Polynomial time approximation schemes for the traveling repairman and other minimum latency problems.

René Sitters *

Abstract

We give a polynomial time, $(1+\epsilon)$-approximation algorithm for the traveling repairman problem (TRP) in the Euclidean plane, on weighted planar graphs, and on weighted trees. This improves on the known quasi-polynomial time approximation schemes for these problems. The algorithm is based on a simple technique that reduces the TRP to what we call the *segmented TSP*. Here, we are given numbers $l_1, \ldots, l_K$ and $n_1, \ldots, n_K$ and we need to find a path that visits at least n_h points within path distance l_h from the starting point for all $h \in \{1, \ldots, K\}$. A solution is α-approximate if at least n_h points are visited within distance αl_h. It is shown that any algorithm that is α-approximate for *every constant* K in some metric space, gives an $\alpha(1+\epsilon)$-approximation for the TRP in the same metric space. Subsequently, approximation schemes are given for this segmented TSP problem in different metric spaces. The segmented TSP with only one segment ($K = 1$) is equivalent to the k-TSP for which a $(2+\epsilon)$-approximation is known for a general metric space. Hence, this approach through the segmented TSP gives new impulse for improving on the 3.59-approximation for TRP in a general metric space. A similar reduction applies to many other minimum latency problems. To illustrate the strength of this approach we apply it to the well-studied scheduling problem of minimizing total weighted completion time under precedence constraints, $1|prec|\sum w_j C_j$, and present a polynomial time approximation scheme for the case of interval order precedence constraints. This improves on the known 3/2-approximation for this problem. Both approximation schemes apply as well if release dates are added to the problem.

1 Introduction

The traveling repairman problem (TRP) (also known as the minimum latency problem) is similar to the well-known traveling salesman problem (TSP). An instance is given by points in a metric space and a feasible solution is a path Π, starting at a given origin r, that visits each of the points. The *completion time* of a point v is the distance from r to v on path Π and the objective is to minimize the total completion time of the points. Hence, the problem can be seen as a traveling repairman who's aim is to minimize the average arrival time at the clients. The traveling salesman on the other hand aims at minimizing the travel time of the salesman himself.

The approximability of the TRP has been the subject of many papers [2, 4, 5, 12, 13, 14, 15, 17, 18, 19, 23, 26, 28, 29]. The first approximation ratio, given by Blum et al. [13], was 144 and the current smallest ratio for general metrics is 3.59 due to Chaudhuri et al. [14]. Even for trees, the best polynomial time approximation ratio was 3.59 until recently Archer and Blasiak reduced it to 3.03 [4]. $\mathcal{NP}$-hardness of the tree case was shown in [28]. For the Euclidean plane, a 3.59-approximation follows from the TRP algorithm by Goemans and Kleinberg [19] in combination with the polynomial time approximation scheme (PTAS) for the Euclidean k-TSP by Arora [7]. A quasi-polynomial time approximation scheme for the TRP on trees, the Euclidean plane and on weighted planar graphs was given by Arora and Karakostas [10, 12]. Arora and Karakostas write that they 'do not know whether the running time can be reduced to polynomial'. Here we show that this is indeed possible.

All known TRP algorithms solve some form of the k-TSP or k-MST as a subroutine. In the k-TSP (k-MST), one needs to find the shortest tour (tree) that visits at least k of the n input points. For example, the algorithm by Goemans and Kleinberg [19] first computes approximate k-TSP tours for all $k \leqslant n$ and then combines a subset of these tours into one TRP solution. An alternative approach is to make subtours of geometrically increasing length and to visit a maximum number of points in each subtour. The obtained ratio is 3.59α, where α is the approximation ratio of the k-TSP (or k-MST). So far, the best ratio for k-TSP and k-MST is $2+\epsilon$ [11]. Chaudhuri et al. [14] found a way to bypass this factor 2 in the analysis and noted that breaking the barrier of 3.59 would probably involve an approach different than combining small tours. Indeed, the exact algorithm for TRP on the line [2] and the quasi-PTAS [12] for the plane and trees are different since they find a solution directly by one dynamic program.

*Vrije Universiteit, Amsterdam. Department of Econometrics and Operations Research.

Our approach here is to combine both ideas. Instead of solving n times a k-TSP, we solve a polynomial number of relatively large subtour problems which we call the *segmented TSP*. Dynamic programming is applied to combine a subset of these subtours into one solution. As in [19], the subtours are of geometrically increasing length. However, the multiplication factor is not an absolute constant but a large constant depending on ϵ. This way, we loose only a $1+\epsilon$ factor due to the returns to the origin. The downside is that the number of subproblems increases as well as their complexity. We show that it is enough for a PTAS to solve only a polynomial number of these subproblems approximately. More precesily, it is shown that any α-approximation algorithm for the segmented TSP gives an $\alpha(1+\epsilon)$-approximation for the TRP in the same metric space. Subsequently, we give a PTAS for the segmented TSP on weighted trees, the Euclidean plane and weighted planar graphs. An intersting by-product is that this gives a new direction for improving on the general 3.59 approximation since any approximation ratio better than 3.59 for the segmented TSP gives an improved factor for the TRP! The segmented TSP has not been studied before and no constant factor approximation is known for general metric spaces. It seems unlikely though that the factor 3.59 will show up in the analysis of the segmented TSP.

The same approach can be applied to many sequencing problem with minimum total (weighted) completion time objective. This is illustrated by our second application, which is the notorious scheduling problem of minimizing total weighted completion time under precedence constraints, known as $1|prec|\sum w_jC_j$ in the standard scheduling notation. (See e.g. Graham et al. [20].) The approximability of this problem has been studied in many papers. The problem is known to be $\mathcal{NP}$-hard [24, 25] and several 2-approximation algorithms are known. The paper [3] gives a recent overview on the status of this problem. We present a polynomial time approximation scheme for the case of interval order precedence constraints. Woeginger [30] gave a 1.62-approximation algorithm and a 3/2-approximation was given by Ambühl et al [3]. Somewhat surprisingly, the same paper shows that scheduling interval orders is in fact $\mathcal{NP}$-hard. Hence, our PTAS closes the gap in the approximabilty for this problem.

1.1 Preliminaries .

DEFINITION 1.1. *An instance of the traveling repairman problem (TRP) is given by a set V of $n+1$ points, one of them is the origin r, and symmetric integer distances d_{ij} satisfying the triangle inequality. A solution is a permutation $\Pi = (v_0, v_1, \ldots, v_n)$ of V, where $v_0 = r$. The completion time $C(v)$ of a point $v = v_i$ is the distance from r to v on the path defined by Π: $C(v_i) = \sum_{j=1}^{i} d_{v_{i-1},v_i}$. The goal is to find a solution with minimum total completion time: $\sum_{v\in V} C(v)$.*

With loss of a $(1+\epsilon)$-factor in the approximation factor, we may assume that all distances are polynomially bounded. Consequently, the length of the optimal tour is polynomially bounded. More precisely, (see also [8]) we may assume that all distances are in $\{0, 1, \ldots, B\}$ with $B = O(n^2/\epsilon)$. We use the notation $\tilde{O}(.)$ when ϵ is assumed constant. For example, $\epsilon n = \tilde{O}(n)$ and $n^{1/\epsilon^2} = n^{\tilde{O}(1)}$.

It is convenient for the analysis to see a solution Π as a path that is traversed with at most unit speed. That means, we assume there is a continues path of length d_{v_{i-1},v_i} between consecutive points v_{i-1} and v_i in Π. The completion time $C(v)$ of input point v is then defined as the time at which v is visited for the first time.

In [12], the authors note that any solution Π can be replaced by a concatenation of $\gamma = O(\log n/\epsilon)$ TSP-paths with only a $(1+\epsilon)$ factor increase in value. That means, the solution can be partitioned into γ segments such that replacing each segment S by a shortest path that visits the same points as S and has the same start and endpoint as S, increases the value of the solution by at most a factor $(1+\epsilon)$. The proof follows easily by letting the number of points visited by the segments decrease geometrically. Here, we prove the same lemma through the alternative approach of partitioning the timeline in intervals of geometrically increasing length. We shall not use Lemma 1.1 directly but will use a similar argument later when we partition the timeline in only a *constant number* of intervals.

LEMMA 1.1. *With loss of a factor $1+\epsilon$ in the approximation, we may assume that* OPT *is a concatenation of $O\left(\frac{\log n}{\epsilon}\right)$ TSP-paths.([12])*

Proof. Consider time-points $1, (1+\epsilon), (1+\epsilon)^2, \ldots, (1+\epsilon)^\gamma$, where $(1+\epsilon)^\gamma = O(n^3/\epsilon)$ is an upper bound on the length of the optimal tour. Then, $\gamma = O((\log n)/\epsilon)$. Now, replace the path between any two consecutive time-points by a TSP-path. The completion time of any point is increased by at most a factor $1+\epsilon$. □

By Lemma 1.1, it is enough to restrict to solutions composed of $\gamma = O((\log n)/\epsilon)$ TSP-paths. In [12], a solution composed of at most γ TSP-paths is found by one dynamic program. Consequently, the $\log n$ shows up naturally in the exponent of the running time. A simple example on the line shows that $\Omega(\log n)$ paths are needed for a PTAS: Let $n = 2^k - 1$ and place 2^{k-i} points at $x = (-2)^i$, for $i = 1, \ldots, k$. For this

example, there is no constant approximate solution that is a concatenation of $o(\log n)$ TSP-paths. Hence, if we stick with the TSP-paths approach then the only way to improve on the running time is to have a better understanding of the dependency between the paths. The key inside in our approach is that the TSP-paths can be clustered in groups of K consecutive TSP-paths each, where K is a constant which depends on ϵ only, and such that there is only very limited dependency between the groups. That means the problem on $O(\log n)$ TSP paths basically reduces to a problem on K TSP paths. Consequently, known TSP algorithms can be modified for these subproblems on K TSP-paths. The dependency is limited in the sense that it is enough to solve only a polynomial number of these subproblems. Then, dynamic programming is used to combine solutions for subproblems into one tour.

1.2 Segmented TSP What we shall denote as the segmented TSP is a generalization of the known k-TSP in which one needs to find, for a given TSP-instance and number $k \leqslant n$, a tour of minimal length that visits at least k points. A $(2+\epsilon)$-approximation was given by Arora and Karakostas [11]. The problem can be solved exactly on a tree metric and a PTAS is known for Euclidean spaces of fixed dimension [7]. For our PTAS, we need a more general problem that we denote by segmented TSP. It corresponds with the k-TSP problem for $K = 1$.

DEFINITION 1.2. *An instance of segmented TSP is given by a set V of $n+1$ points, one of them is the origin, and symmetric integer distances d_{ij} satisfying the triangle inequality. Also given are numbers $l_1 \leqslant l_2 \leqslant \cdots \leqslant l_K$ and numbers $n_1 \leqslant n_2 \leqslant \cdots \leqslant n_K$. A solution is a tour that starts and ends in the origin such that at least n_h vertices are visited within the first l_h distance for all $h \in \{1, \ldots, K\}$ and such that the length of the tour is at most l_K. We say that an algorithm solves the problem if it always finds a solution if one exists. We say that an algorithm is an α-approximation ($\alpha \geqslant 1$) if for any feasible instance it finds a tour that visits at least n_h vertices within path-distance αl_h for all h and such that its length is at most αl_K.*

NB. One may also consider the segmented TSP without the restriction that the solution must end in the origin. This restriction is convenient for our purpose.

THEOREM 1.1. *If, for any metric space, there is a polynomial time α-approximation algorithm for the segmented TSP for every constant number of segments, then there is a polynomial time $\alpha(1+\epsilon)$-approximation algorithm for the Traveling Repairman Problem in the same metric space for every constant ϵ.*

In Section 3, we show that the segmented TSP with a constant number of segments can be solved exactly for weighted trees. Further, we show that there is a PTAS for the Euclidean plane and for weighted planar graphs.

COROLLARY 1.1. *There exists a PTAS for the (unweighted) Traveling Repairman Problem in the Euclidean plane, for edge-weighted trees, and for edge-weighted planar graphs.*

The following useful definition and lemma apply to the segmented TSP in general and are used in Section 2.

DEFINITION 1.3. *Let I be a segmented TSP instance. The j-th completion time of I is denoted by C_j^I and is defined as follows. The first n_1 completion times are l_1, the next $n_2 - n_1$ completion times are l_2, and so on.*

Note that C_j^I is an upper bound on the j-th completion time in any feasible solution for I. The following lemma is immediate.

LEMMA 1.2. *Let $\mathcal{T}$ be a α-approximate solution for segmented TSP instance I and denote the j-th completion time in $\mathcal{T}$ by $C_j^{\mathcal{T}}$. Then, $C_j^{\mathcal{T}} \leqslant \alpha C_j^I$ for any j.*

2 Reducing TRP to segmented TSP

The reduction is done by the following steps. First, it is shown that we may restrict to solutions that return in the origin at time points t_i, where $t_i/t_{i-1} = (1+\epsilon)^K$ for some large K depending on ϵ only. For this, we use a simple probabilistic argument. The part of the tour between time points t_i and t_{i+1} is called the i-th. *subtour*. Each of these $\Gamma = \tilde{O}(\log n)$ subtours can be partitioned into K subpaths where the ratio of end time and start time of each path is $1+\epsilon$. We call these subpaths *segments*. In the optimization, we may approximate the completion time of a point by the endpoint of the segment that it is on. If we would know for each subtour the points to be visited, then an approximate solution can easily be computed given a segmented TSP algorithm. Clearly, we cannot afford to guess these subsets. However, as we show in this section, for large enough K, we can afford to revisit in subtour i, all points that were visited in the preceding subtours. Consequently, in the dynamic programming there is no need to keep track of *subsets* of points and we only need to enumerate over the *number* of points visited. This requires only a polynomial number of segmented TSP instances to solve. Remarkably, the number Γ of subtours is not dominating the running time, which is only polynomial in Γ.

2.1 Restricting the solution space Assume $0 < \epsilon \leqslant 1$ and let K be an integer depending on ϵ only.

To simplify notation, we write $\delta = 1 + \epsilon$. Choose h_0 uniformly at random from $\{0, 1, \ldots, K-1\}$ and let

$$A_i = \delta^{(i-1)K+h_0}, \text{ for } i \geqslant 0. \tag{2.1}$$

Consider an optimal solution, OPT, and let L be its length, i.e., the largest completion time. (In general, we denote by OPT the solution itself as well as its value.) For $i \geqslant 1$, let OPT_i be the solution restricted to the first length A_i. (Note in particular that OPT_1 has length δ^{h_0}.) Let Γ be the smallest integer such that $A_\Gamma \geqslant L$. Hence, we may assume[1] $\Gamma = \tilde{O}(\log n)$. The modified solution OPT$'$ is defined as follows:

OPT$'$: For $i = 1$ to Γ, start OPT_i at time $t_i := 3A_{i-1}$ and return to the origin.

(The constant 3 above may be replaced by any constant strictly larger than 2 for the proof to work.) Let v be an arbitrary point of the instance and let $C(v)$ and $C'(v)$ be its completion time in, respectively, OPT and OPT$'$, where the completion time is the first moment that the point is visited. Let $\mathbb{E}[\text{OPT}']$ be the expected value of OPT$'$ over the random choice of h_0.

LEMMA 2.1. *For large enough $K = O(1/\epsilon^2)$, it holds that $\mathbb{E}[C'(v)] \leqslant (1+\epsilon)C(v)$ for any input point v. Hence, $\mathbb{E}[\text{OPT}'] \leqslant (1+\epsilon)\text{OPT}$.*

Proof. Feasibility holds if it is possible to return to the origin after each OPT_i before beginning the next path OPT_{i+1} at time t_{i+1}. This is clearly true if $t_i + 2A_i \leqslant t_{i+1}$ for all i. Since $t_i = 3A_{i-1}$, this is equivalent with

$$3A_{i-1} + 2A_i \leqslant 3A_i \;\Leftrightarrow A_i/A_{i-1} \geqslant 3 \Leftrightarrow \delta^K \geqslant 3.$$

Hence, for feasibility it is enough to take $K = O(1/\log\delta) = O(1/\epsilon)$. Now, let us compute the expected value of OPT$'$. Consider an arbitrary point v of the instance and let i' be the smallest index such that $A_{i'} \geqslant C(v)$, that means, point v is visited in OPT$'$ for the first time by path $\text{OPT}_{i'}$. Let $\delta^{q-1} < C(v) \leqslant \delta^q$, for some integer $q \geqslant 0$. (Note that $q \geqslant 0$ since the minimum distance and hence the minimum completion time is at least 1.) Then the expected value of $A_{i'}$ is

$$\mathbb{E}[A_{i'}] = \frac{1}{K}\sum_{h=0}^{K-1}\delta^{q+h} = \frac{\delta^{q+K}-\delta^q}{K(\delta-1)} < \frac{\delta^{q+K}}{K(\delta-1)}.$$

[1] More precisely, we have $A_\Gamma \geqslant L = O(n^3/\epsilon)$, where $A_\Gamma \geq (1+\epsilon)^{(\Gamma-1)K}$. Hence, $\Gamma K = O(\log_{1+\epsilon}(n^3/\epsilon)) \Rightarrow \Gamma = O(\log n/(\epsilon K) = O(\epsilon \log n)$ if we take $K = \Theta(1/\epsilon^2)$.

Remember that $t_{i'} = 3A_{i'-1} = 3\delta^{-K}A_{i'}$ and note that $t_{i'} = C'(v) - C(v)$. Hence,

$$\begin{aligned}\mathbb{E}[C(v')] - C(v) &= \mathbb{E}[t_{i'}] = \frac{3}{\delta^K}\mathbb{E}[A_{i'}] < \frac{3}{\delta^K}\frac{\delta^{q+K}}{(\delta-1)K} \\ &= \frac{3\delta^q}{(\delta-1)K} < \frac{3\delta}{(\delta-1)K}C(v).\end{aligned}$$

It follows that $\mathbb{E}[C'(v)] \leqslant (1+\epsilon)C(v)$, for any point v if $\frac{3\delta}{(\delta-1)K} \leqslant \epsilon$, i.e., if

$$K \geqslant \frac{3\delta}{\epsilon(\delta-1)} = \frac{3(1+\epsilon)}{\epsilon^2} = O\left(\frac{1}{\epsilon^2}\right).$$

□

Since $\mathbb{E}[\text{OPT}'] \leqslant (1+\epsilon)\text{OPT}$ there must be some h_0 for which the corresponding deterministic solution OPT$'$ satisfies $\text{OPT}' \leqslant (1+\epsilon)\text{OPT}$. From now on we consider OPT$'$ to be this deterministic solution. For $j = 1, \ldots, n$, let D_j be the j-th completion time of OPT$'$ (where we only consider the first appearance of each point). Equivalently, we can define D_j as the completion time of the j-th point on the first subtour that visits at least j points. The properties of solution OPT$'$ are listed in the next lemma.

LEMMA 2.2. *Solution OPT$'$ has the properties:*

(i) It is in the origin at time $t_i = 3\delta^{(i-2)K+h_0}$ for all $i = 1, 2, \ldots, \Gamma$, where $K = O\left(\frac{1}{\epsilon^2}\right)$, $\Gamma = O(\epsilon \log n)$ and h_0 is some fixed number in $\{0, 1, \ldots, K-1\}$. We call the tour between t_i and t_{i+1} the i-th subtour.

(ii) The number of points on the i th. subtour is non-decreasing in i.

(iii) For $j = 1, \ldots, n$, let D_j be the completion time of the j-th point on the first subtour that visits at least j points. Then $\sum_{j=1}^n D_j \leqslant (1+\epsilon)\text{OPT}$.

Now consider any tour that satisfies properties (i) and (ii) and let D_j be defined as in (iii). Then, clearly the j-th completion time is no more than D_j. Hence, the lemma shows that we may restrict to solutions which have properties (*i*) and (*ii*) and among those tours minimize $\sum_{j=1}^n D_j$ as defined in (*iii*). We shall prove that minimizing $\sum_{j=1}^n D_j$ can be done easily by dynamic programming if we have an algorithm for the following subproblem.

2.2 The subproblem

DEFINITION 2.1. *An instance of the subproblem is given by $i \in \{1, \ldots, \Gamma\}$ and numbers $m' \leqslant m'' \in$*

$\{0, 1, \ldots, n\}$. *A solution is a tour that starts at the origin at time t_i and returns before time t_{i+1} and visits exactly m'' points. The value of a solution is the sum of completion times of points $m'+1, \ldots, m''$ on this tour (which is zero if $m' = m''$). The objective is to find a solution with minimum value. Note that an instance i, m', m'' may not have a feasible solution. For any feasible instance, let* $\textsc{Sub}_i(m', m'')$ *be its optimal value.*

Let m_i be the number of points visited by the partial solution $\textsc{Opt}_i$. Then clearly,

$$\textsc{Opt}' \geqslant \sum_{i=1}^{\Gamma} \textsc{Sub}_i(m_{i-1}, m_i).$$

Definition 2.2. *An (α, β) approximation algorithm for the subproblem is an algorithm that finds for any* feasible *instance (i, m', m'') a tour that starts in the origin at time αt_i and ends in the origin before time αt_{i+1}, visits exactly m'' points, and for which the total completion time of the points $m'+1, \ldots, m''$ is at most $\alpha\beta\textsc{Sub}_i(m', m'')$.*

Assume we have an (α, β)-approximation algorithm Alg for the subproblem. Let $\textsc{Alg}_i(m', m'')$ be the value returned by the algorithm for instance (i, m', m'') and let it be infinite if no solution was found. For any sequence of integers $0 \leq \hat{m}_1 \leqslant \cdots \leq \hat{m}_\Gamma = n$ we get a tour of total completion time

$$\sum_{i=1}^{\Gamma} \textsc{Alg}_i(\hat{m}_{i-1}, \hat{m}_i) \leqslant \alpha\beta \sum_{i=1}^{\Gamma} \textsc{Sub}_i(\hat{m}_{i-1}, \hat{m}_i) \tag{2.2}$$

by concatenating the tours $\textsc{Alg}_i(\hat{m}_{i-1}, \hat{m}_i)$. Minimizing the left side of (2.2) over all values $0 \leq \hat{m}_1 \leqslant \cdots \leq \hat{m}_\Gamma = n$ is easy since they form a non-decreasing sequence. To be precise, let $\textsc{Alg}_1(m'') = \textsc{Alg}_1(0, m'')$ for all $m'' \leqslant n$ and for $k = 2, \ldots, \Gamma$, let

$$\textsc{Alg}_k(m'') = \min_{m' \leqslant m''} \textsc{Alg}_{k-1}(m') + \textsc{Alg}_k(m', m'').$$

Then, the minimum is given by $\textsc{Alg}_\Gamma(n)$. Let the values $\hat{m}_i$ minimize the left side of (2.2) and let m_i be the number of points visited by the partial solution $\textsc{Opt}_i$. Then, we find a solution of total completion time at most

$$\begin{aligned} \alpha\beta \sum_{i=1}^{\Gamma} \textsc{Sub}_i(\hat{m}_{i-1}, \hat{m}_i) &\leq \alpha\beta \sum_{i=1}^{\Gamma} \textsc{Sub}_i(m_{i-1}, m_i) \\ &\leqslant \alpha\beta\textsc{Opt}' \leq \alpha\beta(1+\epsilon)\textsc{Opt}. \end{aligned}$$

The number of subproblems is $O(\Gamma n^2)$ and given all approximate values, the dynamic programming takes $O(\Gamma n^2)$ time. Further, the number of choices for h_0 is K (See Equation 2.1). Hence, it takes only $O(K\Gamma n^2) = \tilde{O}(n^2 \log n)$ calls to the approximation algorithm for the subproblem to get an $\alpha\beta(1+\epsilon)$-approximation for the Traveling Repairman Problem.

Approximating the subproblem. We show how to get an $(\alpha, 1+\epsilon)$-approximation for the subproblem if we have an α-approximation algorithm for the segmented TSP.

Let (i, m', m'') be a feasible instance of the subproblem. For $h = 0 \ldots K$, define *time-point*

$$t_i^{(h)} = (1+\epsilon)^h t_i, \text{ (Hence, } t_i^{(K)} = t_{i+1}^{(0)} = t_{i+1}.) \tag{2.3}$$

Recall the definition of the segmented TSP problem. A polynomial number of segmented TSP instances is solved (approximately). Let $l_h = t_i^{(h)} - t_i$, $h = 1, \ldots, K$ and hence, these are fixed given the index i. The numbers n_h take all possible integer values for which $n_1 \leqslant n_2 \leqslant \cdots \leqslant n_K = m''$. This gives $O(n^K)$ instances. Solve all these instances by some α-approximate segmented TSP algorithm and determine the solution with smallest total completion time of the points $m'+1, \ldots, m''$. Let T be this solution and let T' be the solution T started at time αt_i. We show that T' is an $(\alpha, 1+\epsilon)$-approximation for the subproblem (i, m', m'').

The length of T is at most $\alpha(t_{i+1} - t_i)$. Hence, T' completes before time αt_{i+1}. Also, it visits exactly m'' points. Now let $\textsc{Alg}_i(m_{i-1}, m_i)$ be the value of T' for subproblem (i, m', m''). Consider an optimal solution Π for subproblem (i, m', m'') and let I be the segmented TSP instance given by the numbers n_h, where n_h is the number of points visited by Π until time $t_i^{(h)}$. Let C_j^Π be the j-th completion time in Π and let C_j^I be the j-th completion time of I_Π as defined in Definition 1.3. Then,

$$(1+\epsilon)C_j^\Pi \geqslant (t_i + C_j^I).$$

Instance I is among the enumerated instance. Hence, using Lemma 1.2,

$$\begin{aligned} \textsc{Alg}_i(m_{i-1}, m_i) &\leqslant (m'' - m')\alpha t_i + \alpha \sum_{j=m'+1}^{m''} C_j^I \\ &= \alpha \sum_{j=m'+1}^{m''} (t_i + C_j^I) \\ &\leqslant \alpha \sum_{j=m'+1}^{m''} (1+\epsilon) C_j^\Pi \\ &= (1+\epsilon)\alpha\textsc{Sub}_i(m_{i-1}, m_i). \end{aligned}$$

Running time. For each subproblem, $O(n^K)$ segmented TSP instances are solved and we simply store the best one. There are $\tilde{O}(n^2 \log n)$ instances for the subproblem and the dynamic program runs in $\tilde{O}(n^2 \log n)$ time. Hence, the total running time is $n^{O(K)} = n^{O(1/\epsilon^2)}$ multiplied by the running time of the α-approximation algorithm for the segmented TSP.

3 Approximating the segmented TSP

By Theorem 1.1, any α-approximation algorithm for segmented TSP implies a $(1+\epsilon)\alpha$-approximation algorithm for the Traveling Repairman Problem in the same metric space. Here, we consider the approximability of segmented TSP in different metric spaces. Remember the definition of an α-approximation algorithm for the segmented TSP problem: It finds a solution such that n_i points are visited before time αl_i, where n_i and l_i are given. (For ease of notation we use an index i instead of h as used in the previous section.) When, we assume that the number of segments K is constant, then we may guess the number of points visited on each of the segments. More precisely, we denote by segment i, the path that runs between distance l_{i-1} (excluded) and l_i (included). We guess the numbers μ_i of points visited on segment i, where $n_i = \sum_{j=1}^{i} \mu_j$ for all i. The number of choices is only $O(n^K)$, which is polynomial if K is a constant. Further, we assume that all l_i are integer and denote $\lambda_i = l_i - l_{i-1}$. Hence, from now, we assume that the segmented TSP instance is given by numbers λ_i and μ_i $(i = 1, \ldots, K)$ and we need to visit *exactly* μ_i points on the i-th segment.

N.B. By 'guessing' we mean enumerating over all possible values and we say that we are *able to guess* a certain value if the number of possible values is polynomialy bounded.

3.1 Edge-weighted tree The metric space is given by a tree T with non-negative integer weights on the edges. The distance between any two points u, v is the length of the unique path between u and v on T. The TSP is trivial on trees since a tour is optimal if and only if it is a depth-first search on T. Also, the k-TSP can easily be solved by dynamic programming: For each vertex v and number $j \leqslant k$, store the length $l(v, i)$ of the shortest tour in the subtree rooted at v which visits exactly j vertices. The value is easily computed from the table of values of the children of v.

The generalization to segmented TSP is straightforward. First, turn the tree into a rooted binary tree such that only leaves need to be visited. This can be done with only a constant factor increase in the number of points by adding edges of length zero. For each node v unequal to the root we define a *vector of crossing information* as follows. The edge above v is traversed at most $2K$ times. This gives at most K subtours in the tree rooted at v which start and end at v. For each of these we guess the start time and end time and we guess the number of points that each of the K segments have on this subtour. For all possible vectors we only store if this is feasible or not. A vector is feasible if it can be obtained from feasible vectors of its two children. For any leaf, a vector is feasible if there is exactly one subtour and the start time equals its end time and it contains exactly one vertex (namely v). Note that the time of visit determines the segment that v is on. For the root we only consider the case of one subtour starting at time 0 and ending at time $l_K = \sum_{i=1}^{K} \mu_i$ and for which segment i contains exactly μ_i points. The running time is $n^{O(K)}$.

3.2 Euclidean plane We show that for any feasible segmented TSP instance we can find a $(1+\epsilon)$-approximate solution in time $n^{O(K)}$. That means, the solution is a concatenation of K paths, where the i-th path has length at most $(1+\epsilon)\lambda_i$ and visits exactly μ_i points. It is important to note that the ϵ used in this section has nothing to do with the ϵ of Section 2. That means, in this section, K is an arbitrary integer constant.

Arora and Karakostas [12] give a quasi polynomial time approximation scheme for the Traveling Repairman Problem in the Euclidean plane. (See also [8].) The algorithm in [12] is based on the refined TSP-PTAS [7], which is more efficient than the simpler version that was published earlier [6]. In the latter paper, it was shown that here is a $(1+\epsilon)$-approximate TSP tour that crosses the boundary of each square in the quadtree only $O(\log n/\epsilon)$ times. In the refined PTAS, it was proven that $O(1/\epsilon)$ crossings satisfy too. In combination with Lemma 1.1 this led the authors of [12] to a TRP algorithm with $n^{O(\log n/\epsilon^2)}$ running time. The proof contains many details but intuitively it does follow easily from the next three observations: (i) all lengths are polynomially bounded, (ii) the solution is composed of $O(\log n/\epsilon)$ TSP-paths, and (iii) there are only $O(1/\epsilon)$ crossings per square per TSP-path. Hence, for a given square we can afford to guess for each crossing basically all information that we want and still end up with quasi-polynomial running time. In the segmented TSP problem, the solution is composed of only K TSP paths. Hence, for constant K we should expect a better running time. A minor issue is that we have a restriction on the length of each of the K segments. This is easily solved by using Markov's inequality, as we show below in the

discussion of the structure theorem. Another issue is that we cannot copy the approach for the Euclidean case to the planar graph case, as was done in [12]. For planar graphs, the role of square boundaries is played by the Jordan curves, and the structural theorem in this case states that there is a $(1+\epsilon)$-approximate TSP-tour that crosses the curve $O(\log n/\epsilon^2)$ times, and not $O(1/\epsilon)$ times, as for the Euclidean case. This leads to a quasi-polynomial running time for the planar segmented TSP. However, we will show a PTAS for Euclidean segmented TSP which applies even if we adopt the simpler TSP PTAS [6] that allows $O(\log n/\epsilon)$ crossings of the dissection squares. Then, the same approach implies a PTAS for weighted planar graphs.

The TSP-PTAS [6, 7, 8] contains numerous details. Here we only address those that are of interest for our modification and refer to the survey [8] for omitted details.

Structure theorem The rounding of the instance and the construction of the quadtree and portals remains basically the same: Take the smallest bounding box and define a grid of polynomial dimension. Move input points to the middle of grid cells. Then, place an enclosing box of double side length at random on top of it. Next, make the dissection tree. The depth is $\tilde{O}(\log n)$. We let the number of portals for each dissection square be $O(\log n/\epsilon)$. By scaling distances, we may assume that for each grid cell and segment i, the part of the segment that lies inside the cell has integer length.

For the Euclidean TSP problem it is known [8] that there is a tour Π that crosses the boundary of each dissection square only in portals, and at most twice in each portal, and for which the expected length is at most $(1+\epsilon)$ times optimal. The same is true for the Traveling Salesman Path problem [8]. The expectation is over the random shift of the enclosing box. More precisely, for any path of length S in the bounding box, the expected length of the detour that is needed to make it portal respecting is ϵS. Now consider a feasible segmented TSP instance given by numbers λ_i and μ_i. It follows directly that there is solution $\mathcal{T}$ that crosses only at portals and each portal at most $2K$ times such that each segment i visits μ_i points and has length $L_i \geqslant \lambda_i$ and such that $\mathbb{E}[L_i - \lambda_i] \leqslant \epsilon\lambda_i$. Again, the expectation is over the random shift of the box. Note that this is not enough for our purpose since we want each of the K differences to be at most $\epsilon\lambda_i$ simultaneously. Since K is constant, this is easily solved by Markov's inequality: $\Pr[L_i - \lambda_i \geqslant 2K\epsilon\lambda_i] \leqslant 1/(2K)$ for each i. Then, by the union bound, $\Pr[L_i - \lambda_i \geqslant 2K\epsilon\lambda_i$ for at least one $i] \leqslant 1/2$. Hence, in stead of an expected $(1+\epsilon)$-approximate solution we get a $(1+2K\epsilon)$-approximate solution with probability at least $1/2$. The additional factor $2K$ is no issue since K is a constant. (Again, remember that K is an absolute constant independent of ϵ in this section.)

Dynamic Programming Note that in the dynamic programming we do not solve an optimization problem but only search for a feasible solution. An instance I of a subproblem in the DP is given by:

(1) A dissection square S.

(2) For each segment i the number of points and the length of segment i inside S.

(3) For each portal of S, and all segments i, the number of times segment i crosses it (0,1,or 2) and in which direction (in or out).

(4) For each segment i, the first and last crossing with S are specified.

(5) A pairing of the crossings with S.

Note that we only guess the length and number of points for each segment and not for each crossing as was done in [12]. Hence, we can afford $\tilde{O}(\log n)$ crossings. Clearly, the number of choices for items (1)–(4) is $n^{O(K/\epsilon)}$. The pairing of the crossings can be done almost independently for each segment since we know for each crossing the segment it belongs to and we know the first and last crossing of each segment. Hence, the number of pairings is bounded by $2^{O(\log n/\epsilon)K} = n^{O(K/\epsilon)}$.

First, consider the base case. By the rounding step, all points coincide and are in the middle of the cell. Clearly, it would be optimal to serve all these by the same segment. However, we assumed the number of points on each segment i to be given by μ_i. Hence, we should allow the midpoint to be visited by multiple segments. Clearly, each segment needs to cross the midpoint at most once. Feasibility can be checked in $O(Km)$ time, where $m = O(\log n/\epsilon)$ is the number of portals per square. For the smallest dissection square containing the root vertex we have the additional restriction that segment 1 starts in the root and segment K ends in the root.

Consider an arbitrary (non-base) instance I given by (1)–(5). We check if there is a feasible instance for each of its children which together are consistent with instance I. That means, the number of points and lengths should add up to the right value and all crossing and pairings should be consistent. Further, one needs to exclude combinations that form subtours. For each instance I there are $n^{O(K/\epsilon)}$ combinations of instances for its four children to check. The time for checking a single combination is only linear in the number of portals. For the largest square (the enclosing

box) we only need to verify one instance: the $(1+\epsilon)$-approximation, i.e., there are no crossings and segment i has length $(1+\epsilon)\lambda_i$ and contains exactly μ_i points.

3.3 Planar graph A PTAS for the TSP on weighted planar graphs was given by Arora et al. [9]. The separator in this case is a Jordan curves that divides the graph into an exterior and interior part. The number of portals is $m = O(\log n/\epsilon^2)$ and each portal is crossed at most twice. In [12] the authors note that their QPTAS for Euclidean TRP carries over directly to weighted planar graphs. The running time for the QPTAS is $n^{O(\log^2 n/\epsilon^3)}$ since there are $\log n/\epsilon^2$ crossing for each of the $\log n/\epsilon$ segments and the DP in [12] makes expensive guesses (about length and number of points) for each crossing. Theorem 1.1 reduces the number of segments to a constant. Moreover, the DP used in Section 3.2 only needs to make expensive guesses for each segment and not for each crossing. Hence, we conclude that a PTAS similar to the Euclidean PTAS works for planar graphs too. The total running time for planar graphs is $n^{O(K/\epsilon^2)}$.

4 Generalizations and variants

The approximation schemes for TRP in $\mathbb{R}^2$, weighted trees and planar graphs apply as well if *release dates* are added. The transformation from Opt to Opt$'$ works still fine in that case since the solution is only moved forward in time. Hence, Opt$'$ is feasible and the total completion time is increased by at most a factor $1+\epsilon$. In the reduction to segmented TSP, we need to consider segmented TSP instances with release dates. By rounding release times (by at most a factor $1+\epsilon$) we may assume that points are released only at the start times of the K segments. Equivalently, we may assume that we have sets $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_K$ of points such that the j-th segment can only visit points from S_j. In the dynamic programs, except for the base case, we do not consider which points are visited but only store the number for each of the segments. The base case can still be efficiently solved since the number of segments is constant.

The PTAS applies as well if our objective function is a linear combination of total completion time, $\sum_j C_j$, and the length of the path. That means, the problem is to find a path, starting in the origin, that minimizes $\alpha \sum_j C_j + \beta \max_j C_j$ for some $\alpha, \beta \geqslant 0$. To see this, define the path Opt$'$ in exactly the same way. For any input point p we have $\mathbb{E}[C'(p)] \leqslant (1+\epsilon)C(p)$. In particular, this applies to the last point on the path. Hence, $\mathbb{E}[\max_j C'_j] \leqslant (1+\epsilon)\max_j C_j$, where C_j ($C_{j'}$) is the j-th completion time in Opt (Opt$'$). In total we get that

$$\mathbb{E}[\alpha \sum_j C'_j + \beta \max_j C'_j] \leqslant (1+\epsilon)(\alpha \sum_j C_j + \beta \max_j C_j).$$

In the algorithm we guess h_0 for which the inequality above holds without expectation. Also, we guess the corresponding length L' of the tour Opt$'$. Then we apply the same DP but we restrict to tours of length at most $(1+\epsilon)L'$.

In the *Randomized Search Ratio* problem one has to find a (random) path starting from the root r and visiting all points and the goal is to minimize $\max_v \mathbb{E}[C(v)]/d(r,v)$, where $d(r,v)$ is the distance from r to v . In [12], the authors mention that E. Tardos observed the following: If the minimum latency problem has a PTAS for a certain class of metrics, then the randomized search ratio problem has an approximation scheme for that same class of metrics. Thus, our PTAS implies a PTAS for the randomized search ratio for trees, planar graphs and the Euclidean plane.

The PTAS also applies to the *The k-repairman problem* in which one needs to find k repairman paths that together visit all points. The transition from Opt to Opt$'$ is the same: All repairman are in the origin at the same time. In the segmented TSP we need to find k segmented TSP-paths simultaneously. For constant k, there is only a polynomial increase in the running time.

Open problems The generalization to weighted completion times is straightforward if weights are polynomially bounded. However, for general weights it is not clear how to adjust the approximation scheme.

Another interesting problem that is closely related is that of finding a metric embedding on a line such that the average distortion is minimized [16]. One can show that the k-TRP with $k = 2$ is a special case of this metric embedding problem. The authors of [16] use ideas of the QPTAS for the traveling repairman problem to obtain a QPTAS for the average distortion problem. It is not clear whether our ideas can be used to obtain a PTAS for metric line-embedding as well.

5 Single machine scheduling under precedence constraints

The reduction used for the TRP applies to almost any problem of minimizing the total (weighted) completion, assuming that weights are polynomially bounded. Of course, this doesn't mean that it is always useful since the subproblem may be harder to approximate than the original. First, we give a rough sketch how to apply it to the simple scheduling problem $1|r_j|\sum C_j$ and then give a detailed proof for the more challenging problem of scheduling under precedence constraints. A PTAS for the first was given by Afrati et al [1].

Example:$1|r_j|\sum C_j$. We have a single machine and n jobs with processing times p_j and release times r_j for $j = 1, \ldots, n$. The objective is to find a schedule that minimizes the total completion time $\sum_j C_j$, where C_j is the completion time of job j. Now, the subproblem is defined on an interval from t_i to $(1+\epsilon)^K t_i$ for some i, and where K depends on ϵ only. For given $n' \leqslant n''$, the problem is to find a feasible schedule on a subset of the jobs that minimizes $\sum_{j=n'+1}^{n''} C_j$. Now partition the interval in K subintervals as before where the ratio of start and end time of a subinterval is $1+\epsilon$. Hence, we may assume that jobs are released only at the beginning of subintervals. Say that a job is large if its processing time is more than ϵ times the length of the smallest subinterval (which is the first). Then, the number of large jobs in the optimal solution to the subproblem is bounded by a constant and we guess all of them. The small jobs can be added greedily such that each subinterval is overpacked by at most ϵ times its length. □

One of the most intriguing scheduling problems is that of minimizing total weighted completion times on a single machine under precedence constraints. ($1|prec|\sum_j w_j C_j$, in the notation by Graham et al [20].) The problem is known to be $\mathcal{NP}$-hard [24, 25] and several 2-approximation algorithms are known. The paper by Ambühl et al. [3] gives a recent overview on the status of this problem. Exact polynomial time algorithms are known for some special cases, e.g., for series parallel posets [24]. Surprisingly, interval ordered precedence constraints are not one of these. Woeginger [30] gave a 1.62-approximation algorithm and a 3/2-approximation was given by Ambühl et al [3]. The same paper shows that scheduling interval orders is in fact $\mathcal{NP}$-hard. Here, we give a polynomial time approximation scheme for interval ordered precedence constraints.

An instance of the scheduling problem is given by n jobs to be processed on a single machine that can process at most one job at a time. Each job j has a nonnegative integer processing time p_j and weight w_j. A partial order on the jobs defines the precedence constraints between jobs. That means, if $j_1 \prec j_2$, then job j_1 must be completed before j_2 can start. The goal is to find a non-preemptive schedule that minimizes $\sum_{j=1}^{n} w_j C_j$, where C_j is the completion time of job j.

DEFINITION 5.1. *A partial order on a set J is an interval order if there is a function that assigns to each $j \in J$ a closed interval $[l_j, r_j]$ such that $j_1 \prec j_2$ if and only if $r_{j_1} < l_{j_2}$. It is easy to see that for any interval order there is a corresponding set of intervals for which all $2|J|$ endpoints are different.*

A theorem by Woeginger [30] states that for general precedence constraints, we may restrict our approximation analysis to the case $1 \leqslant p_j \leqslant n^2$ and $1 \leqslant w_j \leqslant n^2$, where n is the number of jobs. In fact, this theorem can be applied to the special case of interval orders since its proof only reverses the precedence constraints, and since the reverse of an interval order is again an interval order (see [30]).

5.1 Reducing the problem The reduction is almost the same as what we did for the TRP problem. Let $K = O(1/\epsilon^2)$ and choose h_0 uniformly at random from $\{0, 1, \ldots, K-1\}$. The numbers A_i are as before. Consider an optimal solution, OPT, and let OPT_i be the solution restricted to the jobs that complete not later than A_i, for $i = 1, 2, \ldots$. The schedule OPT′ is defined by simply concatenating all the solutions OPT_i. Note that jobs appear multiple times since any job that appears in OPT_i appears as well in $\text{OPT}_{i'}$ for all $i' \geqslant i$. In general, we allow jobs to appear more than once and call these *pseudo schedules*. The completion times and precedence constraints apply only to the first appearance of each job.

As before, denote by t_i the time at which OPT_i starts in OPT′. The solution OPT′ is well-defined if $t_i \geqslant t_{i-1} + A_{i-1}$ for all i. Let $t_i = cA_{i-1}$, then $cA_{i-1} \geqslant cA_{i-2} + A_{i-1}$ holds if $c/(c-1) \geqslant A_{i-1}/A_{i-2}$. For any constant $c > 1$ we can choose K such that $c/(c-1) \geqslant A_{i-1}/A_{i-2} = \delta^K$. For simplicity, let us just take $t_i = 3A_{i-1}$ as before. This creates unnecessary idle time but at least we can blindly copy the analysis of the TRP. Let C_j (C'_j) be completion time of job j in OPT (OPT′). Then, following the proof of Lemma 2.1, we have for any job j that

$$\mathbb{E}[C'_j] \leqslant (1+\epsilon) C_j,$$

where the expectation is over the random choice of h_0. Taking the weighted sum we have

$$\mathbb{E}[\text{OPT}'] = \mathbb{E}[\sum_j w_j C'_j] \leqslant (1+\epsilon) \sum_j w_j C_j = (1+\epsilon)\text{OPT}.$$

From now assume that h_0 is chosen such that the inequality holds without expectation: $\text{OPT}' \leqslant (1+\epsilon)\text{OPT}$.

We call the schedule between two consecutive time points t_i a *subschedule*. Note that in OPT′, each subschedule is a feasible schedule on its own. The total weight of jobs in the i-th subschedule is the weight completed by OPT_i and hence, is non-decreasing in i. Let $W = \sum_j w_j$. Then, $W \leqslant n^3$, since $w_j \leqslant n^2$ for all j. For any $w \in \{1, 2, \ldots, W\}$, let D^w be the first moment at which OPT′ completes a total weight of at

least w (where for any job we only count the weight of its first appearance and the weight is only counted when the job completes.) Equivalently, we may define D^w as the first moment at which some subschedule completes a total weight of at least w. Then,

$$\text{OPT}' = \sum_j w_j C'_j = \sum_{w=1}^{W} D^w. \tag{5.4}$$

The properties of the pseudo schedule OPT$'$ are listed in the next lemma.

LEMMA 5.1. *Solution* OPT$'$ *has the following properties:*

(i) No job is processed at time t_i and the subschedule between time points t_i and t_{i+1} is a feasible schedule on itself. Here, $t_i = 3(1+\epsilon)^{(i-2)K+h_0}$ for all $i = 1, 2, \ldots, \Gamma$, where $K = O\left(\frac{1}{\epsilon^2}\right)$, $\Gamma = O(\epsilon \log n)$ and h_0 is some fixed number in $\{0, 1, \ldots, K-1\}$.

(ii) The total weight of jobs scheduled in the i-th subschedule is non-decreasing in i.

(iii) For any $w \in \{1, 2, \ldots, W\}$, define D^w as the first moment at which some subschedule completes a total weight of at least w. Then, $\sum_{w=1}^{W} D^w \leqslant (1+\epsilon)$OPT.

Now consider any pseudo schedule that satisfies (i) and (ii) and let D^w be as defined in (*iii*) and let C^w be the moment that the schedule completes a total weight of at least w. Then (using 5.4)

$$\sum_j w_j C_j = \sum_{w=1}^{W} C^w \leqslant \sum_{w=1}^{W} D^w.$$

(Equality holds for OPT$'$.) Hence, we may restrict to pseudo schedules which have properties (*i*) and (*ii*) and among those, minimize $\sum_{w=1}^{W} D^w$ as defined in (*iii*). This can be done approximately by dynamic programming as before if we have an approximation algorithm for the following subproblem on subschedules.

Subproblem An instance of a subproblem is given by $i \in \{1, \ldots, \Gamma\}$ and numbers $w' \leqslant w'' \in \{0, 1, \ldots, n^3\}$. A solution is a schedule that starts at time t_i and completes before time t_{i+1} and completes a total weight of at least w''. let C^w be the moment that the schedule completes a total weight of at least w. The objective is to minimize $\sum_{w=w'+1}^{w''} C^w$. Note that an instance (i, w', w'') may not be feasible. For any feasible instance, let $\text{SUB}_i(w', w'')$ be its optimal value.

DEFINITION 5.2. *An (α, β) approximation algorithm for the subproblem is an algorithm that finds for any* feasible *instance (i, w', w'') a schedule that does not start before time αt_i and ends before time αt_{i+1}, completes a total weight of at least w'', and for which $\sum_{w=w'+1}^{w''} C^w \leqslant \alpha\beta \text{SUB}_i(w', w'')$.*

Note that the total weight w'' is not approximated in the definition above. For example, completing a total weight of $(1-\epsilon)w''$ is not sufficient to obtain a PTAS.

Assume we have an (α, β)-approximation algorithm ALG for the subproblem. Let $\text{ALG}_i(w', w'')$ be the value returned by the algorithm for instance (i, w', w'') and let it be infinite if no solution was found. For any sequence of integers $0 \leq \hat{w}_1 \leqslant \cdots \leq \hat{w}_\Gamma = W$ we get a pseudo schedule of total weighted completion time

$$\sum_{i=1}^{\Gamma} \text{ALG}_i(\hat{w}_{i-1}, \hat{w}_i) \leqslant \alpha\beta \sum_{i=1}^{\Gamma} \text{SUB}_i(\hat{w}_{i-1}, \hat{w}_i) \tag{5.5}$$

by concatenating the schedules $\text{ALG}_i(\hat{w}_{i-1}, \hat{w}_i)$. Minimizing the left side of (5.5) over all values $0 \leq \hat{w}_1 \leqslant \cdots \leq \hat{w}_\Gamma = W$ is easy since they form a non-decreasing sequence and the minimum can be computed by a simple dynamic program similarl to what was done for the TRP. Let the values $\hat{w}_i$ minimize the left side of (5.5) and let w_i be the total weight in the partial solution OPT_i. Then, algorithm ALG finds a solution of total weighted completion time at most

$$\begin{aligned} \alpha\beta \sum_{i=1}^{\Gamma} \text{SUB}_i(\hat{w}_{i-1}, \hat{w}_i) &\leq \alpha\beta \sum_{i=1}^{\Gamma} \text{SUB}_i(w_{i-1}, w_i) \\ &\leqslant \alpha\beta \text{OPT}' \leq \alpha\beta(1+\epsilon)\text{OPT}. \end{aligned}$$

The number of subproblems is $O(\Gamma n^6)$ and given all approximate values, the optimal values $\hat{w}_i$ can be computes in $O(\Gamma n^6)$ time. Further, the number of choices for h_0 is K (See Equation 2.1). Hence, it takes $O(K\Gamma n^6) = \tilde{O}(n^6 \log n)$ calls to the approximation algorithm for the subproblem to get an $\alpha\beta(1+\epsilon)$-approximation for our scheduling problem.

5.2 Approximating the subproblem. We show how to get a $(1+\epsilon, 1+\epsilon)$-approximation for the subproblem. In this section, we fix an arbitrary subproblem with parameters i, w', w'' and fix an optimal solution SUB*. Again, the first step is to partition the interval from t_i till t_{i+1} into K parts that we shall denote as *slots*. As before (Equation (2.3)), let

$$t_i^{(h)} = (1+\epsilon)^h t_i, \text{ for } h = 0, \ldots, K.$$

From now, the approach will differ from what we did for the TRP. The general idea is as follows. Since the number of slots in a subschedule is a constant K, and all weights and processing times are polynomially bounded, we can afford to guess a lot of information about SUB*. We shall do this in such a way that the remaining jobs can be scheduled greedily. For the ease of analysis, we extend SUB* by putting all unscheduled jobs at the end. We say that they are scheduled in a virtual slot $K+1$. Now, for each job j we guess a set of possible slots $S_j \subseteq \{1, 2, \ldots, K+1\}$ with the following properties:

(P1) Any job j in SUB* completes in some slot in S_j. (It may start in an earlier slot though.)

(P2) If $j_1 \prec j_2$ then $\max(S_{j_1}) \leqslant \min(S_{j_2})$.

The first property is easily satisfied. For example, if we let S_j be the set of all $K+1$ slots for each j. The second property is implied by the interval order precedence constraints as we shall prove in Lemma 5.2 below. After this lemma, we prove that we get a PTAS for any class of precedence constraints for which we can prove (P1) and (P2) and for which we may restrict to polynomially bounded weights and processing times. Roughly speaking, the consequence of (P1) and (P2) is that we only need to deal with precedence constraints within a slot. However, within a slot any order of the jobs that satisfies the precedence constraints is fine since all completion times are within a factor $(1+\epsilon)$.

LEMMA 5.2. *For interval orders, we can guess sets S_j for $j = 1, 2, \ldots, n$, that satisfy properties (P1) and (P2).*

Proof. Let $[l_j, r_j]$ be the interval for job j in the interval order. As noted, we may assume that the $2n$ values l_j, r_j are all different. For any $h \in \{1, \ldots, K+1\}$, let J^h be the set of jobs that complete in slot h in SUB*. Note that J^h may be empty. For each non-empty set, guess the job j^h with the largest value l_{j^h}, i.e., $l_{j^h} = \max\{l_j \mid j \in J^h\}$ and define $S_{j^h} = \{h\}$. (Note that there are $n^{O(K)}$ possible guesses.) For any other job, the set S_j is defined as the unique maximal subset of $\{1, \ldots, K+1\}$ that satisfies the following four necessary conditions.

(a) If for some h, the guess was $J^h = \emptyset$, then $h \notin S_j$.

(b) If $j \prec j^h$ for some slot h, then $\max(S_j) \leqslant h$.

(c) If $j^h \prec j$ for some slot h, then $\min(S_j) \geqslant h$.

(d) If $l_j > L^h$ for some h, then $h \notin S_j$.

(In (c), one might replace $\geqslant h$ by $\geqslant h+1$ since $h \notin S_j$ follows from (d).) Assume that we guessed all jobs j^h correctly. Then, property (P1) follows directly since the conditions (a)–(d) are clearly necessary. To prove (P2) assume that $j_1 \prec j_2$. We distinguish three cases:
Case 1: $j_2 = j^h$ for some j^h. It follows from (b) that $\max(S_{j_1}) \leqslant h = \min(S_{j_2})$, since $S_{j_2} = \{h\}$.
Case 2: $j_1 = j^h$ for some j^h. It follows from (c) that $\min(S_{j_2}) \geqslant h = \max(S_{j_1})$, since $S_{j_1} = \{h\}$.
Case 3: Now assume that $j_1, j_2 \neq j^h$ for any j^h. Let $h = \min(S_{j_2})$. Then by (a), $J^h \neq \emptyset$. Then by (d), $l_{j_2} < l_{j^h}$. It follows form $j_1 \prec j_2$ that $r_{j_1} < l_{j_2} < l_{j^h}$, Hence, $j_1 \prec j^h$ and then (b) implies $\max(S_{j_1}) \leqslant h = \min(S_{j_2})$. □

Assume from now on that we have sets S_j satisfying (P1) and (P2).

Constructing the schedule We will construct a $(1+\epsilon, 1+\epsilon)$-approximate schedule σ. The construction is done as follows. First, we assign each job j to some slot in S_j. Jobs that are not assigned to any of the first K slots are implicitly assigned to the virtual slot $K+1$. The slots $1, 2, \ldots, K$ are placed one after the other in this order, startingat time $(1+\epsilon)t_i$, and within a slot the jobs are placed in any arbitrary order that satisfies the precedence constraints. By property (P2), the resulting schedule σ is guaranteed to be feasible. The word *slot* is ambiguous here since the start and end time of slots in σ are not fixed and do not match those of SUB*. We will show however that in the final schedule σ, the end time of slot h is at most a factor $1+\epsilon$ larger than that of slot h in SUB*.

Say that a job is *large* if its processing time is at least $f(\epsilon)t_i$, where $f(\epsilon)$ is some function of ϵ to be specified later. Call it *small* otherwise. Since there can only be a constant number (depending on ϵ) of large jobs scheduled in SUB* we

- guess all large jobs together with the slot $(1, \ldots, K)$ in which they complete in SUB* and assign a job to slot h in σ if it completes in slot h in SUB*.

It remains to assign the small jobs. Note that there are at most 2^{K+1} different sets S_j. For any $S \subseteq \{1, \ldots, K+1\}$, let $J_S = \{j \mid S_j = S \text{ and } j \text{ is small}\}$.

- For every pair (S, h), with $h \in \{1, \ldots, K\}$ we guess the total processing time over all jobs $j \in J_S$ which complete in SUB* in slot h. Let $P(S, h)$ be this value.

For each $S \subseteq \{1, \ldots, K+1\}$ place the jobs in J_S in non-decreasing order w_j/p_j and do the following:

- For slots $h = 1$ to K, assign jobs from J_S in order w_j/p_j to slot h until the total processing time of jobs from J_S assigned to h becomes at least $P(S, h)$ or until all jobs from J_S are assigned.

Given this assignment of jobs to slots, we schedule jobs within each slot in an arbitrary order that satisfies the precedence constraints. Note that there are only $\tilde{O}(1)$ large jobs and we can guess all of them together with their slots. Also, the number of pairs (S,h) is $2^{K+1}K = \tilde{O}(1)$ and for each pair, the number of possible values $P(S,h)$ is $O(n^3)$ since $p_j \leqslant n^2$ for all j. Hence, the total number of choices for the guesses is $n^{\tilde{O}(1)}$. Let σ be the schedule that follows from correct guesses about SUB*.

LEMMA 5.3. *Schedule σ is a $(1+\epsilon, 1+\epsilon)$-approximation for the subproblem.*

Proof. By property (P2) and since we scheduled jobs within a slot in an order satisfying the precedence constraint, the schedule is feasible.

Next we show that slot h in σ ends before time $(1+\epsilon)t_i^{(h)}$. Let $P_\sigma(S,h)$ be the total processing time of jobs from J_S which are assigned to slot h in σ. Further, let $P(h)$ be the total processing time of jobs that complete in slot h in SUB* and let $P_\sigma(h)$ be the total processing time of jobs assigned to slot h in σ. Remember that a job is small if its processing time is at most $f(\epsilon)t_i$. By the greedy assignment of small jobs we have that

$$P_\sigma(S,h) \leqslant P(S,h) + f(\epsilon)t_i.$$

The number of possible sets S is 2^{K+1}. Now, take $f(\epsilon) = \epsilon^2 \cdot 2^{-(K+1)}$. Then,

$$P_\sigma(h) \leqslant P(h) + 2^{K+1}f(\epsilon)t_i = P(h) + \epsilon^2 t_i.$$

Slot 1 in SUB* has length ϵt_i and the total processing time assigned to slot 1 is at most $P(1) + \epsilon^2 t_i \leqslant \epsilon t_i + \epsilon^2 t_i = (1+\epsilon)\epsilon t_i$. Slot 1 is the smallest slot in SUB*. Hence, in general, the total time assigned to the first h slots is at most $(1+\epsilon)$ times the length of the first h slots in SUB*. That means, slot h in σ ends before time $(1+\epsilon)t_i + (1+\epsilon)(t_i^{(h)} - t_i) = (1+\epsilon)t_i^{(h)}$.

Next, we prove the bound on the value of the schedule σ. Take arbitrary $S \subseteq \{1,\dots,K+1\}$. If SUB* completes a total weight w of jobs form J_S by the end of slot h, then our schedule will have completed at least the same weight of jobs form J_S by the end of slot h too, since we scheduled the jobs in w_j/p_j order. For any $w \in \{1,\dots,w''\}$, let C^{*w} be the time at which SUB* completes a total weight of at least w. Consider arbitrary w and assume that time C^{*w} falls in slot h. Then our schedule completes a total weight of at least w before the end time of slot h. Hence, before time $(1+\epsilon)t_i^{(h)} = (1+\epsilon)^2 t_i^{(h-1)} < (1+\epsilon)^2 C^{*w}$. In particular, this applies to any $w \in \{w'+1,\dots,w''\}$. Hence,

$$\text{ALG}_i(w',w'') \leqslant (1+\epsilon)^2 \text{SUB}_i(w',w''). \qquad \square$$

The PTAS for interval ordered precedence constraints can easily be adjusted to deal with release dates. First, the release dates may be rounded such that jobs are released at the beginning of slots. Next, the release date restrictions are added to the sets S_j as defined in the proof of Lemma 5.2. The rest remains the same.

References

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, *Approximation Schemes for Minimizing Average Weighted Completion Time with Release Dates*, Proceedings of the 40th Annual Symposium on Foundations of Computer Science, (1999), 32–43.

[2] F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, and N. Papakostantinou, *The complexity of the travelling repairman problem*, RAIRO Journal on Information Theory and Applications, 20 (1986), 79–87.

[3] Christoph Ambühl, Monaldo Mastrolilli, Nikolaus Mutsanas, and Ola Svensson, *On the approximability of single-machine scheduling with precedence constraints*, Mathematics of Operations Research 36, (2011), 653–669.

[4] A. Archer and A. Blasiak, *Improved approximation algorithms for the minimum latency problem via prize-collecting strolls*, Proc. 21th Symp. on Discrete Algorithms, 2010, 429–447.

[5] A. Archer and D.P. Williamson, *Faster approximation algoriths for the minimum latency problem*, Proc. 14th Symp. on Discrete Algorithms, Baltimore, Maryland, 2003, 88–96.

[6] S. Arora, *Polynomial-time approximation schemes for euclidean TSP and other geometric problems*, Proc. 37th Symp. Foundations of Computer Science, (1996), 2–12.

[7] ———, *Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems*, Journal of the ACM,4 5 (1998), 753–782.

[8] ———, *Approximation schemes for NP-hard geometric optimization problems: A survey.*, Mathematical Programming, 97 (2003), 43–69.

[9] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn, *A polynomial-time approximation scheme for weighted planar graph TSP*, Proc. of the 9th ACM–SIAM Symposium on Discrete Algorithms, (1998), 33–41.

[10] S. Arora and G. Karakostas, *Approximation schemes for minimum latency problems*, Proc. 31st ACM Symposium on Theory of Computing, Atlanta, (1999), 688–693.

[11] ———, *A $2+\epsilon$-approximation for the k-mst problem*, Proc. 11th Symp. on Discrete Algorithms, San Francisco, California, (2000), 754–759.

[12] ———, *Approximation schemes for minimum latency problems*, SIAM Journal on Computing, 32 (2003), 1317–1337.

[13] A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, and M. Sudan, *The minimum latency problem*, Proc. 26th ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, (1994), 163–171.

[14] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar, *Paths, trees, and minimum latency tours*, Proc. 44th Symp. Foundations of Computer Science, Cambridge, Massachusettes, (2003), 36–45.

[15] T. Dewilde, D. Cattrysse, S. Coene, F.C.R. Spieksma, and P. Vansteenwegen, *Heuristics for the traveling repairman problem with profits*, Computers & Operations Research 40 (2013), 1700-1707

[16] K. Dhamdhere, A. Gupta, and R. Ravi, *Approximation algorithms for minimizing average distortion*, Theory of Computing Systems, 39 (2006), 93–111.

[17] J. Fakcharoenphol, C. Harrelson, and S. Rao, *The k-traveling repairman*, Proc. 14th Symp. on Discrete Algorithms, Baltimore, Maryland, (2003), 646–654.

[18] A. García, P. Jodrá, and J. Tejel, *A note on the travelling repairman problem*, Networks, 40 (2002), 27–31.

[19] M.X. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, Mathematical Programming, 82 (1998), 111–124.

[20] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of Discrete Mathematics, 5 (1979), 287–326.

[21] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric algorithms and combinatorial optimization*, Springer, (1988).

[22] E. Koutsoupias, C.H. Papadimitriou, and M. Yannakakis, *Searching a fixed graph*, Proc. 23rd International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, vol. 1099, Paderborn, Germany, Springer, (1996), 280–289.

[23] S.O. Krumke, W.E. de Paepe, D. Poensgen, and L. Stougie, *News from the online traveling repairman problem*, Theoretical Computer Science, 295 (2003), 279–294.

[24] E.L. Lawler, *Sequencing jobs to minimize total weighted completion time subject to precedence constraints*, Ann. Discrete Math., 2 (1978), 75–90.

[25] J.K. Lenstra and A.H.G. Rinnooy Kan, *The complexity of scheduling under precedence constraints*, Operations Research, 26 (1978), 22–35.

[26] Viswanath Nagarajan and R. Ravi, *The directed minimum latency problem*, Proc. 11th int. workshop, APPROX 2008, and 12th int. workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization, 2008, 193–206.

[27] J. B. Sidney, *Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs.*, Operations Research, 23 (1975), 283–298.

[28] R.A. Sitters, *The minimum latency problem is NP-hard for weighted trees*, Proc. 9th Int. Conf. Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, vol. 2337, Springer, (2002), 230–239.

[29] J.N. Tsitsiklis, *Special cases of traveling salesman and repairman problems with time windows*, Networks, 22 (1992), 263–282.

[30] G.J. Woeginger, *On the approximability of average completion time scheduling under precedence constraints.*, Discrete Applied Math., 131 (2003), 237–252.