On Approximating A Geometric Prize-Collecting Traveling Salesman Problem With Time Windows

(EXTENDED ABSTRACT)

Reuven Bar-Yehuda[1], Guy Even[2], and Shimon (Moni) Shahar[3]

[1] Computer Science Dept., Technion, Haifa 32000, Israel. reuven@cs.technion.ac.il
[2] Electrical-Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel. guy@eng.tau.ac.il
[3] Electrical-Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel. moni@eng.tau.ac.il

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. A geometric interpretation of the problem is used that generalizes the Erdős-Szekeres Theorem. 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.

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 [T92] 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. [KNI98] 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 [KN01] considered multiple vehicles on a line. They presented a 2-approximation algorithm for the case without deadlines. Augustine and Seiden [AS02] 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 [ES35]. 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 [F75]. 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, ℓ) denote a metric space, where V is a set of points and ℓ is a metric. The input of a TW-TSP instance over the metric space (V, ℓ) consists of: (i) A subset $S \subseteq V$ of points. (ii) Each element $s \in S$ is assigned a *profit* $p(s)$, a *release time* $r(s)$, and *deadline* $d(s)$. (iii) 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:

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

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:

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

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 $L = \frac{\max_v |I_v|}{\min_u |I_u|}$. 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(A)). 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 $(v, d(v))$.

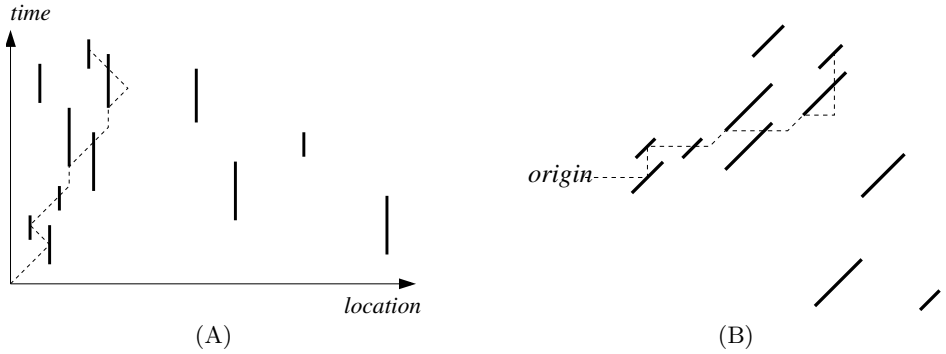

Fig. 1. (A) A two-dimensional diagram of an instance of TW-TSP on a line. (B) A MAX-MONOTONE-TOUR instance obtained after rotation by 45 degrees.

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. 1(B)). 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.

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 $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$ 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 (resp. 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. 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. $k \geq k'/2$.

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.

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

Proof. Let $C_1, \ldots, C_m$ denote the set of grid cells that p^* traverses. We decompose the sequence of traversed cells into blocks. Point (x_1, y_1) *dominates* point (x_2, y_2) if $x_1 \leq x_2$ and $y_1 \leq y_2$. A block B *dominates* a block 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 $B_1, B_2, \ldots, B_{m'}$ 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 in a_1. The block B_1 consists of the cells $C_1 \cup \cdots \cup C_{i_1}$. 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_i}. Let C_{i_2} denote the last cell whose right side is in b_2. The block B_2 consists of the cells $C_{i_1+1} \cup \cdots \cup C_{i_2}$. We continue decomposing the cells into blocks in this manner. Figure 2 depicts such a decomposition.

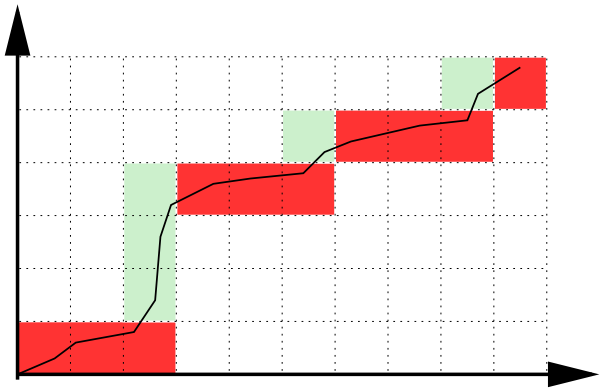

Fig. 2. A decomposition of the cells traversed by an optimal curve into alternating horizontal and vertical blocks.

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 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.

In the full version we present (i) a strongly polynomial version of the algorithm, and (ii) a reduction of the approximation ratio to $(4 + \varepsilon)$.

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

In this section we present an algorithm with an approximation ratio of $8 \cdot \log L$, where $L = \frac{\max_v |I_v|}{\min_u |I_u|}$. We begin by considering the case that the length of every time window is in the range $[1, 2)$.

Time windows in $[1, 2)$. The algorithm for unit time windows applies also for this case and yields the same approximation ratio. Note that the choice of grid square size and the shifting of the grid implies that each slanted segment intersects exactly one horizontal grid line and exactly one vertical grid line.

Arbitrary time windows. In this case we partition the slanted segments to length sets; the ith 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 $8 \cdot \log L$.

In the full version we present an algorithm with a $(4+\varepsilon) \cdot \log L$-approximation ratio.

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 $S_1, \ldots, S_{\log n}$. 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 an 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.

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.

The proof is similar to the proof of Theorem 1 and appears in the full version.

Partitioning into combs. The partitioning is based on computing a balanced interval tree [BKOS00, p. 214]. The algorithm recursively bisects the set of intervals using vertical lines, and the comb S_i equals the set of slanted segments intersected by the bisectors belonging to the ith level. The depth of the interval tree is at most $\log n$, and hence, at most $\log n$ combs are obtained. Figure 3(A) 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 the intersection point. Figure 3(B) depicts a single comb; in this case the comb corresponding to the second level of the interval tree.

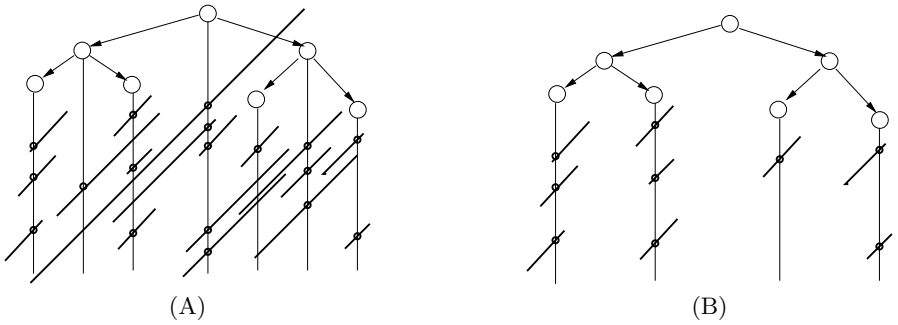

Fig. 3. (A) An interval tree corresponding to a set of slanted segments. (B) A comb induced by an interval tree.

Finding a tour starting in the origin. The approximation algorithm can be modified to find a TW-tour starting in the origin at the price of increasing the approximation ratio by a factor of two. Given a comb S_i, we consider one of two tours starting at the origin v_0. The first tour is simply the vertical ray starting in the origin. This tour intersects all the slanted segments S'_i whose projection on the x-axis contains the origin. The second tour is obtained by running the algorithm with respect $S''_i = S_i \cup \{v_0\} \setminus S'_i$. Note that S''_i is a 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 $w(e)$ thats equals the sum of profits of the slanted segments that intersect e.
In the full version we present a $(4+\varepsilon)$-approximation algorithm for a comb.

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 Π 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 Π 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 of length i 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 $L_0, \ldots, L_{k^*}$. 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. 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}

```
for all state (v,t) ∈ L_j, and every u ≠ v do
    t' ← max(r(u), t + ℓ(v,u))
    if t' < d(u) then
        L_{j+1} ← replace-if-min(L_{j+1}, (u,t')).
    end if
end for
```

The correctness of the algorithm is based on the following claim, the proof of which appears in the full version.

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

Part (ii) of the previous claim 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.

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).

Claim. The approx-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 $i_1 < i_2 < \cdots < i_a$ 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 $b = 1, \ldots, a-1$,

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

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.

4.5 Non-unit profits

In the full version we consider instances of asymmetric TW-TSP with non-unit profits $p(v)$. We present (i) a trivial reduction of Knapsack to asymmetric TW-TSP and (ii) discuss a variation of the dynamic programming algorithm with an approximation ratio of $(1+\varepsilon) \cdot (\lfloor \sigma(\Pi) \rfloor + 1)$.

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:

$$\begin{aligned} t_0 &= 0 \\ t_i &= \max\{t_{i-1} + h(v_{i-1}) + \ell(v_{i-1}, v_i), r(v_i)\}. \end{aligned} \tag{2}$$

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 would like to thank Sanjeev Khanna and Piotr Krysta for helpful discussions. We especially thank Piotr for telling us about references [T92] and [KNI98]; this enabled finding all the other related references. We thank Wolfgang Slany for suggesting a nurse scheduling problem which motivated this work.

References

[AS02] John Augustine and Steven S. Seiden , "Linear Time Approximation Schemes for Vehicle Scheduling", SWAT 2002, LNCS 2368, pp. 30-39, 2002.

[BKOS00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwartzkopf, "Computational Geometry - Algorithms and Applications", Springer Verlag, 2000.

[ES35] P. Erdős and G. Szekeres, "A combinatorial problem in geometry", Compositio Math., 2, 463– 470, 1935.

[F75] M. L. Fredman, "On computing the length of longest increasing subsequences", Discrete Math. 11 (1975), 29–35.

[KNI98] Y. Karuno, H. Nagamochi, and T. Ibaraki, "A 1.5-approximation for single-vehicle scheduling problem on a line with release and handling times", Technical Report 98007, 1998.

[KN01] Yoshiyuki Karuno and Hiroshi Nagamochi, "A 2-Approximation Algorithm for the Multi-vehicle Scheduling Problem on a Path with Release and Handling Times", ESA 2001, LNCS 2161, p. 218–229, 2001.

[T92] John N. Tsitsiklis, "Special Cases of Traveling Salesman and Repairman Problems with Time Windows", *Networks*, Vol. 22, pp. 263–282, 1992.