



# On approximating a geometric prize-collecting traveling salesman problem with time windows <sup>☆</sup>

Reuven Bar-Yehuda <sup>a</sup>, Guy Even <sup>b,\*</sup>, Shimon (Moni) Shahar <sup>b</sup>

<sup>a</sup> *Computer Science Department, Technion, Haifa 32000, Israel*

<sup>b</sup> *Department of Electrical Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel*

Received 15 July 2003

Available online 10 January 2004

---

## Abstract

We study a scheduling problem in which jobs have locations. For example, consider a repairman that is supposed to visit customers at their homes. Each customer is given a time window during which the repairman is allowed to arrive. The goal is to find a schedule that visits as many homes as possible. We refer to this problem as the prize-collecting traveling salesman problem with time windows (TW-TSP).

We consider two versions of TW-TSP. In the first version, jobs are located on a line, have release times and deadlines but no processing times. We present a geometric interpretation of TW-TSP on a line that generalizes the longest monotone subsequence problem. We present an  $O(\log n)$  approximation algorithm for this case, where  $n$  denotes the number of jobs. This algorithm can be extended to deal with non-unit job profits.

The second version deals with a general case of asymmetric distances between locations. We define a density parameter that, loosely speaking, bounds the number of zig-zags between locations within a time window. We present a dynamic programming algorithm that finds a tour that visits at least  $OPT/density$  locations during their time windows. This algorithm can be extended to deal with non-unit job profits and processing times.

© 2003 Elsevier Inc. All rights reserved.

---

<sup>☆</sup> An extended abstract of this paper appeared in 11th Annual European Symposium on Algorithms, in: Lecture Notes in Comput. Sci., vol. 2832, Springer-Verlag, 2003, pp. 55–66.

\* Corresponding author.

*E-mail addresses:* [reuven@cs.technion.ac.il](mailto:reuven@cs.technion.ac.il) (R. Bar-Yehuda), [guy@eng.tau.ac.il](mailto:guy@eng.tau.ac.il) (G. Even), [moni@eng.tau.ac.il](mailto:moni@eng.tau.ac.il) (S. Shahar).

## 1. Introduction

We study a scheduling problem in which jobs have locations. For example, consider a repairman that is supposed to visit customers at their homes. Each customer is given a time window during which the repairman is allowed to arrive. The goal is to find a schedule that visits as many homes as possible. We refer to this problem as the prize-collecting traveling salesman problem with time windows (TW-TSP).

### *Previous work*

The goal in previous works on scheduling with locations differs from the goal we consider. The goal in previous works is to minimize the makespan (i.e., the completion time of the last job) or minimize the total waiting time (i.e., the sum of times that elapse from the release times till jobs are served). Tsitsiklis [8] considered the special case in which the locations are on a line. Tsitsiklis proved that verifying the feasibility of instances in which both release times and deadlines are present is strongly NP-complete. Polynomial algorithms were presented for the cases of (i) either release times or deadlines, but not both, and (ii) no processing time. Karuno et al. [6] considered a single vehicle scheduling problem which is identical to the problem studied by Tsitsiklis (i.e., locations on a line and minimum makespan). They presented a 1.5-approximation algorithm for the case without deadlines (processing and release times are allowed). Karuno and Nagamochi [7] considered multiple vehicles on a line. They presented a 2-approximation algorithm for the case without deadlines. Augustine and Seiden [1] presented a PTAS for single and multiple vehicles on trees with a constant number of leaves.

### *Our results*

We consider two versions of TW-TSP. In the first version, TW-TSP on a line, jobs are located on a line, have release times, deadlines, but no processing times. We present an  $O(\log n)$  approximation algorithm for this case, where  $n$  denotes the number of jobs. Our algorithm also handles a weighted case, in which a profit  $p(v)$  is gained if location  $v$  is visited during its time window.

The second version deals with a general case of asymmetric distances between locations (asymmetric TW-TSP). We define a density parameter that, loosely speaking, bounds the number of zig-zags between locations within a time window. We present a dynamic programming algorithm that finds a tour that visits at least  $OPT/density$  locations during their time windows. This algorithm can be extended to deal with non-unit profits and processing times.

### *Techniques*

Our approach is motivated by a geometric interpretation. We reduce TW-TSP on a line to a problem called MAX-MONOTONE-TOUR. In MAX-MONOTONE-TOUR, the input consists of a collection of slanted segments in the plane, where the slope of each segment is 45 degrees. The goal is to find an  $x$ -monotone curve starting at the origin that intersects as many segments as possible. MAX-MONOTONE-TOUR generalizes the longest monotone subsequence problem [4]. A basic procedure in our algorithms involves the construction

of an arc weighted directed acyclic graph and the computations of a max-weight path in it [5]. Other techniques include interval trees and dynamic programming algorithms.

### Organization

In Section 2, we formally define TW-TSP. In Section 3, we present approximation algorithms for TW-TSP on a line. We start with an  $O(1)$ -approximation algorithm for the case of unit time-windows and end with an  $O(\log n)$ -approximation algorithm. In Section 4 we present algorithms for the non-metric version of TW-TSP.

## 2. Problem description

We define the prize-collecting traveling salesman problem with time-windows (TW-TSP) as follows. Let  $(V, \ell)$  denote a metric space, where  $V$  is a set of points and  $\ell$  is a metric. The input of a TW-TSP instance over the metric space  $(V, \ell)$  consists of:

- A subset  $S \subseteq V$  of points.
- Each element  $s \in S$  is assigned a *profit*  $p(s)$ , a *release time*  $r(s)$ , and *deadline*  $d(s)$ .
- A special point  $v_0 \in S$ , called the *origin*, for which  $p(v_0) = r(v_0) = d(v_0) = 0$ .

The points model cities in TSP jargon or jobs in scheduling terminology. The distance  $\ell(u, v)$  models the amount of time required to travel from  $u$  to  $v$ . We refer to the interval  $[r(v), d(v)]$  as the *time window* of  $v$ . We denote the time window of  $v$  by  $I_v$ .

A *tour* is a sequence of pairs  $(v_i, t_i)$ , where  $v_i \in V$  and  $t_i$  is an arrival time. (Recall that the point  $v_0$  is the origin.) The feasibility constraints for a tour  $\{(v_i, t_i)\}_{i=0}^k$  are as follows:

$$t_0 = 0, \quad t_{i+1} \geq t_i + \ell(v_i, v_{i+1}).$$

A *TW-tour* is a tour  $\{(v_i, t_i)\}_{i=0}^k$  that satisfies the following conditions:

- (1) The tour is simple (multiplicity of every vertex is one).
- (2) For every  $0 \leq i \leq k$ ,  $v_i \in S$ .
- (3) For every  $0 \leq i \leq k$ ,  $t_i \in I_{v_i}$ .

The *profit* of a TW-tour  $\mathcal{T} = \{(v_i, t_i)\}_{i=0}^k$  is defined as

$$p(\mathcal{T}) = \sum_{i=0}^k p(v_i).$$

The goal in TW-TSP is to find a TW-tour with maximum profit.

We refer to TW-tours simply as sequences of points in  $S$  without attaching times since we can derive feasible times that satisfy  $t_i \in I_{v_i}$  as follows:

$$t_0 = 0, \quad t_i = \max\{t_{i-1} + \ell(v_{i-1}, v_i), r(v_i)\}. \quad (1)$$

One can model multiple jobs residing in the same location (but with different time windows) by duplicating the point and setting the distance between copies of the same point to zero (hence the metric becomes a semi-metric).

### 3. TW-TSP on a line

In this section we present approximation algorithms for TW-TSP on a line. TW-TSP on a line is a special case of TW-TSP in which  $V = \mathbb{R}$ . Namely, the points are on a the real line and  $\ell(u, v) = |u - v|$ .

We begin by reducing TW-TSP on a line to a geometric problem of intersecting as many slanted segments as possible using an  $x$ -monotone curve. We then present a constant ratio approximation algorithm for the special case in which the length of every time window is one. We use this algorithm to obtain an  $O(\log L)$ -approximation, where  $L = \max_v |I_v| / \min_u |I_u|$ . Finally, we present an  $O(\log n)$ -approximation algorithm, where  $n$  denotes the size of  $S$ .

For simplicity we consider the case of unit point profits (i.e.,  $p(v) = 1$ , for every  $v$ ). The case of weighted profits easily follows.

#### 3.1. A reduction to MAX-MONOTONE-TOUR

We depict an instance of TW-TSP on a line using a two-dimensional diagram (see Fig. 1). The  $x$ -axis corresponds to the value of a point. The  $y$ -axis corresponds to time. A time window  $[r(v), d(v)]$  of point  $v$  is drawn as a vertical segment, the endpoints of which are  $(v, r(v))$  and  $(v, d(v))$ .

We now rotate the picture by 45 degrees. The implications are: (i) segments corresponding to time windows are segments with a 45 degree slope, and (ii) feasible tours are (weakly)  $x$ -monotone curves; namely, a curve with slopes in the range  $[0, 90]$  degrees.

This interpretation reduces TW-TSP on a line to the problem of MAX-MONOTONE-TOUR defined as follows (see Fig. 2). The input consists of a collection of slanted segments in the plane, where the slope of each segment is 45 degrees. The goal is to find an  $x$ -monotone curve starting at the origin that intersects as many segments as possible. Note that the MAX-MONOTONE-TOUR problem generalizes the longest monotone subsequence problem [4]; the reduction is obtained by considering segments of zero length.

We remark that our results can be extended also to instances of MAX-MONOTONE-TOUR in which the segments have arbitrary slopes. Namely, it is not essential that all the segments have the same slope of 45 degrees.

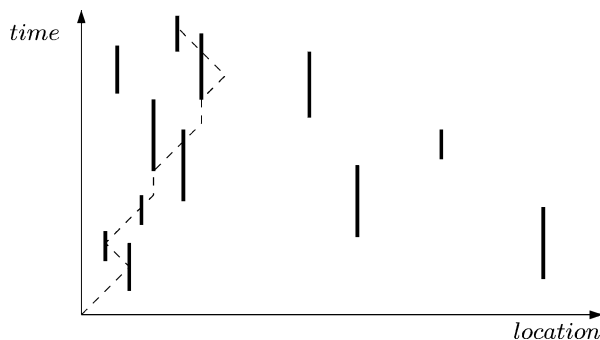


Fig. 1. A two-dimensional diagram of an instance of TW-TSP on a line.

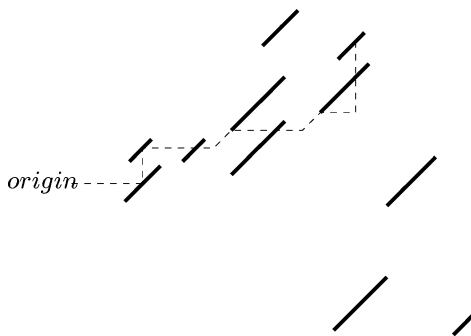


Fig. 2. A MAX-MONOTONE-TOUR instance obtained after rotation by 45 degrees.

### 3.2. Unit time windows

In this section we present an 8-approximation algorithm for the case of unit time windows. In terms of the MAX-MONOTONE-TOUR problem, this means that the length of each slanted segment is 1.

We begin by overlaying a grid whose square size is  $1/\sqrt{2} \times 1/\sqrt{2}$  on the plane. We shift the grid so that endpoints of the slanted segments do not lie on the grid lines. It follows that each slanted segment intersects exactly one vertical (respectively horizontal) line of the grid. (A technicality that we ignore here is that we would like the origin to be a grid-vertex even though the grid is shifted.) Consider a directed-acyclic graph (DAG) whose vertices are the crossings of the grid and whose edges are the vertical and horizontal segments between the vertices. We direct all the horizontal DAG edges in the positive  $x$ -direction, and we direct all the vertical DAG edges in the positive  $y$ -direction. We assign each edge  $e$  of the DAG a weight  $w(e)$  that equals the number of slanted segments that intersect  $e$ . The algorithm computes a path  $p$  of maximum weight in the DAG starting from the origin (see [3, p. 538]). The path is the tour that the agent will use. We claim that this is an 8-approximation algorithm.

**Theorem 1.** *The approximation ratio of the algorithm is 8.*

We prove Theorem 1 using the two claims below. Given a path  $q$ , let  $k(q)$  denote the number of slanted segments that intersect  $q$ . Let  $p^*$  denote an optimal path in the plane, and let  $p'$  denote an optimal path restricted to the grid. Let  $k^* = k(p^*)$ ,  $k' = k(p')$ , and  $k = k(p)$ .

**Claim 1.**  $k \geq k'/2$ .

**Proof.** Let  $w(q)$  denote the weight of a path  $q$  in the DAG. We claim that, for every grid-path  $q$ ,

$$w(q) \geq k(q) \geq w(q)/2.$$

The fact that  $w(q) \geq k(q)$  follows directly from the definition of edge weights. The part  $k(q) \geq w(q)/2$  follows from the fact that every slanted segment intersects exactly two grid

edges. Hence, a slanted segment that intersects  $q$  may contribute at most 2 to  $w(q)$ . Since the algorithm computes a maximum weight path  $p$ , we conclude that

$$k(p) \geq w(p)/2 \geq w(p')/2 \geq k(p')/2,$$

and the claim follows.  $\square$

**Claim 2.**  $k' \geq k^*/4$ .

**Proof.** Let  $C_1, \dots, C_m$  denote the set of grid cells that  $p^*$  traverses. We decompose the sequence of traversed cells into blocks. We say that point  $(x_1, y_1)$  dominates point  $(x_2, y_2)$  if  $x_1 \leq x_2$  and  $y_1 \leq y_2$ . We extend domination to sets (blocks, respectively) as follows: a set (block, respectively)  $B$  dominates a set (block, respectively)  $B'$  if every point in  $B$  dominates every point in  $B'$ . Note that if point  $p_1$  dominates point  $p_2$ , then it is possible to travel from  $p_1$  to  $p_2$  along an  $x$ -monotone curve.

Let  $B_1, B_2, \dots, B_{m'}$  denote the decomposition of the traversed cells into horizontal and vertical blocks. The odd indexed blocks are horizontal blocks and the even indexed blocks are vertical blocks. We present a decomposition in which  $B_i$  dominates  $B_{i+2}$ , for every  $i$ .

We define  $B_1$  as follows. Let  $a_1$  denote the horizontal grid line that contains the top side of  $C_1$ . Let  $C_{i_1}$  denote the last cell whose top side is contained in  $a_1$ . The block  $B_1$  consists of the cells  $C_1 \cup \dots \cup C_{i_1}$ . The block  $B_2$  is defined as follows. Let  $b_2$  denote the vertical grid line that contains the right side of cell  $C_{i_1}$ . Let  $C_{i_2}$  denote the last cell whose right side is contained in  $b_2$ . The block  $B_2$  consists of the cells  $C_{i_1+1} \cup \dots \cup C_{i_2}$ . We continue decomposing the cells into blocks in this manner. Figure 3 depicts such a decomposition.

Consider the first intersection of  $p^*$  with every slanted segment it intersects. All these intersection points are in the blocks. Assume that at least half of these intersection points belong to the horizontal blocks (the other case is proved analogously). We construct a grid-path  $\tilde{p}$  as follows. The path  $\tilde{p}$  passes through the lower left corner and upper right corner of every horizontal block. For every horizontal block,  $\tilde{p}$  goes from the bottom left corner

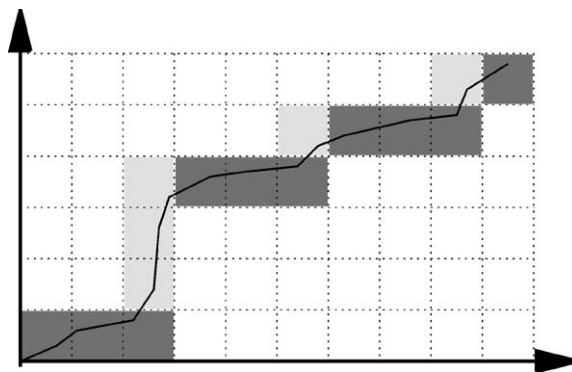


Fig. 3. A decomposition of the cells traversed by an optimal  $x$ -monotone curve into alternating horizontal and vertical blocks. The solid line depicts an optimal  $x$ -monotone curve (the slanted segments are not depicted). The dark (light, respectively) rectangles denote horizontal (vertical, respectively) blocks.

to the upper right corner along one of the following sub-paths: (a) the bottom side followed by the right side of the block, or (b) the left side followed by the top side of the block. For each horizontal block, we select the sub-path that intersects more slanted segments. The path  $\tilde{p}$  hops from a horizontal block to the next horizontal block using the vertical path between the corresponding corners.

Note that if a slanted segment intersects a block, then it must intersect its perimeter at least once. This implies that, per horizontal block,  $\tilde{p}$  is 2-approximate. Namely, the selected sub-path intersects at least half the slanted segments that  $p^*$  intersects in the block. Since at least half the intersection points reside in the horizontal blocks, it follows that  $\tilde{p}$  intersects at least  $k^*/4$  slanted segments. Since  $p'$  is an optimal path in the grid, it follows that  $k(p') \geq k(\tilde{p})$ , and the claim follows.  $\square$

### 3.2.1. Strongly polynomial running time

The size of the DAG constructed by the algorithm is weakly polynomial. Namely, the grid has to bound all the slanted segments. Hence, the width of the grid is linear in (i) the difference between the last deadline and the first release time (i.e.,  $\max_{v \in S} d(v) - \min_{v \in S} r(v)$ ) and (ii) the distance between the rightmost point and the leftmost point (i.e.,  $\max_{v', v'' \in S} v' - v''$ ).

The algorithm can be easily modified to a strongly polynomial algorithm by the following modification. Consider the bounding box of the slanted segments. Within this bounding box, the grid is constructed by drawing evenly spaced horizontal and vertical lines (i.e., distance  $1/\sqrt{2}$ ). We consider a grid line to be *interesting* if it intersects a slanted segment. Since the length of every slanted segment is 1, the number of interesting grid lines is linear. We now consider the subgrid consisting only of interesting grid lines. The maximum number of slanted segments crossed by a monotone path in the subgrid induced by the interesting grid lines equals  $k'$ . The reason is that if  $p'$  is a path in the grid, then there exists a path  $p''$  in the subgrid of interesting grid lines that intersects the same slanted segments at the same points.

We point out that one need not perform exact calculations with irrational numbers. The reason is that, for every arrangement of  $n$  slanted segments, there exists a shift of the grid, such that the minimum distance between an endpoint of a slanted segment and grid line is  $\Omega(1/n)$ . Such a shift can be computed with precision  $O(1/n)$ . Once the shift amount is computed, an approximate grid is computed; the grid lines of the approximate grid are obtained by rounding the positions of the exact grid lines with an error that is  $O(1/n)$ .

### 3.2.2. Reducing the constant

The approximation ratio in Theorem 1 can be reduced to  $(4 + \varepsilon)$ , for every constant  $1 \geq \varepsilon > 0$ , using the following construction. Let  $P = (x_1, y_1)$  and  $Q = (x_2, y_2)$  be two points on the grid, where  $P$  dominates  $Q$ , namely,  $x_1 \leq x_2$  and  $y_1 \leq y_2$  (we often say also that  $Q$  is *after*  $P$ ). Let  $b'$  denote a constant integer ( $b' = 1 + \lceil 8/\varepsilon \rceil$ ). We now refer to the subgrid induced by the interesting lines simply as the grid. Define a *turn* in a grid path to be a transition from a horizontal grid segment to vertical grid segment, or vice versa. The number of monotone rectilinear paths in the grid from  $P$  to  $Q$  with at most  $b'$  turns is bounded by  $O(n^{b'})$ . We construct a directed graph in which the vertex set is the set of

grid points and the arc set consists of all the pairs of points  $(P, Q)$ , where  $P$  dominates  $Q$ . Every arc  $(P, Q)$  is assigned a weight  $w(P, Q)$  as follows:

- For every monotone path  $\pi$  in the grid from  $P$  to  $Q$ , let  $I(\pi)$  denote the set of slanted segments that intersect  $\pi$  and do not intersect a grid segment  $(Q, Q')$  (note that  $Q$  dominates  $Q'$  so the grid segment  $(Q, Q')$  is after the path  $\pi$ ).
- Define  $w(P, Q)$  to be the maximum  $|I(\pi)|$ , where  $\pi$  is a monotone path in the grid from  $P$  to  $Q$  with at most  $b'$  turns.

We now compute a heaviest path  $p$  in this directed graph. We denote by  $w(p)$  be the weight of  $p$ . Recall that  $k^*$  denotes the maximum number of slanted segments intersected by a monotone path in the plane. The following claim shows the approximation ratio of the algorithm is  $(4 + \varepsilon)$ .

**Claim 3.**

$$k(p) \geq \frac{k^*}{4 + \varepsilon}.$$

**Proof.** Consider an optimal tour  $p^*$ . As in Claim 2, we decompose  $p^*$  into horizontal and vertical blocks. We enumerate the blocks in the decomposition, i.e.,  $B_0, B_1, \dots, B_\ell$ . Without loss of generality,  $p^*$  ends in the upper right corner of the grid, and hence, the upper right corner of  $B_\ell$  is the upper right corner of the grid. Therefore, without loss of generality, the path  $p$  also ends in the right upper corner of the grid. This guarantees that the weight of the last edge in the path  $p$  equals the number of slanted segments intersected by the corresponding grid-path (i.e., slanted segments cannot be intersected after the last edge).

We define the gain  $g_i$  of block  $B_i$  to be the number of slanted segments that  $p^*$  intersects in  $B_i$  for the first time. Let  $b = b' - 1 = \lceil 8/\varepsilon \rceil$ . There exists an offset  $a \in \{0, \dots, k - 1\}$  such that  $\sum_{\{j: \text{mod}(j,b)=a\}} g_j \leq k^*/b$ .

Let  $P_0$  denote the origin,  $P_1$  denote the upper right corner of the block  $B_a$ . For  $1 < j \leq (\ell - a)/b + 1$ , let  $P_j$  denote the upper right corner of the block  $B_{a+(j-1)b}$ . If the upper right corner of  $B_\ell$  is not defined as the last point, then we define  $P_{\lceil(\ell-a)/b+1\rceil}$  to be the upper right corner of  $B_\ell$ . Let  $r$  denote the index of the last defined point.

We construct the path  $\pi$  as follows. The path traverses the points  $P_0 \rightarrow P_1 \rightarrow \dots \rightarrow P_r$ . Between every two consecutive points  $P_j$  and  $P_{j+1}$  the path  $\pi$  follows a path  $\pi'$  between  $P_j$  and  $P_{j+1}$  that intersects  $w(P_j, P_{j+1})$  slanted segments. Such a path must exist according to the definition of  $w(P_j, P_{j+1})$ . It suffices to show that  $k(\pi) \geq k^*/(4 + \varepsilon)$ .

Consider two consecutive points  $P_j$  and  $P_{j+1}$ . Let  $B_s, \dots, B_t$  denote the set of blocks between the points  $P_j$  and  $P_{j+1}$ . We claim that

$$w(P_j, P_{j+1}) \geq \frac{1}{4} \cdot \sum_{i=s}^{t-1} g_i. \tag{2}$$

Equation (2) is proved as follows. Let  $J$  denote the subset of slanted segments that (i) can be intersected by grid-paths from  $P_j$  to  $P_{j+1}$  and (ii) can be intersected also after  $P_{j+1}$ .

Note that intersections with slanted segments in  $J$  do not contribute to the weight of the arc  $(P_j, P_{j+1})$ . Moreover, since every slanted segment is of unit length, the lower left corner of block  $B_t$  dominates every slanted segment in  $J$ . Hence, slanted segments intersected in blocks  $B_s, \dots, B_{t-1}$  do not belong to  $J$ . Following the proof of Claim 2, we can construct a grid-path from  $P_j$  to  $P_{j+1}$  that intersects at least  $\frac{1}{4} \cdot \sum_{i=s}^{t-1} g_i$  slanted segments not in  $J$ . Hence, Eq. (2) follows. (In fact, Eq. (2) is the only place where we rely on the slanted segments having unit length.)

It follows that

$$\begin{aligned} w(\pi) &= \sum_{j=0}^r w(P_j, P_{j+1}) \geq \frac{1}{4} \cdot \left( \sum_{i=0}^{\ell} g_i - \sum_{\{j: \text{mod}(j,b)=a\}} g_j \right) \\ &\geq \frac{1}{4} \cdot k^* \cdot \left( 1 - \frac{1}{b} \right) = k^* \cdot \frac{1}{4} \cdot \left( 1 - \frac{1}{\lceil 8/\varepsilon \rceil} \right) > k^* \cdot \frac{1}{4 + \varepsilon}. \end{aligned}$$

Where the last inequality holds for every  $\varepsilon \leq 1$ .

Finally, we note that for every tour  $q$ ,  $w(q) \leq k(q)$ , since the construction guarantees that, for every tour  $q$ , every slanted segment  $I$  may contribute to the weight of at most one edge in  $q$ . Therefore, intersections with slanted segments are not counted multiple times, and the claim follows.  $\square$

### 3.3. An $O(\log L)$ -approximation

In this section we present an algorithm with an approximation ratio of  $16 \cdot \log L$ , where  $L = \max_v |I_v| / \min_u |I_u|$ . We begin by considering the case that the length of every time window is in the range  $[1, 2)$ .

#### 3.3.1. Time windows in $[1, 2)$

The algorithm for unit time windows applies also for this case and yields an approximation ratio that is twice as large. Note that the choice of grid square size and the shifting of the grid implies that each slanted segment intersects at most two horizontal grid lines and at most two vertical grid lines. This increases the approximation ratio to 16 (i.e., the ratio  $k'/k$  is at most 4).

#### 3.3.2. Arbitrary time windows

In this case we partition the slanted segments to length sets; the  $i$ th length set consists of all the slanted segments whose length is in the range  $[2^i, 2 \cdot 2^i)$ . We apply the algorithm to each length set separately, and pick the best solution. The approximation ratio of this algorithm is  $16 \cdot \log L$ .

Moreover, the proof of Claim 3 relies only on the fact that each slanted segment intersects at most one vertical grid edge and one horizontal grid edge. In the case of slanted segments whose length is in the range  $[1, 2)$ , we need to skip two blocks instead of one. Hence instead of Eq. (2) we obtain:

$$w(P_j, P_{j+1}) \geq \frac{1}{4} \cdot \sum_{i=s}^{t-2} g_i.$$

By repeating the arguments in Claim 3, we obtain a  $(4 + \varepsilon) \cdot \log L$  approximation algorithm for arbitrary slanted segment lengths.

### 3.4. An $O(\log n)$ -approximation

In this section we present an approximation algorithm for MAX-MONOTONE-TOUR with an approximation ratio of  $O(\log n)$  (where  $n$  denotes the number of slanted segments). For the sake of simplicity, we first ignore the requirement that a TW-tour must start in the origin; this requirement is dealt with in the end of the section.

The algorithm is based on partitioning the set  $S$  of slanted segments to  $\log n$  disjoint sets  $S_1, \dots, S_{\log n}$ . Each set  $S_i$  satisfies a comb-property defined as follows.

**Definition 1.** A set  $S'$  of slanted segments satisfies the *comb property* if there exists a set of vertical lines  $\mathcal{L}$  such that every segment  $s \in S'$  intersects exactly one line in  $\mathcal{L}$ .

We refer to a set of slanted segments that satisfy the comb property as a comb.

We begin by presenting a constant approximation algorithm for combs. We then show how a set of slanted segments can be partitioned to  $\log n$  combs. The partitioning combined with the constant ratio approximation algorithm for combs yields an  $O(\log n)$ -approximation algorithm.

#### 3.4.1. A constant approximation algorithm for combs

Let  $S'$  denote a set of slanted segments that satisfy the comb property with respect to a set  $\mathcal{L}$  of vertical lines. We construct a grid as follows: (1) The set of vertical lines is  $\mathcal{L}$ . (2) The set of horizontal lines is the set of horizontal lines that pass through the endpoints of slanted segments. By extending the slanted segments by infinitesimal amounts, we may assume that an optimal tour does not pass through the grid's vertices. Note that the grid consists of  $2n$  horizontal lines and at most  $n$  vertical lines.

We define an edge-weighted directed acyclic graph in a similar fashion as before. The vertices are the crossings of the grid. The edges are the vertical and horizontal segments between the vertices. We direct all the horizontal DAG edges in the positive  $x$ -direction, and we direct all the vertical DAG edges in the positive  $y$ -direction. We assign each edge  $e$  of the DAG a weight  $w(e)$  that equals the number of slanted segments that intersect  $e$ . The algorithm computes a maximum weight path  $p$  in the DAG. We claim that this is a 12-approximation algorithm.

**Theorem 2.** *The approximation ratio of the algorithm is 12.*

**Proof.** The proof is similar to the proof of Theorem 1. We use the same notation where

- (i)  $k^*$  denotes the maximum number of slanted segments that an  $x$ -monotone curve can intersect;
- (ii)  $k'$  denotes the maximum number of slanted segments that a curve restricted to the grid can intersect; and

(iii)  $k = k(p)$  denotes the number of slanted segments intersected by the algorithm’s solution.

We first claim that

$$k \geq k'/3. \tag{3}$$

Every slanted segment only intersects a single vertical grid line, and hence a single vertical grid edge. Let a *column* denote the area bounded by two consecutive vertical lines. The set of horizontal grid edges contained in the same column constitutes an anti-chain in the DAG. Namely, a path in the DAG may contain at most one horizontal grid edge per column. Since a slanted segment intersects exactly one vertical line, it follows that it is contained in two consecutive columns. It follows that a slanted segment may intersect at most two horizontal grid edges along a path in the DAG. We conclude that, for every path  $q$  in the DAG,  $k(q) \geq w(q)/3$ , and hence Eq. (3) follows.

Following the proof of Claim 2, we claim that

$$k' \geq k^*/4. \tag{4}$$

The theorem follows from Eqs. (3) and (4).  $\square$

### 3.4.2. Partitioning into combs

The partitioning is based on computing a balanced interval tree [2, p. 214]. Such a tree is constructed recursively as follows: At first, bisect the set of slanted segments using a vertical line, and the comb simply equals the set of slanted segments intersected by the bisector. Now, apply this bisection to each set of slanted segments enclosed by two consecutive bisectors of the previous stage. The comb  $S_i$  equals the set of slanted segments intersected by the bisectors belonging to the  $i$ th level of the recursion. The depth of the interval tree is at most  $\log n$ , and hence, at most  $\log n$  combs are obtained. Figure 4 depicts an interval tree corresponding to a set of slanted segments. Membership of a slanted segment  $s$  in a subset corresponding to a vertical line  $v$  is marked by a circle positioned at

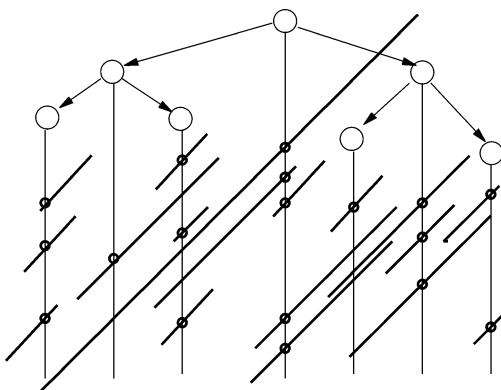


Fig. 4. An interval tree corresponding to a set of slanted segments.

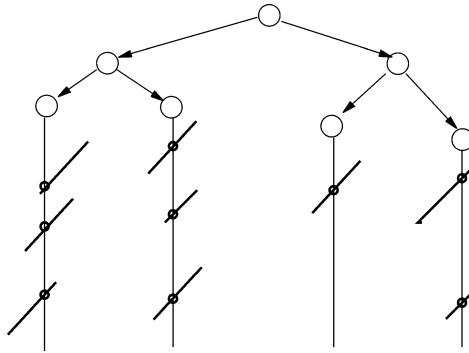


Fig. 5. A comb induced by an interval tree.

the intersection point. Figure 5 depicts a single comb; in this case the comb corresponding to the second level of the interval tree.

### 3.4.3. Finding a tour starting in the origin

The approximation algorithm can be modified to find a TW-tour starting at the origin at the price of slightly increasing the approximation ratio as follows. The first comb is the set of slanted segments that intersect the vertical line that passes through the origin. This means that the number of combs increases by at most one. We also add the vertical line that passes through the origin to the grid of every comb.

**Remark.** The algorithm for TW-TSP on a line can be easily extended to non-unit point profits  $p(v)$ . All one needs to do is assign grid edge  $e$  a weight  $w(e)$  that equals the sum of profits of the slanted segments that intersect  $e$ .

### 3.5. Reducing the constant for comb

Despite the fact that a segment may intersect many grid cells, a similar construction to the one presented in Section 3.2.2 is applicable for combs as well. The comb property ensures that, for every rectilinear path, every slanted segment intersects at most three grid-edges that appear in two consecutive blocks. Hence we may avoid the multiple counts in the arcs lengths by losing a factor of  $O(\varepsilon)$  of the segments, improving the constant in theorem 2 to  $4 + \varepsilon$ .

## 4. Asymmetric TW-TSP

In this section we present algorithms for the non-metric version of TW-TSP. Asymmetric TW-TSP is a more general version of TW-TSP in which the distance function  $\ell(u, v)$  is not a metric. Note that the triangle inequality can be imposed by metric completion (i.e., setting  $\ell(u, v)$  to be the length of the shortest path from  $u$  to  $v$ ). However, the distance function  $\ell(u, v)$  may be asymmetric in this case.

#### 4.1. Motivation

One way to try to solve TW-TSP is to

- (i) identify a set of candidate arrival times for each point,
- (ii) define an edge weighted DAG over pairs  $(v, t)$ , where  $v$  is a point and  $t$  is a candidate arrival times. The weight of an arc  $(v, t) \rightarrow (v', t')$  equals  $p(v')$ ,
- (iii) find a longest path in the DAG with respect to edge weights.

There are two obvious obstacles that hinder such an approach. First, the number of candidate arrival times may not be polynomial. Second, a point may appear multiple times along a DAG path. Namely, a path zig-zagging back and forth to a point  $v$  erroneously counts each appearance of  $v$  as a new visit. The algorithms presented in this section cope with the problem of too many candidate points using the lexicographic order applied to sequences of arrival times of TW-tours that traverse  $i$  points (with multiplicities). The second problem is not solved. Instead we introduce a measure of density that allows us to bound the multiplicity of each point along a path.

#### 4.2. Density of an instance

The quality of our algorithm for asymmetric TW-TSP depends on a parameter called the *density* of an instance.

**Definition 2.** The density of a TW-TSP instance  $\Pi$  is defined by

$$\sigma(\Pi) = \max_{u,v} \frac{|I_u|}{\ell(u, v) + \ell(v, u)}.$$

Note that  $\sigma(\Pi)$  is an upper bound on the number of “zig-zags” possible from  $u$  to  $v$  and back to  $u$  during the time window  $I_u$ . We refer to instances in which  $\sigma(\Pi) < 1$  as instances that satisfy the *no-round trips within time-windows* condition.

#### 4.3. Unit profits & no-round trips within time-windows

We first consider the case in which (i)  $\sigma(\Pi) < 1$ , and (ii) the profit of every point is one. In this section we prove the following theorem.

**Theorem 3.** *There exists a polynomial algorithm that, given an asymmetric TW-TSP instance  $\Pi$  with unit profits and  $\sigma(\Pi) < 1$ , computes an optimal TW-tour.*

**Proof.** Let  $k^*$  denote the maximum number of points that a TW-tour can visit. We associate with every tour  $\mathcal{T} = \{v_i\}_{i=0}^{k^*}$  the sequence of arrival times  $\{t_i\}_{i=0}^{k^*}$  defined in Eq. (1). Let  $\mathcal{T}^* = \{(v_i^*, t_i^*)\}_{i=0}^{k^*}$  denote a TW-tour whose sequence of arrival times is lexicographically minimal among the optimal TW-tours. We present an algorithm that computes an optimal tour  $\mathcal{T}$  whose sequence of arrival times equals that of  $\mathcal{T}^*$ .

We refer to a TW-tour that visits  $i$  points that ends in point  $v$  as  $(v, i)$ -lexicographically minimal if its sequence of arrival times is lexicographically minimal among all TW-tours that visit  $i$  points and end in point  $v$ . We claim that every prefix of  $\mathcal{T}^*$  is also lexicographically minimal. For the sake of contradiction, consider a TW-tour  $\mathcal{S} = \{u_j\}_{j=0}^i$  in which  $u_i = v_i^*$  and the arrival time to  $u_i$  in  $\mathcal{S}$  is less than  $t_i^*$ . We can substitute  $\mathcal{S}$  for the prefix of  $\mathcal{T}^*$  to obtain a lexicographically smaller optimal tour. The reason this substitution succeeds is that  $\sigma(\Pi) < 1$  implies that  $u_a \neq v_b^*$ , for every  $0 < a < i$  and  $i < b \leq k^*$ .

The algorithm is a dynamic programming algorithm based on the fact that every prefix of  $\mathcal{T}^*$  is lexicographically minimal. The algorithm constructs layers  $L_0, \dots, L_{k^*}$ . Layer  $L_i$  contains a set of states  $(v, t)$ , where  $v$  denotes the endpoint of a TW-tour that arrives at  $v$  at time  $t$ . Moreover, every state  $(v, t)$  in  $L_i$  corresponds to a  $(v, i)$ -lexicographically minimal TW-tour. Layer  $L_0$  simply contains the state  $(v_0, 0)$  that starts in the origin at time 0. Layer  $L_{j+1}$  is constructed from layer  $L_j$  as described in Algorithm 1. The procedure “replace-if-min” is given a layer  $L_{j+1}$  and a state  $(u, t')$  and updates  $L_{j+1}$  as follows: If  $L_{j+1}$  does not contain a state with  $u$  as its point, then  $(u, t')$  is added to  $L_{j+1}$ . Otherwise, let  $(u, t'') \in L_{j+1}$  denote the state in  $L_{j+1}$  that contains  $u$  as its point. The state  $(u, t')$  is added to  $L_{j+1}$  if  $t' < t''$ . If  $(u, t')$  is added, then  $(u, t'')$  is removed from  $L_{j+1}$ . Note that each layer contains at most  $n$  states, namely, at most one state per point. The algorithm stops as soon as the next layer  $L_{j+1}$  is empty. Let  $L_j$  denote the last non-empty layer constructed by the algorithm. The algorithm picks a state  $(v, t) \in L_j$  with a minimal time and returns a TW-tour (that visits  $j$  points) corresponding to this state.

**Algorithm 1.** Construct layer  $L_{j+1}$ .

1. **for all** state  $(v, t) \in L_j$ , and every  $u \neq v$  **do**
2.      $t' \leftarrow \max(r(u), t + \ell(v, u))$
3.     **if**  $t' \leq d(u)$  **then**
4.          $L_{j+1} \leftarrow \text{replace-if-min}(L_{j+1}, (u, t'))$ .
5.     **end if**
6. **end for**

The correctness of the algorithm is based on the following claim.

**Claim 4.**

- (i) If  $\mathcal{T}$  is a  $(v, i)$ -lexicographically minimal TW-tour that arrives in  $v$  at time  $t$ , then  $(v, t) \in L_i$ ;
- (ii) Every state  $(v, t)$  in layer  $L_i$  corresponds to a  $(v, i)$ -lexicographically minimal TW-tour.

**Proof.** The proof is by induction on  $i$ . The induction basis for  $i = 0$  is trivial. Consider a  $(u, i + 1)$ -lexicographically minimal TW-tour arriving in  $u$  at time  $t'$ . Assume that its prefix of length  $i$  ends in  $v$  at time  $t$ . It follows that this prefix is  $(v, i)$ -lexicographically minimal. By the induction hypothesis, it follows that  $(v, t) \in L_i$ . The construction of  $L_{i+1}$  implies that  $(u, t') \in L_{i+1}$ , as required. This completes the proof of the first part.

To prove the second part observe that if  $(u, t') \in L_{j+1}$  during the construction of  $L_{j+1}$ , then there is a TW-tour that ends in  $u$ , visits  $j + 1$  points, and arrives to  $u$  at time  $t'$ . Finally, if  $(u, t') \in L_{j+1}$ , then it must correspond to a  $(u, j + 1)$ -lexicographically minimal tour, otherwise, by part (i) it would have been kicked out of  $L_{j+1}$ . This completes the proof of the claim.  $\square$

Part (ii) of Claim 4 implies that the last layer constructed by the algorithm is indeed  $L_{k^*}$ . Since every prefix of  $\mathcal{T}^*$  is lexicographically minimal, it follows that layer  $L_i$  contains the state  $(v_i^*, t_i^*)$ . Hence, the algorithm returns an optimal TW-tour. This TW-tour also happens to be lexicographically minimal, and the theorem follows.  $\square$

#### 4.4. Arbitrary density

In this section we consider instances with arbitrary density and unit profits. The dynamic programming algorithm in this case proceeds as before but may construct more than  $k^*$  layers. We show that at most  $k^* \cdot (\lfloor \sigma(\Pi) \rfloor + 1)$  layers are constructed. A path  $q$  corresponding to a state in layer  $L_j$  may not be simple, and hence,  $k(q)$  (the actual number of visited points) may be less than  $j$  (the index of the layer).

The following claim proves the approximation ratio of the dynamic programming algorithm.

**Claim 5.** *The approximation ratio of the dynamic programming algorithm is  $\lfloor \sigma(\Pi) \rfloor + 1$ .*

**Proof.** Consider a path  $p = \{v_i\}_{i=0}^j$  corresponding to a state in layer  $L_j$ . Let  $t_i$  denote the arrival time to  $v_i$  in  $p$ . We claim that the multiplicity of every point along  $p$  is at most  $\lfloor \sigma(\Pi) \rfloor + 1$ . Pick a vertex  $v$ , and let  $i_1 < i_2 < \dots < i_a$  denote the indexes of the appearances of  $v$  along  $p$ . Since self-loops are not allowed, it follows that between every two appearances of  $v$ , the path visits another vertex. Density implies that, for every  $b = 1, \dots, a - 1$ ,

$$t_{i_{b+1}} - t_{i_b} \geq \frac{|I_v|}{\sigma(\Pi)}.$$

It follows that

$$t_{i_a} - t_{i_1} \geq (a - 1) \cdot \frac{|I_v|}{\sigma(\Pi)}.$$

Since  $r(v) \leq t_{i_1} < t_{i_a} \leq d(v)$ , it follows that  $\sigma(\Pi) \geq a - 1$ . We conclude that the multiplicity of  $v$  in  $p$  is at most  $\lfloor \sigma(\Pi) \rfloor + 1$ .

The index of the last layer found by the algorithm is at least  $k^*$ , and hence, the path computed by the algorithm visits at least  $k^*/(\lfloor \sigma(\Pi) \rfloor + 1)$  points, and the claim follows.  $\square$

#### 4.5. Non-unit profits

In this section we consider instances of asymmetric TW-TSP with non-unit profits  $p(v)$ . We first point out a trivial reduction of knapsack to asymmetric TW-TSP and then

discuss a variation of the dynamic programming algorithm with an approximation ratio of  $(1 + \varepsilon) \cdot (\lfloor \sigma(\Pi) \rfloor + 1)$ .

4.5.1. *Knapsack hardness*

Consider a knapsack instance with element sizes  $w(v)$ , element values  $p(v)$ , and knapsack size  $B$ . We reduce it to an asymmetric TW-TSP instance as follows. The point set  $V$  is the set of elements plus the origin. The distance  $\ell(u, v) = w(v)$ , for every  $u \neq v$ . The profit of  $v$  is  $p(v)$ . The time interval  $I_v = [0, B]$ , for every  $v$ .

4.5.2. *A  $(1 + \varepsilon) \cdot (\lfloor \sigma(\Pi) \rfloor + 1)$ -approximation algorithm*

We modify the dynamic programming algorithm as follows. The layers are indexed  $L_{i,j}$ . A state in layer  $L_{i,j}$  corresponds to a path that traverses  $i$  points (multiplicities are counted). A state in layer  $L_{i,j}$  is a triple  $(v, t, p)$  where  $v$  denotes a point,  $t$  denotes an arrival time, and  $p$  denotes a profit in the interval  $[(1 + \varepsilon)^j, (1 + \varepsilon)^{j+1})$ . (Note that we assume that all profits  $p(v)$  are at least 1.) Each layer  $L_{i,j}$  may contain at most one state per point. The update rule is that a state  $(v, t, p) \in L_{i,j}$  generates a state  $(u, t', p')$  which is considered for insertion to layer  $L_{i+1, \lfloor \log_{1+\varepsilon} p' \rfloor}$ . The replacement takes place only if the state  $(u, t'', p'')$  already in  $L_{i+1, \lfloor \log_{1+\varepsilon} p' \rfloor}$  satisfies  $t' < t''$ . It can be shown that the approximation ratio of this dynamic programming algorithm is  $(1 + \varepsilon) \cdot (\lfloor \sigma(\Pi) \rfloor + 1)$ .

4.6. *Processing times*

Our algorithms for asymmetric TW-TSP can be modified to handle also processing times. The processing time of point  $v$  is denoted by  $h(v)$  and signifies the amount of time that the agent must spend at a visited point. The definition of arrival times is modified to:

$$t_0 = 0, \quad t_i = \max\{t_{i-1} + h(v_{i-1}) + \ell(v_{i-1}, v_i), r(v_i)\}. \tag{5}$$

The definition of density with processing times becomes

$$\sigma(\Pi) = \max_{u,v} \frac{|I_u|}{\ell(u, v) + \ell(v, u) + h(u) + h(v)}.$$

States are generated by the dynamic programming algorithm as follows. The arrival time  $t'$  of state  $(u, t')$  generated by state  $(v, t)$  is

$$t' = \max\{t + h(v) + \ell(v, u), r(u)\}.$$

**Acknowledgments**

We thank Sanjeev Khanna and Piotr Krysta for helpful discussions. We especially thank Piotr for telling us about references [6,8]; this enabled finding all the other related references. We thank Wolfgang Slany for suggesting a nurse scheduling problem which motivated this work.

## References

- [1] J. Augustine, S.S. Seiden, Linear time approximation schemes for vehicle scheduling, in: SWAT 2002, in: *Lecture Notes in Comput. Sci.*, vol. 2368, 2002, pp. 30–39.
- [2] M. de Berg, M. van Kreveld, M. Overmars, O. Schwartzkopf, *Computational Geometry—Algorithms and Applications*, Springer-Verlag, 2000.
- [3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw–Hill, 1990.
- [4] P. Erdős, G. Szekeres, A combinatorial problem in geometry, *Compositio Math.* 2 (1935) 463–470.
- [5] M.L. Fredman, On computing the length of longest increasing subsequences, *Discrete Math.* 11 (1975) 29–35.
- [6] Y. Karuno, H. Nagamochi, T. Ibaraki, A 1.5-approximation for single-vehicle scheduling problem on a line with release and handling times, Technical Report 98007, 1998; Journal version by the same authors appeared as: Better approximation ratios for the single-vehicle scheduling problems on line-shaped networks, *Networks* 39 (4) (2002) 203–209.
- [7] Y. Karuno, H. Nagamochi, A 2-approximation algorithm for the multi-vehicle scheduling problem on a path with release and handling times, in: ESA 2001, in: *Lecture Notes in Comput. Sci.*, vol. 2161, 2001, pp. 218–229; *Discrete Appl. Math.* 129 (2) (2003) 433–447.
- [8] J.N. Tsitsiklis, Special cases of traveling salesman and repairman problems with time windows, *Networks* 22 (1992) 263–282.