A Generalized Threshold Algorithm for the Shortest Path Problem with Time Windows

Warren B. Powell
Department of Civil Engineering and Operations Research
Princeton University
Princeton, NJ 08544

Zhi-Long Chen
Department of Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104-6315

Abstract: In this paper, we present a new labeling algorithm for the shortest path problem with time windows (SPPTW). It is generalized from the threshold algorithm for the unconstrained shortest path problem. Our computational experiments show that this generalized threshold algorithm outperforms a label setting algorithm for the SPPTW on a set of randomly generated test problems. The average running time of the new algorithm is about 40% less than the label setting algorithm, which is today the best algorithm based on published experimental evidence.

The shortest path problem with time windows (SPPTW) is a generalization of the classical (unconstrained) shortest path problem (SPP) involving the added complexity of time windows. The SPPTW can be described as follows. Let $G = (V, A)$ be a directed graph where $V = N \cup \{p, q\}$ is the set of nodes with source node p and sink node q, A is the set of arcs. Each node $i \in V$ has a time window $[a_i, b_i]$ within which node i can be visited. Each arc (i, j) has a positive duration t_{ij} which is the time needed to travel from node i to node j, and a positive or negative cost c_{ij}. An arc (i, j) is defined in the set A only if it is feasible, i.e. if it meets the condition: $a_i + t_{ij} \leq b_j$. The objective of the SPPTW is to find the least cost path between the given source p and the given sink q while respecting the time window condition at each visited node.

The shortest path problem with time windows was first introduced in Desrosiers, Pelletier and Soumis [6] and Desrosiers, Soumis and Desrochers [8] as a subproblem for the multiple traveling salesman problem with time windows. Desrochers and Soumis [4] present a label correcting algorithm they call the *generalized permanent labeling algorithm* (GPLA), and show that it outperformed the label correcting algorithm. As a rule, label setting algorithms, such as Dijkstra's algorithm (Dijkstra [10]), have a better worst-case complexity than label correcting algorithms, such as the Bellman-Ford algorithm (Bellman [1]), the D'Esopo-Pape algorithm [14], the threshold algorithm of Glover, Glover and Klingman [13], and the small-label-first algorithm (Bertsekas [2]). By contrast, from the viewpoint of the average-case performance, it is well-known that most of these label correcting algorithms run faster than the label setting algorithm when solving both randomly generated and real-world problems (see e.g. Dial *et al.* , [9], and Bertsekas [2]). Actually, not only in the shortest path problem, but also in many other problems, it is not surprising for one algorithm with higher worst case complexity to outperform another algorithm with lower worst case complexity because the worst case seldom happens in both randomly generated and real-world problems. We show the correctness of this statement once again in this paper. The new algorithm for the SPPTW we propose later in this paper has a higher worst-case complexity but runs faster in general compared to an existing algorithm that is described below.

Desrochers and Soumis [4] propose a labeling setting algorithm that they call the *generalized permanent labeling algorithm* (GPLA). This is generalized from the *best-first* strategy of Dijkstra's algorithm for

the SPP and the concept of *buckets* introduced by Denardo and Fox [3]. The basic procedure of their algorithm is the treatment of a label instead of a node. The algorithm maintains a candidate list L which contains some labels. At each iteration, the lexicographic minimum label of L is computed, then a certain bucket is estimated based on the lexicographic minimum label and the durations and costs of the arcs in the network. All the labels in L that fall in the bucket are selected to be treated next. They show that once a label is treated, it becomes permanent, i.e., it will never be dominated by any label generated later on. Hence the GPLA only treats necessary labels. This is the major advantage of the GPLA. However, in the GPLA, it is time-consuming to compute the lexicographic minimum label of L at each iteration. The worst-case complexity of this algorithm is $O(D^2)$. They report that the GPLA outperforms the one of Desrosiers, Pelletier and Soumis [6] in most cases. Furthermore, the GPLA can solve the problems with up to 2500 nodes and 250,000 arcs in a few minutes.

This paper makes the following contributions: (1) We provide an adaptation of the threshold algorithm of Glover, Glover and Klingman [13] for the shortest path problem with time windows. (2) We show that this algorithm is substantially faster than the label setting algorithm presented in Desrochers and Soumis [4], which also presents work that shows that their label setting algorithm is faster than an earlier label correcting algorithm (see Desrosiers, Pelletier and Soumis [6]). (3) At the same time, we show that the complexity of the new algorithm is worse than the complexity of label setting algorithms. Our finding that the label setting algorithm is faster, but with a poorer worst case complexity, is consistent with most of the published research on shortest path algorithms. However, it is not obvious that this result would hold true for the time-constrained case.

The contribution of this paper, then, is to provide a comparison between a label correcting algorithm and the threshold algorithm for shortest path problems with time windows. At the same time, we do *not* claim to have the fastest algorithm, as this claim depends on a variety of implementation details. For example, Desrosiers *et al.* [7] review a variation of the label correcting algorithm using a pulling process, introduced by Desrochers and Soumis [5]. However, there is no experimental evidence, in the published literature, documenting the relative performance of this method. Our paper strives only to undertake a comparison of a label correcting algorithm with the threshold algorithm. A limitation of any experimental research into the performance

of algorithms is the details of the implementation. In our comparison, the same programmer implemented both algorithms using similar programming techniques.

This paper is organized as follows. In section 1 we introduce some basic concepts that will be used throughout this paper. Then the new algorithm is proposed in Section 2, and computational experiments are presented in Section 3. Finally, we give a concluding remark in Section 4.

1 Problem and Notations

In the labeling algorithms (see, e.g., Gallo and Pallottino [12]) for the unconstrained shortest path problem (SPP), exactly one label is associated with one node. The label is simply the distance of the current path from the source to this node. In the case of the SPPTW, a set of two dimensional labels have to be associated with one node since there may exist many efficient paths from the source to this node due to the time window constraint (*efficient path* is defined later). With each path from the source p to node i satisfying the time windows is associated a two-dimensional (time, cost) label corresponding to the start time of service at node i and the cost of the path, respectively. At node i, these labels will be denoted by (T_i^k, C_i^k) $(k = 1, 2, ...)$ to indicate the characteristics of the k^{th} path from p to i. The indices k and i may be dropped when the context is unambiguous. These labels are calculated iteratively along the path $P_i = (i_0, i_1, i_2, ..., i_H)$, where $i_0 = p$ and $i_H = i$ as:

$$(T_{i_0}, C_{i_0}) = (a_p, 0)$$

$$(T_{i_h}, C_{i_h}) = (max\{a_{i_h}, T_{i_{h-1}} + t_{i_{h-1},i_h}\}, C_{i_{h-1}} + c_{i_{h-1},i_h}), \quad h = 1, 2, ..., H.$$

The most basic operation in labeling algorithms is the comparison of different labels. To compare two two-dimensional labels, we use two well-known ordering relationships, *dominance ordering* and *lexicographic ordering*.

Definition 1 Let (a_1, b_1) and (a_2, b_2) be two two-dimensional labels. The first label *dominates* the second, i.e. $(a_1, b_1) \prec (a_2, b_2)$ if and only if $a_1 \leq a_2$, $b_1 \leq b_2$, and at least one inequality is strict.

Definition 2 Let (a_1, b_1) and (a_2, b_2) be two two-dimensional labels. The first label is *lexicographically less than* the second, i.e. $(a_1, b_1) <_L (a_2, b_2)$ if and only if $a_1 < a_2$; or $a_1 = a_2$ and $b_1 < b_2$. Denote $(a_1, b_1) \leq_L (a_2, b_2)$ if (a_1, b_1) is lexicographically less than or equal to (a_2, b_2).

The dominance ordering is not a total ordering. Thus, two labels may not dominate each other. By contrast, the lexicographic ordering is a total ordering, so if the first label is not lexicographically less than the second, then the second label must be lexicographically less than or equal to the first.

Given a set of labels, to compare an individual label with all other labels in the set, we use the concepts: *efficient label* and *lexicographic minimum label*.

Definition 3 Given a set of two-dimensional labels Q, a label $(a, b) \in Q$ is said to be *efficient with respect to* Q if no other labels at Q dominates it.

Definition 4 Given a set of two-dimensional labels Q, a label $(a, b) \in Q$ is said to be the *lexicographic minimum label* of set Q if it is lexicographically less than or equal to any other label at Q, i.e. $(a, b) \leq_L (x, y)$ for any $(x, y) \in Q \setminus \{(a, b)\}$.

Definition 5 Let (T_i, C_i) be a label at node i, it is called an *efficient label of node* i if it is efficient with respect to the set of labels of node i. Its corresponding path from p to i is called an *efficient path of node* i.

Let (T_i^1, C_i^1) and (T_i^2, C_i^2) be two labels at node i, if (T_i^1, C_i^1) dominates (T_i^2, C_i^2), then we can remove (T_i^2, C_i^2) from the label set of node i since any label originating from label (T_i^2, C_i^2) will be dominated by a label originating from label (T_i^1, C_i^1). So, for each node, we need only to keep the labels undominated by any other label of this node, i.e. the efficient labels of this node. An efficient path of node i is the shortest path arriving at node i at the time no later than T_i. This implies the possibility of several efficient paths for each node. The shortest path from p to q respecting the time window constraints is obtained directly from the least cost label among the set of efficient labels at node q.

Let $\Gamma(i) = \{j | (i, j) \in A\}$ be the set of successors of node i and $R_i = \bigcup_k \{(T_i^k, C_i^k)\}$ be the set of efficient labels of node $i \in V$. There are two kinds of basic procedures in labeling algorithms for the SPPTW.

One is called the *treatment of a label*, the other is the *treatment of a node* . They are similar to the procedure of *scanning a node* in the labeling algorithms for the SPP.

The treatment of a label (T_i^k, C_i^k): consists of creating a new label for each $j \in \Gamma(i)$ by adding arcs (i,j) to the path from p to i associated with label (T_i^k, C_i^k). The new label for a given $j \in \Gamma(i)$ is:

$$f_{ij}(T_i^k, C_i^k) = \begin{cases} (max\{a_j, T_i^k + t_{ij}\}, C_i^k + c_{ij}), & \text{if } T_i^k + t_{ij} \leq b_j \\ \phi, & \text{otherwise} \end{cases}$$

Then the new set of efficient labels at node j is given by

$$EFF(f_{ij}(T_i^k, C_i^k) \cup R_j),$$

where $EFF(X)$ is the set of efficient labels among the label set X.

The treatment of a node i: treats all labels in R_i. The set of new labels at each node $j \in \Gamma(i)$ is $\bigcup_k f_{ij}(T_i^k, C_i^k)$. Hence, the new set of efficient labels at node j is given by $EFF(\bigcup_k f_{ij}(T_i^k, C_i^k) \cup R_j)$.

The time windows and positive durations guarantee the finiteness of feasible paths, but they do not guarantee the feasible paths are all elementary, i.e. each node in a feasible path is visited only once. We do allow cycles to exist in a feasible path for the SPPTW. The SPPTW is NP-hard in the ordinary sense and there exist pseudopolynomial algorithms for it, for example, the one we shall present later in this paper. By the way, we should mention that when no cycle is allowed in a feasible path, the problem is called the elementary shortest path problem with time windows (ESPPTW) which is NP-hard in the strong sense and hence there is no pseudopolynomial algorithm for it unless $P = NP$. For the complexity issue of the SPPTW and ESPPTW mentioned here, see Dror [11].

2 The Generalized Threshold Algorithm for the SPPTW

In this section, we propose a new labeling algorithm for the SPPTW. It is generalized from the threshold method of Glover, Glover and Klingman [13] for the SPP. For convenience, we call our algorithm the *generalized threshold algorithm* (GTA). It is worth noting that the threshold

algorithm is one of the fastest algorithms for the SPP and it always outperforms the Dijkstra's algorithm and the Bellman-Ford algorithm (see, e.g. [13], [2]).

The basic procedure of the GTA is the treatment of a label. As in Glover, Glover and Klingman [13], we use three queues Q_1, Q_2, and Q_3 to maintain the candidate list L of labels. At any time, the labels in Q_1 are all lexicographically less than those ones in Q_2 that are lexicographically less than those ones in Q_3. At each iteration, the label removed from L is the bottom label of Q_1, a label entering L is compared with the current threshold label. If it is lexicographically less than or equal to the threshold label, then it is added at the bottom of Q_2, otherwise it is added at the bottom of Q_3. When Q_1 becomes empty, then all the labels in Q_2 are transferred to Q_1; when both Q_1 and Q_2 are empty, the threshold label is adjusted to a level above the lexicographic minimum label of Q_3, and the labels of Q_3 that are lexicographically less than or equal to the threshold label are transferred to Q_1. The algorithm terminates when Q_1, Q_2 and Q_3 are all empty.

We present the new algorithm as follows where R_j represents the set of efficient labels of node j.

Algorithm GTA:

Step 0: *Initialize*

(0.1) Initialize the set of efficient labels for each node $i \in V$:

$R_p = \{(T_p^1 = a_p, C_p^1 = 0)\}$,

$R_i = \{(T_i^1 = a_i, C_p^1 = \infty)\}$, for each $i \in V \setminus \{p\}$

(0.2) Initialize three queues Q_1, Q_2, and Q_3:

$Q_1 = \{(T_p^1 = a_p, C_p^1 = 0)\}$,

$Q_2 = \phi$,

$Q_3 = \phi$

(0.3) Initialize threshold label (T_{thres}, C_{thres}):

$(T_{thres}, C_{thres}) = (t_0, c_0)$

Step 1: *Select the bottom label from* Q_1

If $Q_1 = \phi$, go to Step 3.

Otherwise, select and remove the bottom label from Q_1

Step 2: *Treat the selected label* (T_i^k, C_i^k)

For each $j \in \Gamma(i) = \{j|(i,j) \in A\}$, do the following:

(2.1) *Check time window*

if $T_i^k + t_{ij} \leq b_j$, go to (2.2);

otherwise, choose next $j \in \Gamma(i)$, go to (2.1)

(2.2) *Creat new label* (T_j^{new}, C_j^{new}) *for node* j

$T_j^{new} = max\{a_j, T_i^k + t_{ij}\}$,

$C_j^{new} = C_i^k + c_{ij}$

(2.3) *Check if* (T_j^{new}, C_j^{new}) *is efficient for node* j

If (T_j^{new}, C_j^{new}) is dominated by some label in R_j, then choose next $j \in \Gamma(i)$, go to (2.1)

otherwise do the following:

(2.3.1) Find every label in R_j which is dominated by (T_j^{new}, C_j^{new}), remove it from R_j, and remove it from Q_1, Q_2 or Q_3 if it is in Q_1, Q_2 or Q_3.

(2.3.2) Update the set of efficient labels of node j: $R_j = R_j \cup \{(T_j^{new}, C_j^{new})\}$

(2.3.3) Compare (T_j^{new}, C_j^{new}) with (T_{thres}, C_{thres}). If $(T_j^{new}, C_j^{new}) \leq_L (T_{thres}, C_{thres})$, then add (T_j^{new}, C_j^{new}) at the bottom of Q_2, otherwise add (T_j^{new}, C_j^{new}) at the bottom of Q_3

Go to Step 1.

Step 3: *Partition labels*

If $Q_2 \neq \phi$, transfer all the labels in Q_2 to Q_1, update Q_2 to be empty, go to Step 1.

otherwise, do the following:

(3.1) if $Q_3 = \phi$, STOP. otherwise, go to (3.2).

(3.2) update the threshold label (T_{thres}, C_{thres}), find all the labels in Q_3 which are lexicographically less than or equal to the new threshold label (T_{thres}, C_{thres}), transfer these labels to Q_1, remove these labels from Q_3, go to step 1.

Threshold labels used in the algorithm are computed as follows. Denote the initial threshold label by $(T^0_{thres}, C^0_{thres})$, and denote the current and the next threshold labels by $(T^n_{thres}, C^n_{thres})$, and $(T^{n+1}_{thres}, C^{n+1}_{thres})$, respectively.

$$(T^0_{thres}, C^0_{thres}) = (T_{basic}, C_{basic}),$$
$$(T^{n+1}_{thres}, C^{n+1}_{thres}) = (T^n_{thres}, C^n_{thres}) + (1,1) + (T_{basic}, C_{basic}),$$

where (T_{basic}, C_{basic}) is computed from the characteristics of the network, such as the density of the network, the duration t_{ij} and cost c_{ij} of each arc $(i,j) \in A$. In the computational experiment we have done, we use the following formula for computing (T_{basic}, C_{basic}).

$$(T_{basic}, C_{basic}) = (T_{avg}, C_{avg}) * PARAM/DENSE,$$

where:

$DENSE = \min(50, |A|/|V|)$,

$T_{avg} = \frac{1}{|A|} \sum_{(i,j)\in A} t_{ij}$, the average duration of all the arcs in the network,

$C_{avg} = \frac{1}{|A|} \sum_{(i,j)\in A} c_{ij}$, the average cost of all the arcs in the network,

$5 \leq PARAM \leq 10$.

If at some iteration, no label in Q_3 is lexicographically less than or equal to the updated threshold label, then the threshold label is updated by the following formula:

$$(T_{thres}, C_{thres}) = (T_{lexmin}, C_{lexmin}) + (T_{basic}, C_{basic}),$$

where

(T_{lexmin}, C_{lexmin}) = lexicographic minimum label of the current Q_3.

This can guarantee that at least one label in Q_3 will pass the threshold test.

Now, we give an estimation of the worst-case complexity of the GTA.

Lemma 1 Whenever Q_1 becomes empty, at least one label of some node will stay efficient for that node permanently from then on.

Proof: Clearly, at any time, each label in Q_2 is lexicographically less than all the labels in Q_3 since each label in Q_2 is lexicographically less than or equal to the threshold while each in Q_3 is lexicographically greater than the same threshold. Except the initialization step, Q_1 is always formed by some labels from Q_2 if $Q_2 \neq \phi$, otherwise from Q_3. Then before treating any label of Q_1, each label in Q_1 is lexicographically less than each one in Q_2 and Q_3, which implies that the lexicographic minimum label of Q_1 is the lexicographic minimum label of $Q_1 \cup Q_2 \cup Q_3$ as well. Let this label be (T_i^k, C_i^k) associated with node i. Since $t_{uv} > 0$ for all $(u, v) \in A$, and every label generated from now on will originate from some label in the present Q_1, Q_2 or Q_3, thus every label generated from now on is lexicographically greater than some label in the present Q_1, Q_2 or Q_3. So, every label generated from now on will be lexicographically greater than (T_i^k, C_i^k), which implies that (T_i^k, C_i^k) will stay undominated by any other label of node i from now on. In other words, (T_i^k, C_i^k) will stay efficient for node i permanently from now on. After all the labels in the present Q_1 are treated, (T_i^k, C_i^k) will disappear from Q_1, and a new Q_1 will be formed, hence a new label like (T_i^k, C_i^k) will appear again. □

If we define an *iteration* to be the process starting with treating the first label of Q_1 until this Q_1 becomes empty, then we have the following result.

Lemma 2 The algorithm will terminate within at most D iterations, where $D = \sum_{i \in V}(b_i - a_i + 1)$.

Proof: Since there are at most $b_i - a_i + 1$ efficient label at a node i, hence there are totally at most D efficient labels for the whole set of nodes. By *Lemma 1*, each iteration generates at least one efficient label, so the result follows immediately. □

Theorem 1 The worst-case time complexity of the algorithm is bounded by $O(D^3)$.

Proof: We first estimate the time needed to treat a label. When

treating a label (T_i^k, C_i^k) of some node i, the algorithm creates a new label (T_j^{new}, C_j^{new}) for each node $j \in \Gamma(i)$, then tries to update R_j based on (T_j^{new}, C_j^{new}), which can be finished within $O(|R_j|)$ by simply comparing (T_j^{new}, C_j^{new}) with each label of R_j. Since $\sum_{j\in\Gamma(i)} |R_j| \leq \sum_{j\in\Gamma(i)}(b_i - a_i + 1) \leq D$, the time used to treat a label is bounded by $O(D)$. At each iteration, the algorithm treats at most D labels since $|Q_1| \leq D$, thus the time needed at each iteration is bounded by $O(D^2)$. By *Lemma 2*, the total time is therefore bounded by $O(D^3)$. □

Unlike the GPLA, the algorithm GTA may treat unnecessary labels at each iteration which may be dominated by other labels generated at later iterations. The advantage of the GTA over GPLA is that it avoids computing the lexicographical minimum label of the current label set L, which can definitely save cpu time.

3 Performance of the GTA

In this section, we present a computational comparison between the generalized permanent labeling algorithm GPLA of Desrochers and Soumis [4] with our new algorithm GTA presented in section 3.

The algorithms are coded in C and tested on a Silicon Graphics Iris Workstation. The design of the testing problems mainly depends on three parameters:

1. Number of nodes in the network. We use 5 different numbers: 100, 250, 500, 1000 and 2000.

2. Average number of outgoing arcs per node. We use 4 different numbers: 10, 25, 50 and 100.

3. Average time window width. We use 4 different widths: 50, 100, 200 and 400.

Ten different test problems are generated for each feasible combination of the above three parameters. These test problems are generated in a similar way to the one described in Desrochers and Soumis [4] as follows:

1. the nodes are dispersed in the square [0, 500] × [0, 500] by a uniform distribution, the source node is located at the central point

(250, 250);

2. the duration of each arc (i, j), t_{ij} is equal to the Euclidean distance between the node i and j plus a perturbation uniformly distributed in [5, 25];

3. the cost of each arc (i, j), c_{ij} is equal to t_{ij} minus a large number 3333;

4. the center of the time window of a node i is equal to the Euclidean distance from the source node to i plus a perturbation uniformly distributed in [10, 50]. The width of the time window of node i is a number uniformly drawn from $[\frac{2}{3}AVG, \frac{4}{3}AVG]$, where AVG is the predetermined average time window width. The time window of i is thus determined by the center and width.

Table 1 and 2 list the comparison results based on 800 problems generated as above (10 for each possible combination of the three parameters). Every cpu time in the table is the average of the ten problems with the same combination of the three parameters.

From the experiments we have done, we can conclude that the GTA outperforms the GPLA on every test problem and the average running time of the GTA is about 40% less than that of the GPLA. Particularly, for those problems with looser time windows and larger number of nodes, the performance difference between the GTA and GPLA becomes more obvious and much bigger.

4 Conclusion

We have proposed a new label correcting algorithm for the shortest path problem with time windows based on the threshold algorithm for the unconstrained shortest path problem. The worst case complexity of the new algorithm is higher than that of the generalized permanent labeling algorithm in the literature. By contrast, the average performance of the previous one is always better than the latter one. So, this shows once again that an algorithm with higher worst-case complexity may be faster, in the sense of the average-case performance, than another algorithm with lower worst-case complexity.

Table 1: Execution times (in cpu secs.) of GPLA and GTA on problems with average time window width equal to 50 or 100

problem		average time window width = 50		average time window width = 100	
nodes	arcs	GPLA	GTA	GPLA	GTA
100	1000	0.012	0.008	0.024	0.018
100	2500	0.034	0.026	0.042	0.028
100	5000	0.070	0.048	0.141	0.110
100	10000	0.212	0.178	0.350	0.281
250	2500	0.028	0.019	0.030	0.021
250	6250	0.091	0.062	0.149	0.102
250	12500	0.290	0.184	0.679	0.476
250	25000	0.710	0.468	1.645	1.253
500	5000	0.089	0.0471	0.120	0.068
500	12500	0.258	0.156	0.441	0.280
500	25000	0.933	0.601	1.857	1.235
500	50000	2.494	1.750	5.853	4.441
1000	10000	0.200	0.106	0.283	0.148
1000	25000	0.556	0.321	0.947	0.530
1000	50000	1.820	1.076	3.190	2.102
1000	100000	5.294	3.123	10.850	6.547
2000	20000	0.289	0.130	0.355	0.170
2000	50000	0.896	0.432	1.573	0.841
2000	100000	4.234	2.170	9.871	4.963
2000	200000	14.210	7.057	33.142	16.850
average cpu time		1.636	0.898	3.577	2.023

Table 2: Execution times (in cpu secs.) of GPLA and GTA on problems with average time window width equal to 200 or 400

problem		average time window width = 200		average time window width = 400	
nodes	arcs	GPLA	GTA	GPLA	GTA
100	1000	0.018	0.011	0.034	0.020
100	2500	0.091	0.067	0.241	0.182
100	5000	0.254	0.205	2.844	2.210
100	10000	0.935	0.610	4.782	3.023
250	2500	0.062	0.043	0.100	0.066
250	6250	0.273	0.186	0.639	0.437
250	12500	1.663	1.215	4.321	3.254
250	25000	5.236	4.210	23.180	15.054
500	5000	0.176	0.089	0.324	0.191
500	12500	0.861	0.514	1.985	1.278
500	25000	3.879	2.517	13.201	7.985
500	50000	15.310	9.748	93.870	54.318
1000	10000	0.429	0.236	0.835	0.482
1000	25000	1.720	1.014	4.317	2.636
1000	50000	7.438	4.686	35.670	20.342
1000	100000	32.654	17.980	143.445	77.673
2000	20000	0.542	0.267	1.118	0.600
2000	50000	2.960	1.617	7.412	4.437
2000	100000	18.419	10.610	54.282	29.174
2000	200000	94.246	51.550	302.446	149.503
average cpu time		9.359	5.296	34.752	18.643

Acknowledgement

This research was supported in part by grant AFOSR-F49620-93-1-0098 from the Air Force Office of Scientific Research.

References

[1] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.

[2] D. Bertsekas. A simple and fast label correcting algorithm for shortest paths. *Networks*, 23:703–709, 1993.

[3] E. Denardo and B. Fox. Shortest-route methods: 1. reaching, pruning and buckets. *Operations Research*, 17:357–377, 1979.

[4] M. Desrochers and F. Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *INFOR*, 26:191–212, 1988.

[5] M. Desrochers and F. Soumis. A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, 35:242–254, 1988.

[6] J. Desrosiers, P. Pelletier, and F. Soumis. Plus court chemin avec contraintes d'horaires, r.a.i.r.o. *Recherche Operationnelle*, 17:357–377, 1983.

[7] J. Desrosiers, M. Solomon, and F. Soumis. Time constrained routing and scheduling. In C. Monma, T. Magnanti, and M. Ball, editors, Handbook in Operations Research and Management Science*, Volume on* Networks. North Holland, 1995.

[8] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[9] R. Dial, F. Glover, D. Karney, and D. Klingman. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*, 9:215–248, 1979.

[10] E. Dijkstra. A note on two problems in cnnection with graphs. *Nurmer. Math.*, 1:269–271, 1959.

[11] M. Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42:977–978, 1994.

[12] G. Gallo and S. Pallottino. Shortest path algorithms. *Annals of Operations Research*, 7:3–79, 1988.

[13] F. Glover, R. Glover, and D. Klingman. The threshold shortest path algorithm. *Networks*, 14:25–36, 1984.

[14] U. Pape. Algorithm 562: Shortest path lengths. *ACM Transactions on Mathematical Software*, 6:450–455, 1980.