "An optimal algorithm for the orienteering tour problem", *ORSA Journal on Computing* 4 (1992), 155-165.

[18] G. Righini, M. Salani, "Dynamic programming algorithms for the elementary shortest path problem with resource constraints", *Electronic Notes in Discrete Mathematics* 17 (2004), 247-249.

[19] G. Righini, M. Salani, "New dynamic programming algorithms for the Resource Constrained Elementary Shortest Path", *Note del Polo - Ricerca* 69 (2005), Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, submitted for publication.

[20] T. Tsiligrides, "Heuristic methods applied to orienteering", *Journal of the Operational Research Society* 35 (1984), 797-809.

Table 1: Solomon's instances - 50 vertices

Instance	Optimum		DSSR HMO				DSSR HMO-All				DSSR MO-All				Basic D.P.
Name	Prize	Vert.	It	Θ	Time(s)	(%)	It	Θ	Time(s)	(%)	It	Θ	Time(s)	(%)	Time(s)
c101_50	270	10	**1**	**0**	0.02	-	**1**	**0**	**0.01**	-	**1**	**0**	**0.01**	-	0.02
c102_50	300	11	5	**4**	1.08	-	5	**4**	1.11	-	**4**	**4**	**0.91**	-	60.30
c103_50	320	11	6	**5**	23.02	-	6	**5**	23.27	-	**4**	6	**22.96**	-	-
c104_50	340	11	6	**5**	81.41	-	**4**	**5**	**46.25**	-	5	8	111.16	-	-
c105_50	300	11	**1**	**0**	**0.02**	-	**1**	**0**	0.03	-	**1**	**0**	0.02	-	0.04
c106_50	280	10	**1**	**0**	**0.02**	-	**1**	**0**	0.02	-	**1**	**0**	0.02	-	0.03
c107_50	310	11	**1**	**0**	0.04	-	**1**	**0**	**0.03**	-	**1**	**0**	0.04	-	0.06
c108_50	320	11	4	3	0.22	-	**2**	**2**	0.11	-	**2**	**2**	0.11	-	**0.10**
c109_50	340	11	7	**6**	0.92	-	**3**	**6**	0.41	-	**3**	**6**	**0.40**	-	0.89
r101_50	126	5	**1**	**0**	0.01	-	**1**	**0**	**0.00**	-	**1**	**0**	**0.00**	-	**0.00**
r102_50	198	9	**5**	**4**	**0.72**	-	**5**	5	1.29	-	**5**	6	1.42	-	3.41
r103_50	214	10	8	**7**	**29.96**	-	8	7	30.07	-	**6**	9	56.38	-	186.37
r104_50	227	10	10	**9**	**152.69**	-	9	10	161.10	-	**6**	12	272.78	-	-
r105_50	159	6	**1**	**0**	**0.01**	-	**1**	**0**	**0.01**	-	**1**	**0**	**0.01**	-	0.02
r106_50	208	10	**4**	**3**	**0.56**	-	**4**	4	0.85	-	**4**	5	0.88	-	4.78
r107_50	220	10	6	**5**	**9.82**	-	6	5	9.70	-	**5**	9	43.78	-	182.21
r108_50	227	10	12	**11**	**410.58**	-	11	13	492.29	-	**7**	14	950.23	-	-
r109_50	192	8	**5**	**4**	0.20	-	**5**	**4**	0.19	-	**5**	5	0.20	-	**0.07**
r110_50	208	9	9	**8**	1.56	-	7	10	1.26	-	**6**	**8**	1.09	-	**0.73**
r111_50	223	9	4	**3**	1.78	-	4	3	1.84	-	**3**	**3**	**1.51**	-	24.75
r112_50	226	10	5	**4**	3.11	-	**4**	**4**	**2.78**	-	**4**	5	2.97	-	42.62
rc101_50	180	7	5	**4**	0.04	-	**4**	**4**	0.03	-	**4**	**4**	0.04	-	**0.01**
rc102_50	230	9	10	**9**	0.69	-	9	10	0.60	-	**6**	11	0.46	-	**0.18**
rc103_50	240	9	12	**11**	2.24	-	10	12	2.13	-	**8**	16	**1.96**	-	2.57
rc104_50	270	10	11	**10**	6.13	-	9	11	**5.18**	-	**7**	15	6.19	-	75.47
rc105_50	210	9	13	**12**	0.71	-	12	14	0.68	-	**9**	15	0.54	-	**0.11**
rc106_50	210	8	19	**18**	0.77	-	11	23	0.46	-	**10**	23	0.45	-	**0.07**
rc107_50	240	10	15	**14**	3.41	-	13	14	3.01	-	**9**	20	3.11	-	**2.35**
rc108_50	250	9	15	**14**	**9.29**	-	15	16	10.81	-	**9**	20	15.60	-	39.99
Average	-	-	6.97	**5.97**	**25.55**	-	5.93	6.59	27.43	-	**4.72**	7.79	51.56	-	1263.00
Solved	-	-	29	-	-	-	29	-	-	-	29	-	-	-	25

Table 2: Solomons's instances - 100 vertices

Instance	Optimum		DSSR HMO				DSSR HMO-All				DSSR MO-All				Basic D.P.
Name	Prize	Vert.	It	Θ	Time(s)	(%)	It	Θ	Time(s)	(%)	It	Θ	Time(s)	(%)	Time(s)
c101_100	320	10	**1**	**0**	0.07	-	**1**	**0**	**0.06**	-	**1**	**0**	**0.06**	-	0.14
c102_100	360	11	4	**3**	5.33	-	**3**	**3**	**3.81**	-	**3**	**3**	4.49	-	-
c103_100	400	11	9	**8**	**1081.04**	-	8	9	1393.38	-	**6**	9	1101.74	-	-
c104_100	420	11	9	**8**	2141.38	-	7	**8**	**1856.39**	-	**6**	9	2166.79	-	-
c105_100	340	10	**1**	**0**	**0.12**	-	**1**	**0**	0.13	-	**1**	**0**	**0.12**	-	0.30
c106_100	340	10	**1**	**0**	**0.14**	-	**1**	**0**	0.15	-	**1**	**0**	0.15	-	0.39
c107_100	370	11	**1**	**0**	**0.20**	-	**1**	**0**	0.20	-	**1**	**0**	0.20	-	0.51
c108_100	370	11	**4**	**3**	1.43	-	**4**	4	1.47	-	**4**	4	1.46	-	**0.93**
c109_100	380	11	12	**11**	14.01	-	8	13	10.65	-	**8**	13	**10.57**	-	11.67
r101_100	198	9	**1**	**0**	0.04	-	**1**	**0**	**0.03**	-	**1**	**0**	0.04	-	0.08
r102_100	286	11	7	**6**	**233.20**	-	7	7	260.04	-	**5**	8	310.79	-	-
r103_100	293	11	10	**9**	**5498.81**	-	9	8	-	0.30	**7**	10	5729.01	-	-
r104_100	(303)	(12)	10	9	-	**2.50**	9	9	-	**2.50**	6	12	-	**2.50**	-
r105_100	247	11	3	**2**	0.35	-	**2**	**3**	**0.23**	-	**2**	**3**	**0.23**	-	0.24
r106_100	293	11	8	**7**	579.44	-	8	8	634.89	-	**5**	8	**334.49**	-	-
r107_100	299	13	9	**8**	**2979.94**	-	9	10	3483.73	-	**6**	11	3514.80	-	-
r108_100	(303)	(12)	9	**8**	-	16.78	9	9	-	**2.50**	**6**	12	-	**2.50**	-
r109_100	277	12	11	10	6.87	-	6	9	3.63	-	**5**	**9**	3.09	-	**1.67**
r110_100	284	13	10	**9**	78.52	-	7	10	72.27	-	**4**	**9**	**30.83**	-	79.05
r111_100	297	12	11	**10**	1932.55	-	9	10	1807.86	-	**7**	12	**1408.80**	-	-
r112_100	298	12	12	**11**	2624.42	-	9	**11**	**2508.17**	-	**7**	14	3177.02	-	-
rc101_100	219	9	4	**3**	0.31	-	**3**	**3**	0.24	-	**3**	**3**	0.23	-	**0.14**
rc102_100	266	10	8	**7**	**6.11**	-	8	9	8.68	-	**6**	11	9.88	-	12.75
rc103_100	266	10	14	**13**	**88.12**	-	12	14	99.25	-	**9**	18	111.44	-	401.34
rc104_100	301	11	12	**11**	304.42	-	11	11	268.63	-	**7**	16	**264.84**	-	-
rc105_100	244	11	8	**7**	2.86	-	8	9	3.08	-	**7**	10	2.95	-	**1.93**
rc106_100	252	11	11	**10**	3.64	-	7	10	2.24	-	**6**	13	2.08	-	**0.90**
rc107_100	277	10	14	**13**	50.76	-	14	13	50.82	-	**10**	19	**49.19**	-	59.16
rc108_100	298	11	10	**9**	77.77	-	9	10	71.10	-	**7**	14	**68.95**	-	959.45
Average	-	-	7.59	**6.59**	**1107.31**	0.66	6.31	7.08	1177.28	0.18	**5.00**	8.37	1127.73	**0.17**	3032.05
Solved	-	-	**27**	-	-	-	26	-	-	-	**27**	-	-	-	17

Table 3: Cordeau's instances

Instance	Optimum		DSSR HMO				DSSR HMO-All				DSSR MO-All				Basic D.P.
Name	Prize	Vert.	It	Θ	Time(s)	(%)	It	Θ	Time(s)	(%)	It	Θ	Time(s)	(%)	Time(s)
pr01_48	308	21	13	**12**	3.79	-	10	12	2.93	-	**4**	15	1.19	-	**0.70**
pr02_96	404	24	22	**21**	101.78	-	13	23	68.75	-	**6**	24	37.52	-	**30.70**
pr03_144	394	22	26	**25**	442.45	-	19	27	318.79	-	**8**	38	**151.73**	-	265.72
pr04_192	489	24	38	**37**	3152.74	-	20	39	1639.49	-	**7**	40	**648.82**	-	1084.80
pr05_240	595	31	23	22	-	12.2	16	25	-	9.3	**7**	**40**	**6815.82**	-	-
pr06_288	(501)	(26)	13	12	-	**29.7**	7	8	-	47.9	4	19	-	45.1	-
pr07_72	298	17	21	**20**	12.13	-	12	22	6.85	-	**6**	21	3.65	-	**1.51**
pr08_144	463	25	25	**24**	338.25	-	11	25	131.94	-	**6**	31	**90.71**	-	128.60
pr09_216	493	29	27	26	-	3.3	17	**26**	3988.45	-	**8**	36	**3270.88**	-	-
pr10_288	(584)	(32)	24	23	-	37.3	21	38	-	11.0	12	62	-	**1.0**	-
Average	-	-	24.17	**23.17**	3285.11	8.25	14.57	24.86	2775.72	6.82	**6.50**	30.63	**2542.03**	**4.61**	3031.20
Solved	-	-	6	-	-	-	7	-	-	-	**8**	-	-	-	6

Table 4: Solomon's instances - 100 vertices - DSSR HMO

Instance	Optimum		DSSR w/o initialization				MS_{10}					MS_{20}				
Name	Best	N	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)
c101_100	320	10	**1**	**0**	**0.07**	-	3	**1**	3	0.08	-	10	**1**	10	0.09	-
c102_100	360	11	4	**3**	5.33	-	2	3	**3**	4.08	-	3	**1**	**3**	**2.26**	-
c103_100	400	11	9	8	1081.04	-	5	3	**7**	**270.74**	-	10	**1**	10	552.99	-
c104_100	420	11	9	**8**	2141.38	-	6	4	9	**1627.66**	-	15	**1**	15	6401.22	-
c105_100	340	10	**1**	**0**	0.12	-	0	**1**	**0**	**0.11**	-	5	**1**	5	0.14	-
c106_100	340	10	**1**	**0**	**0.14**	-	1	**1**	1	0.16	-	6	**1**	6	0.19	-
c107_100	370	11	**1**	**0**	**0.20**	-	1	**1**	1	0.25	-	7	**1**	7	0.31	-
c108_100	370	11	**4**	**3**	**1.43**	-	1	5	5	2.01	-	5	4	8	2.54	-
c109_100	380	11	12	11	14.01	-	4	8	**11**	**6.98**	-	10	**7**	16	7.71	-
r101_100	198	9	**1**	**0**	**0.04**	-	4	**1**	4	0.08	-	12	**1**	12	0.16	-
r102_100	286	11	7	**6**	233.20	-	7	2	8	59.90	-	9	**1**	9	**42.08**	-
r103_100	293	11	10	**9**	5498.81	-	8	3	10	**672.06**	-	12	**2**	13	676.55	-
r104_100	303	13	-	-	-	2.50	7	5	**11**	**4725.23**	-	14	**3**	16	5613.17	-
r105_100	247	11	3	**2**	**0.35**	-	3	**2**	4	0.37	-	7	**2**	8	0.44	-
r106_100	293	11	8	**7**	579.44	-	7	2	8	95.51	-	9	**1**	9	**65.05**	-
r107_100	299	13	9	**8**	2979.94	-	8	3	10	**773.32**	-	12	**2**	13	786.09	-
r108_100	308	13	-	-	-	16.78	7	4	**10**	**2353.73**	-	14	**2**	15	3048.19	-
r109_100	277	12	11	**10**	6.87	-	3	9	11	3.87	-	9	**7**	15	**3.38**	-
r110_100	284	13	10	**9**	78.52	-	4	6	9	28.62	-	9	**4**	12	**26.53**	-
r111_100	297	12	11	**10**	1932.55	-	6	6	11	632.71	-	11	**3**	13	**585.34**	-
r112_100	298	12	12	**11**	2624.42	-	6	6	11	**811.31**	-	15	**3**	17	1348.56	-
rc101_100	219	9	4	**3**	**0.31**	-	4	3	6	0.36	-	8	**2**	9	0.46	-
rc102_100	266	10	8	**7**	6.11	-	1	7	**7**	**4.78**	-	8	**5**	12	5.79	-
rc103_100	266	10	14	**13**	88.12	-	3	11	**13**	**69.78**	-	8	**10**	17	103.20	-
rc104_100	301	11	12	**11**	304.42	-	4	8	**11**	**161.19**	-	10	**6**	15	165.64	-
rc105_100	244	11	8	**7**	2.86	-	2	7	8	2.09	-	7	**4**	10	**1.42**	-
rc106_100	252	11	11	**10**	3.64	-	3	8	**10**	2.17	-	9	**5**	13	**1.51**	-
rc107_100	277	10	14	**13**	50.76	-	6	8	**13**	18.21	-	8	**6**	**13**	**14.98**	-
rc108_100	298	11	10	**9**	77.77	-	6	4	**9**	25.52	-	10	**3**	12	**21.09**	-
Average	-	-	7.59	**6.59**	1107.31	0.66	4.21	4.55	7.72	**425.96**	-	9.38	**3.10**	11.48	671.72	-
Solved	-	-	27	-	-	-	**29**	-	-	-	-	**29**	-	-	-	-
Speed up	-	-	-	-	-	-	-	-	-	**61.53%**	-	-	-	-	39.33%	-

Table 5: Solomon's instances - 100 vertices - DSSR MO-All

Instance	Optimum		DSSR w/o initialization				MS_{10}					MS_{20}				
Name	Best	N	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)
c101_100	320	10	**1**	**0**	**0.06**	-	3	**1**	3	0.11	-	10	**1**	10	0.16	-
c102_100	360	11	3	**3**	4.49	-	2	2	**3**	2.99	-	3	**1**	**3**	**2.25**	-
c103_100	400	11	6	**9**	1101.74	-	5	3	**9**	**491.35**	-	10	**1**	10	555.29	-
c104_100	420	11	6	**9**	2166.79	-	6	3	**9**	**1342.58**	-	15	**1**	15	6426.61	-
c105_100	340	10	**1**	**0**	**0.12**	-	0	**1**	**0**	**0.12**	-	5	**1**	5	0.12	-
c106_100	340	10	**1**	**0**	**0.15**	-	1	1	**1**	0.18	-	6	**1**	6	0.21	-
c107_100	370	11	**1**	**0**	**0.20**	-	1	1	**1**	0.23	-	7	**1**	7	0.28	-
c108_100	370	11	4	**4**	**1**.46	-	1	5	6	1.88	-	5	**4**	9	2.04	-
c109_100	380	11	8	**13**	10.57	-	4	6	**13**	**5.83**	-	10	**5**	17	6.45	-
r101_100	198	9	**1**	**0**	**0.04**	-	4	**1**	4	0.05	-	12	**1**	12	0.09	-
r102_100	286	11	5	**8**	310.79	-	7	2	**8**	60.07	-	9	**1**	9	**42.03**	-
r103_100	293	11	7	**10**	5729.01	-	8	**2**	**10**	**434.24**	-	12	2	14	753.40	-
r104_100	303	13	-	-	-	2.50	7	3	**12**	**3646.72**	-	14	**2**	17	4619.16	-
r105_100	247	11	**2**	**3**	**0.23**	-	3	**2**	4	0.27	-	7	**2**	8	0.41	-
r106_100	293	11	5	**8**	334.49	-	7	2	**8**	95.54	-	9	**1**	9	**64.97**	-
r107_100	299	13	6	**11**	3514.80	-	8	3	13	1245.53	-	12	**2**	13	**786.09**	-
r108_100	308	13	-	-	-	2.50	7	3	**12**	**3862.10**	-	14	**2**	17	4847.69	-
r109_100	277	12	**5**	**9**	**3.09**	-	3	**5**	11	3.55	-	9	6	15	5.41	-
r110_100	284	13	4	**9**	30.83	-	4	**3**	**9**	**20.44**	-	9	**3**	12	26.65	-
r111_100	297	12	7	**12**	1408.80	-	6	4	**12**	**440.67**	-	11	**3**	13	585.23	-
r112_100	298	12	7	14	3177.02	-	6	**3**	**12**	**600.34**	-	15	**3**	17	1348.33	-
rc101_100	219	9	3	**3**	**0.23**	-	4	**2**	6	0.24	-	8	**2**	9	0.31	-
rc102_100	266	10	6	11	9.88	-	1	**4**	**8**	**4.03**	-	8	**4**	13	8.46	-
rc103_100	266	10	9	18	111.44	-	3	**6**	**15**	**61.18**	-	8	7	18	123.19	-
rc104_100	301	11	7	16	264.84	-	4	6	**15**	**193.25**	-	10	**5**	19	305.40	-
rc105_100	244	11	7	**10**	2.95	-	2	6	**10**	1.89	-	7	**4**	12	**1.45**	-
rc106_100	252	11	6	13	2.08	-	3	5	**12**	1.39	-	9	**4**	14	**1.26**	-
rc107_100	277	10	10	19	49.19	-	6	6	**15**	14.76	-	8	**5**	**15**	**13.02**	-
rc108_100	298	11	7	14	68.95	-	6	**3**	**10**	**20.06**	-	10	**3**	14	22.43	-
Average	-	-	5.00	**8.37**	1127.73	0.17	4.21	3.28	8.66	**432.81**	-	9.38	**2.72**	12.14	708.57	-
Solved	-	-	27	-	-	-	**29**	-	-	-	-	**29**	-	-	-	-
Spped up	-	-	-	-	-	-	-	-	-	**61.62%**	-	-	-	-	37.17%	-

Table 6: Cordeau's instances - DSSR HMO

Instance	Optimum		DSSR w/o initialization				MS_{10}					MS_{20}				
Name	Best	N	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)
pr01_100	308	21	13	**12**	3.79	-	3	12	14	3.85	-	13	**7**	19	**2.65**	-
pr02_100	404	24	22	**21**	101.78	-	4	18	**21**	79.66	-	9	**14**	22	**72.34**	-
pr03_100	394	22	26	25	442.45	-	4	20	**23**	232.15	-	8	**18**	25	**230.06**	-
pr04_100	489	24	38	37	3152.74	-	2	36	37	2232.35	-	6	**31**	**36**	**2075.09**	-
pr05_100	595	31	23	22	-	**12.20**	3	22	24	-	**12.20**	6	20	25	-	23.50
pr06_100	(501)	(26)	13	12	-	**29.70**	2	12	13	-	**29.70**	5	10	14	-	35.20
pr07_100	298	17	21	**20**	12.13	-	5	16	**20**	7.12	-	12	**10**	21	**5.08**	-
pr08_100	463	25	25	**24**	338.25	-	3	22	**24**	192.57	-	8	**17**	**24**	**164.74**	-
pr09_100	493	29	27	26	-	3.30	3	**25**	**27**	**6778.26**	-	10	21	30	-	1.30
pr10_100	(584)	(32)	24	23	-	37.30	3	24	26	-	**21.70**	6	20	25	-	37.30
Average	-	-	24.17	**23.17**	3285.11	8.25	3	21	23.7	**3112.60**	**6.36**	8	**16**	24	3135.00	9.73
Solved	-	-	6	-	-	-	**7**	-	-	-	-	6	-	-	-	-
Speed up	-	-	-	-	-	-	-	-	-	**5.25%**	-	-	-	-	4.57%	-

Table 7: Cordeau's instances - DSSR MO-All

Instance	Optimum		DSSR w/o initialization				MS_{10}					MS_{20}				
Name	Best	N	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)	Init	It	Θ	Time(s)	(%)
pr01_100	308	21	4	**15**	1.19	-	3	5	19	1.58	-	13	**3**	21	**1.14**	-
pr02_100	404	24	6	24	37.52	-	4	**6**	**23**	**29.47**	-	9	**6**	24	37.08	-
pr03_100	394	22	**8**	38	151.73	-	4	**8**	32	86.19	-	8	**8**	**28**	**78.33**	-
pr04_100	489	24	**7**	40	648.82	-	2	**7**	40	557.23	-	6	**7**	**38**	**506.27**	-
pr05_100	595	31	**7**	40	6815.82	-	3	9	40	-	1.30	6	**7**	**36**	**4468.41**	-
pr06_100	(501)	(26)	4	19	-	45.10	2	2	10	-	55.30	5	4	24	-	**37.30**
pr07_100	298	17	6	**21**	3.65	-	5	6	**21**	**2.66**	-	12	6	25	2.95	-
pr08_100	463	25	**6**	31	90.71	-	3	**6**	30	74.91	-	8	**6**	**26**	**70.95**	-
pr09_100	493	29	**8**	36	3270.88	-	3	**8**	**35**	**2818.71**	-	10	**8**	37	4230.08	-
pr10_100	(584)	(32)	12	62	-	**1.00**	3	12	62	-	**1.00**	6	11	60	-	3.10
Average	-	-	6.50	30.63	2542.03	4.61	3	6.6	28.6	2517.07	5.76	8	**5.4**	**25**	**2379.52**	**4.04**
Solved	-	-	6	-	-	-	7	-	-	-	-	**8**	-	-	-	-
Speed up	-	-	-	-	-	-	-	-	-	0.98%	-	8	-	-	**6.39%**	-