A Dynamic-Pricing Label-Setting Algorithm for Solving the Elementary Resource Constrained Shortest Path Problem

Ignacio Vitale[1] and Rodolfo Dondo[2]

[1] Facultad de Ingeniería Química, U.N.L, Santiago del Estero 2829, 3000 Santa Fe, Argentina
vitalenacho@gmail.com
[2]Instituto de Desarrollo Tecnológico para la Industria Química (U.N.L. - Conicet), Güemes 3450, 3000 Santa Fe, Argentina
rdondo@santafe-conicet.gov.ar

Abstract. The resource constrained elementary shortest-path problem is a problem used for solving vehicle-routing, production-scheduling and crew-scheduling applications. It occurs as a sub-problem used to implicitly generate the set of all feasible columns in a column-generation solution algorithm. In the problem there is a directed graph with a source node and a destination node, and each arc has a cost and a vector of weights specifying its requirements of resources. A minimum-cost source to destination directed path is sought such that the total consumption of resources does not exceed the resources vehicle-capacity. The problem is NP-hard in the strong sense. We propose a new label-setting heuristic algorithm based on the dynamic update of resources and prices vectors according the partial label-path and embed it into a column generation algorithm to solve some testing problems. Later we perform some numerical experiments in order to study its computational efficiency.

Keywords: resource constrained shortest path, label setting, column generation.

1 Introduction

The elementary resource constrained shortest-path problem (RCSPP) is a problem used for solving vehicle-routing, production-scheduling and crew-scheduling applications. It occurs as a sub-problem used to implicitly generate the set of all feasible columns in a column-generation (CG) algorithm and it is the most time consuming problem of such a procedure. The RCSPP is NP-hard in the strong sense for graphs containing negative cost cycles as demonstrated by by Dror [1]. However, the problem remains NP-hard even if the graph is acyclic. Several solution approaches have been developed for solving the RCSPP up to optimality. The main kinds are: (1) label setting algorithms [2, 3, 4]; (2) label correcting algorithms [5, 6, 7]; (3) constraints programming [8]; and (4) methods based on branch-and-bound [9, 10, 11, 12]. Further details on these techniques and its variations (bidirectional labeling, asymmetric bidi-

rectional labeling, decremental state-space relaxation, among others) are reviewed by [13]. Our motivation for studying the RCSPP comes from application of CG methods to industrial engineering. We are specifically interested in the pricing problem applied to (i) log transportation; (ii) production planning and (iii) bus crew scheduling. In spite of these different applications the subproblem-core is quite similar because it can be modelled as a RCSPP in all of them. We propose a pseudo-polynomial complexity solution procedure for the RCSPP, which is based in the dynamic modification of prices and load vectors within the label generation procedure according the partial label-path. Numerical examples are solved for testing the algorithm performance.

2 Problem Statement

Consider a routes-network represented by an undirected graph $G\{I \cup p, A\}$ with $I = \{i_1, i_2, ..., i_n\}$ denoting the set of nodes or customers and p representing a source /sink node called "depot". Nodes and the depot are connected by a set of arcs $A = \{(i, j) / i,j \in I \cup p\}$. Known resource matrixes $\mathbf{W} = [W_1, W_2, \ldots, W_n]$, with $W_i = [w_{i1},\ldots, w_{iw}]$, and a known prices-vector $\Pi = [\pi_1, \pi_2, \ldots \pi_n]$ are associated to the customer set I. Resources w_{iw} must be collected within a time window $[a_i, b_i]$ on each node $i \in I$. The parameters a_i stand for the earliest possible start-time of the service and parameters b_i state the latest possible start-time of the service at any node. Travel-costs $C = \{c_{ij}\}$ and travel times $\Gamma = \{t_{ij}\}$ are given data for any arc $(i,j) \in A$. Moreover, the service time on node i is denoted st_i. For visiting node $i \in I$, the associated price π_i and the resource vector W_i are accumulated. It is assumed that the triangle inequality is satisfied by the travel costs and travel times, i.e. $c_{ik} + c_{kj} \geq c_{ij}$ and $t_{ik} + t_{kj} \geq t_{ij}$. The solution to the RCSPP problem must: (1) Maximize the net profit collected from the selected subset of nodes $I^{opt} \subseteq I$. This profit is defined as the sum of collected prices minus the cumulated cost incurred by traveling arcs to pick them. (2) The route must start and end on the depot p. (3) The selected nodes must be visited once, so an *elementary path* is designed. (4) Collected resources must never exceed the resource vehicle-capacity q_w. (5) The time-length used to collect loads and prices must be shorter than the maximum allowed working time t^{max}. (6)The service at every customer site i must start within the specified time window $[a_i, b_i]$.

3 A labelling algorithm for deterring repetition of nodes

The computational hardness of this problem has inspired researchers to develop different solution methods to handle this NP-hard problem. The most common approach is to solve the non-elementary relaxation of the problem; i.e. routes that visit customers more than once are allowed and later eliminated. While several papers address the problem of eliminating cycles of 2 and 3 nodes, not much work researched the elimination of all cycles. Beasley and Chirtofides [12] proposed a zero-order programming formulation in which forbidding of visit to nodes more than once is mod-

elled via an additional resource for each node and each of these resources has a limit of one. Vitale and Dondo [13] researched alternative MILP formulations for the elementary shortest path problem with time windows and capacity constraints. Kohl [15] further researched the idea of node-resources proposed on [12] and developed a label-setting algorithm for solving the problem. Feillet et al. [16] took the idea proposed by [12] to develop strong dominance criterions. They also suggested having resources associated to a subset of nodes instead of having a resource associated to each node. The drawback of these methods is that dominance criterions become more complex because dominance of one label over another must be checked for each resource (i.e. capacity-resources + node-resources) leading to an exponential complexity. E.g. if a partial path p_1 doesn't contain some node of another path p_2, the dominance rule will always keep path p_1 independently of the value of accumulated non-node resources. This, in turn, leads to very large labels-banks. To avoid a computational burdensome resources dominance-checking, we have to use an implementation offering pseudo-polynomial efficiency. We, therefore, propose a heuristic procedure that consists on dynamically updating resources and price vectors according the label-path. A label is here defined by a vector of $4 + |W| + |I|$ elements. The first on is the "head node", i.e., the last visited node of the label-path; the second stores the label-path reduced cost, the third one the label-path travel-cost, the fourth element stores the path cumulated travel-time (which it can be seen as a constrained resource). The following $|W|$ elements correspond to the cumulated resources collected along the path, and the last $|I|$ elements store the ordinality of visited nodes, in visiting order, as follows:

Last visited node	Path reduced cost	Path travel cost	Path travel time	Collected resources			Visited nodes		
				w_1	…	w_W	i_j	…	-

```
Step 0: Initialization
        TargetLabel = ∅
        for i ∈ I
        L_p = [p 0 0 0 0 pred_p]
        L_i = [p 0 0 0 0 pred_i]
        end
Step 1: Label extention
        while LabelStore ≠ ∅
        CurrentLabel = LabelStore_1;
        i = LabelStore_1,1; R_i = LabelStore_1,2 ; C_i = LabelStore_1,3; T_i = LabelStore_1,4; L_i = LabelStore_1,5; Pred_i = LabelStore_1,Pred
        if i ∈ Pred_i π_i' = 0 else π_i' = π_i end
        if i ∈ Pred_i q_i' = 0 else q_i' = q_i end
        Step 1.a: Label generation
        for l = 2 to size(LabelStore)
           j = LabelStore_l,1; R_j = LabelStore_l,2
        if L_i + q_i' <= Q_v & T_i + st_i + t_ij <= b_j & R_j > R_i + c_ij - π_i'
           R_j = R_i + c_ij - π_i'; C_j = C_i + c_ij; T_j = min(b_j, T_i + t_ij); L_j = L_i + q_i'; Pred_j = i ∪ Pred_i
           NewLabel = [j R_j C_j T_j L_j Pred_j]
        if NewLabel ∩ LabelStore = ∅
           LabelStore = NewLabel ∪ LabelStore
        Step 1.b: Return to the depot generation
        if R_j + c_jt < 0
           R_p = R_p + c_jp; C_p = C_j + c_jp; T_p = T_j + t_jp; L_p = L_j; Pred_p = j ∪ Pred_j
        if R_p < TargetLabel_1,2 then
           NewTargetLabel = [p R_p C_p T_p L_p Pred_p]
        if TargetLabel ∩ NewTargetLabel = ∅
           TargetLabel = NewTargetLabel ∪ TargetLabel
        end
        end
        end
        end
        end
        Step 1.c: Deletion of repeated labels
        for l = 2 to size(LabelStore)
        if LabelStore_l,1 = NewLabel_1 & LabelStore_l,2 >= NewLabel_2 & LabelStore_l,4 >= NewLabel_4 & LabelStore_l,5 >= NewLabel_5
          delete LabelStore_l;
        end
        end
        end
        end
```

Figure 1: Pseudo-code of the dynamic pricing labeling algorithm

The proposed dynamic pricing label setting algorithm is summarized by the pseudo-code showed in Fig. 1 in which the cumulated resources vector is denoted by L_i. We embedded this heuristic algorithm into a CG algorithm (i.e., we just solve the root node of a branch-and-price tree) for generating columns as longer as it is able to provide new negative reduced-cost columns and; whenever this is no longer possible, we use a MILP formulation to produce the remaining columns. If this formulation also fails to find a new column, the CG procedure ends. The master problem is a linear set partitioning problem and constraints duals taken from its resolution are passed to both pricing methods. The procedure is sketched by the flow chart depicted by Fig. 2.

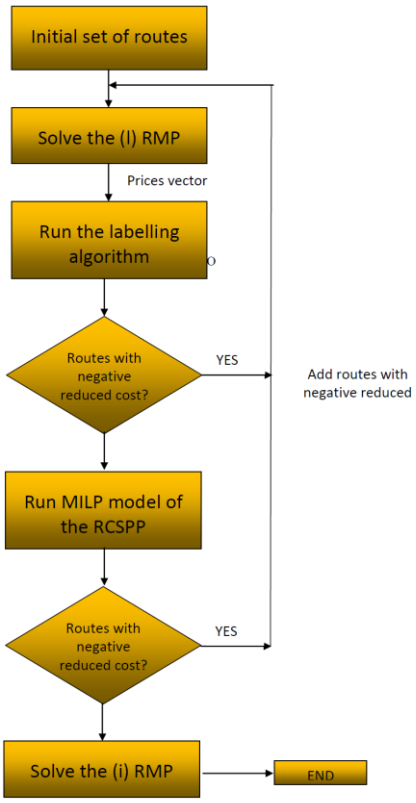


Figure 2: The CG algorithm

4 An illustrative example

As above stated the RCSPP occurs as a sub-problem used to implicitly generate the set of all feasible routes and schedules in the CG reformulation of many routing and production scheduling problems. We embedded the above labelling algorithm and a MILP formulation of the RCSPP [14] into a CG procedure (See Figure 2) in order to solve some testing instances. The algorithm was developed and coded in MatLab us-

ing GAMS [17] just to compute the linear and integer RMPs and to solve the MILP formulations of the pricing problem. Communication between both languages was performed by using the GDXMRW facility developed by [18].

We first illustrate the working logic of the algorithm by solving a toy vehicle routing problem of just 4 nodes, $I = \{i_1, i_2, i_3, i_4\}$ located in the Euclidean plane by their respective coordinates X = [41, 35, 55, 55] and Y = [49, 17, 45, 20]. Depot coordinates are $x_p = 35$ and $y_p = 35$. Also $t^{max} = 150$, the nodes-load vector is W = [10, 42, 13, 19], the vehicle capacity is $q = 50$ units and $ts = 10$. This is a single resource problem and the objective function is the minimization of the travelled distance. The procedure was initialized by the four routes p - i_i – p routes; the problem was solved in just 2 iterations and the solution process is described as follows: the initial master objective had a value $z_{master} = 161.18$ and the prices vector was Π_0 = [30.46, 36.00, 44.70, 50.00]. By using this vector, the slave labelling algorithm generated the columns presented in Table 1.

Table 1: Column generated in the first iteration of the CG for the illustrative example

Column	Base	Path	Reduced cost	Travelling cost	Travelling time	Load
1	p	4-3-1-p	-45.39	79.79	109.79	42
2	p	3-1-p	-23.03	62.15	82.15	23

These columns and their associated costs, resources and paths are expressed as labels by the following two vectors:

[0 -45.39 79.79 109.79 42 (4 3 1 0)]
[0 -23.03 62.15 82.15 23 (3 1 0)]

To illustrate the path propagation process, let us describe how "column 1" was created: The labelling algorithm initially extends paths from the depot p (node of ordinality "0") to any customer i_i and computes it loads price and resources vector $W_{1\text{-}0}$ and $\Pi_{1\text{-}0}$. From any customer i_i, paths are extended to another label (headed by node i_j) considering the updated load and price vectors $W_{1\text{-}i}$ and $\Pi_{1\text{-}i}$. Fig. 3.b depicts the partial path $p – i_4$ with its associated vectors $W_{1\text{-}4}$ and $\Pi_{1\text{-}4}$. From the last visited node, the path is further extended to other nodes generating new labels and later dominance criterions are checked. Dominated labels are deleted and the new ones are stored in the LabelStore. See, for example, the extension of the path $p – i_4$ to i_3 in Fig 3.c. Note that prices and loads picked by the vehicle are zeroed in the dynamic vectors W_k and Π_k (k = iteration number), making unprofitable to return to such visited nodes. From any new path stored in LabelStore, a return path to the depot p is computed and, if it is feasible and it has negative reduced cost, is stored in TargetLabels. To avoid an uncontrolled growth of both LabelStore and TargetLabels, repeated labels are deleted from such label-banks. When all labels of the LabelStore have been either processed or deleted by the dominance check, the algorithm ends and return to the (l)RMP the paths from (and to) the depot stored into TargetLabels. So, after ending the labelling algorithm, both columns described in Table 1 were feed to the (l)RMP. Its solution leaded to a solution value $z_{master} = 115.79$ and a price vector Π_1 = [30.46, 36.00, 21.68,

27.64]. This vector was feed again to the labelling algorithm which this time was unable to return any column. Consequently, the MILP formulation was solved to find any potentially missed column. As this formulation was also unable to do it, the CG procedure ended and, as the solution for the RMP was integer, the optimal solution (detailed in Table 2 and depicted in Figure 4), was found.

Table 2: Optimal solution to the illustrative example

Route	Travel cost	Travel time	Load
p-i_4-i_3-i_1-p	79.79	109.79	42
p-i_2-p	36.00	46.00	42

Figure 3: Extension of the path for generating column "1".

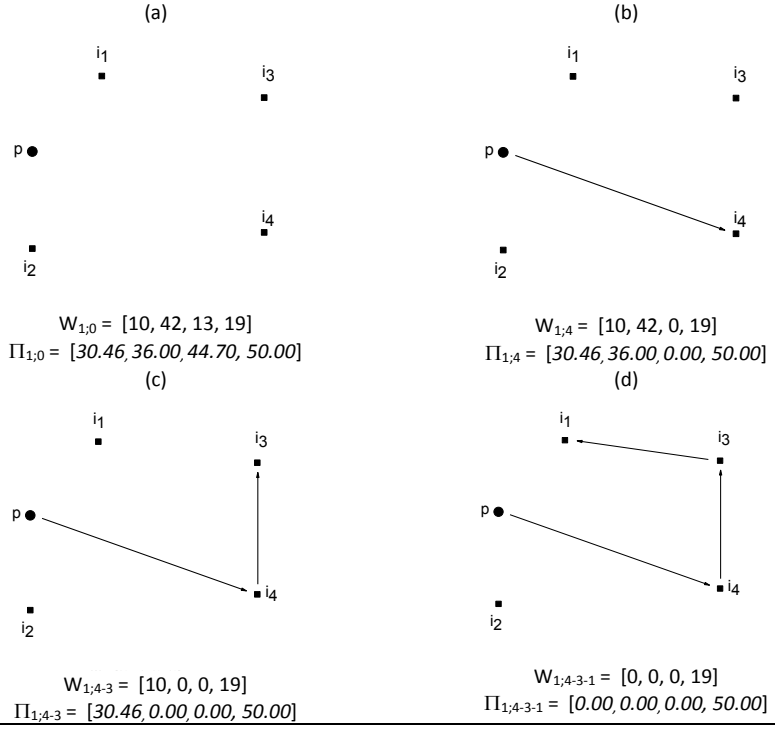
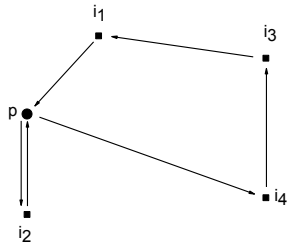


Figure 4: Optimal solution to the illustrative example



5. Results and Discussion

The classical vehicle routing problem with time windows (VRPTW) is considered a testing problem for labelling algorithm to be applied later in more realistic problematics as rich routing and production scheduling problems. Therefore, the proposed algorithm is tested by solving some benchmark VRPTW instances and by comparing its computational performance with a CG algorithm that use three different MILP pricing formulations [14]. The Solomon's testing VRPTW instances [19] have been grouped into C, R and RC categories. C-class problems feature clustered customers. Locations in R-class problems were randomly generated while RC-class problems comprise clustered and randomly located customers. The data set for every category comprises 100 nodes, a depot, similar vehicle capacities but different hard time-window constraints. Euclidean distances among customers and traveling times are numerically identical. Service times are independent of customer requirements and the tour duration cannot exceed a maximum value t^{max}. The objective is the minimization of the total travelled distance. Smaller problems can be generated by selecting the first nodes of each instance. Benchmark problems are further classified into types "1" and "2", like C1 and C2. Type-1 problems have narrow time windows and small vehicle capacities while type-2 problems feature wider time windows and larger vehicle capacities. In order to compare results of the approach that uses the dynamic-pricing labelling procedure with CG algorithms based in "pure" MILP generators we solved all R1-type instances with 25 nodes [13] in a 2.0 GHz 16 GRAM PC. The purpose was to compare times consumed to reach optimality. Our results are summarized in Table 3.

Table 3: Comparison of results for R1 Solomon's instances with 25 nodes.

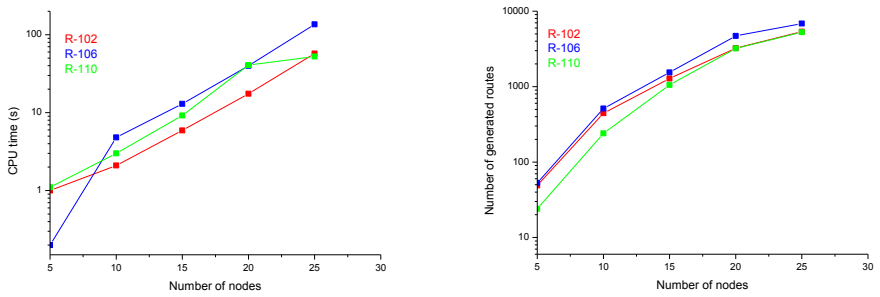
Instance	This approach		CG Formulation 1		CG Formulation 2		CG Formulation 3		Integer solution
	Cols	CPU (s)	Cols	CPU (s)	Cols	CPU (s)	Cols	CPU (s)	
R101	1550	4.7	303	8.9	165	5.1	7.9	163	618.4
R102	5368	42.6	356	40.4	218	10.5	87.5	225	549.7
R103	7815	212.3	457	223.3	327	86.2	225.8	254	455.8
R104	10386	219.2	453	540.6	462	424.9	234.5	383	418.1
R105	2524	7.8	363	16.2	234	43.6	15.4	201	531.7
R106	6846	79.3	401	50.3	310	36.8	34.1	295	466.6
R107	9114	254.6	436	163.3	544	351.1	215.4	348	435.4
R108	10726	351.0	442	311.9	548	553.1	225.3	303	404.4
R109	3845	30.0	709	43.4	536	79.8	104.2	437	442.8
R110	5268	32.4	466	185.4	321	336.6	252.9	289	448.4
R111	8216	249.4	501	127.7	389	180.4	231.6	412	446.2
R112	8857	137.1	429	312.2	542	461.2	201.5	267	409.6
Average	6710	135.4	443	168.6	383	255.8	153.1	298	-

From the above table we can take the following observations: (i) The proposed algorithm faster solved the problem in half of the test instances. (ii) The number of generated columns is an order of magnitude higher than the number produced using just MILP generators. (iii) The average CPU solution-time (135.4) was not massively improved with respect of algorithms based in "pure" MILP generators. In other words,

the much faster solution time per column did not translated in massive improvements of total solution times.

To roughly illustrate the computational complexity associated to the new algorithm, we generated and solved instances by selecting the first 5, 10, 15, 20 and 25 nodes from examples R-102, R-106 and R-110. The information about CPU times and generated labels was recorded and draw as a function of instances sizes and illustrated in Figure 4.

Figure 5: CPU time and number of generated routes as a function of instances sizes



It can be observed that the number of produced columns as a function of the instance size shows a pseudo-polynomial behavior but the CPU solution time is still exponential on the instance size. This means that further research aimed at optimizing the code may be necessary.

5 Conclusions

In this work, we developed a heuristic dynamic-pricing label-setting algorithm for solving the elementary RCSPP and performed some numerical research to test its computational efficiency. The RCSPP is the bottleneck of CG methods designed to solve vehicle routing, crew scheduling and production scheduling problems. The proposed dynamic label-setting algorithm is based on the idea of dynamically updating price and resource vectors according to the partial label-path by zeroing prices and resources values associated to visited nodes. In this way, rather than forbidding revisiting customers, elementary paths are generated by discouraging new visits to already visited nodes. This allows keeping bounded the number of dominance-checks and the labels banks. As this heuristic procedure may miss some profitable routes, we complemented it with a MILP formulation of the problem used to generate the missed routes. The labelling algorithm was first used and then, whenever the CG demands a few but hard to find columns, the CG procedure switches to a MILP formulation. The brief numerical testing presented shows some promising results because we reduced our solution times in half testing instances. Further research aimed at optimizing the code is going on.

References

1. Dror M. Note on the complexity of the shortest path models for column generation in VRPTW. Oper. Res. (1994);42:977–8.
2. Desrochers M. and Soumis, F. A generalized permanent labelling algorithm for the shortest path problems with time windows, INFOR 26 (1988), 193–214.
3. Gallo G. and Pallottino S. Shortest path algorithms, Ann Oper Res 13 (1988), 1–79.
4. Denardo E. and Fox B. Shortest-route methods: reaching, pruning and buckets, Oper Res 27 (1979), 161–186.
5. Di Puglia Pugliese L. and Guerriero F. A computational study of solution approaches for the resource constrained elementary shortest path problem, Ann Oper Res (2012) 201, 131–157.
6. Powell W. and Chen Z. A generalized threshold algorithm for the shortest path problem with time windows. Network design: Connectivity and facilities, P.M. Panalos and D. Du (Editors), American Mathematical Soc., Providence, RI, (1998) 303–318.
7. Glover F., Glover R., and Klingman D. The threshold shortest path algorithm, Networks 14 (1984), 25–36.
8. Rousseau L., Gendreau M. and Pesant G. Using constraint-based operators to solve the vehicle routing problem with time windows. Journal of heuristics. (2002) 8 (1), 43-58.
9. Carlyle W., Royset J., and Wood R. Lagrangian relaxation and enumeration for solving constrained shortest path problems, Networks 52 (2008), 256–270.
10. Muhandiramge R. and Boland N. Simultaneous solution of Lagrangean dual problems interleaved with preprocessing for the weight constrained shortest path problem, Networks 53 (2009), 358–381.
11. Dondo, R. A New MILP Formulation to the Shortest Path Problem with Time Windows and Capacity Constraints. Latin American Applied Research, (2012) 42, 257-265.
12. Beasley J. and Christofides N. An algorithm for the resource constrained shortest path problem, Networks 19 (1989), 379–394.
13. Costa, L.; Contardo, C., Desaulniers, G. Exact Branch-Price-and-Cut Algorithms for Vehicle Routing. . Transportation Science, 2019, 53 (4), 945-985.
14. Vitale, I. and Dondo, R. On Alternative Formulations to the Shortest Path Problem with Time Windows and Capacity Constraints. SIIIO, Simposio Argentino de Informática Industrial e Investigación Operativa (2019).
15. Kohl, N. Exact Methods for Time Constrained and Related Scheduling Problems. Technical Universitty of Dennmark DK-2800 (1995).
16. Feillet, P., Dejax, M., Gendrau, M. and Gueguen, C. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. Networks, (2004) 43 (3) 2016-229.
17. Kalvelagen, E. Columns generation with GAMS (2011). Downloaded from http://amsterdamoptimization.com/pdf/colgen.pdf
18. Dirkse S., Ferris M., and Ramakrishnan J. GDXMRW: Interfacing GAMS and MATLAB (2014).
19. Solomon, M. Algorithms for the Vehicle Routing and Scheduling Problem With Time Window Constraints. Opns. Res. (1987) 35, 254-265.