Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows

Nikhil Bansal
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15232
nikhil@cs.cmu.edu

Avrim Blum
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15232
avrim@cs.cmu.edu

Shuchi Chawla
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15232
shuchi@cs.cmu.edu

Adam Meyerson
Computer Science Department
University of California, Los Angeles
Los Angeles, CA
awm@cs.ucla.edu

ABSTRACT

Given a metric space G on n nodes, with a start node r and deadlines $D(v)$ for each vertex v, we consider the *Deadline-TSP* problem of finding a path starting at r that visits as many nodes as possible by their deadlines. We also consider the more general *Vehicle Routing with Time-Windows* problem, in which each node v also has a release-time $R(v)$ and the goal is to visit as many nodes as possible within their "time-windows" $[R(v), D(v)]$. No good approximations were known previously for these problems on general metric spaces. We give an $O(\log n)$ approximation algorithm for Deadline-TSP, and extend this algorithm to an $O(\log^2 n)$ approximation for the Time-Window problem. We also give a bicriteria approximation algorithm for both problems: Given an $\epsilon > 0$, our algorithm produces a $\log(1/\epsilon)$ approximation, while exceeding the deadlines by a factor of $1 + \epsilon$. We use as a subroutine for these results a constant-factor approximation that we develop for a generalization of the orienteering problem in which both the start and the end nodes of the path are fixed. In the process, we give a 3-approximation to the orienteering problem, improving on the previously best known 4-approximation of [6].

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]:

General Terms

Algorithms, Theory

Keywords

Approximation Algorithms, Vehicle Routing, Traveling Salesman Problem, Orienteering

1. INTRODUCTION

Consider a robot that begins at position r in some metric space and has a number of different locations to visit. However, each location v has a deadline $D(v)$ and only counts if it is visited by that time. What path should the robot take to (approximately) maximize the number of points visited by their deadlines — or, more generally, to maximize the value of such points if each location has an associated value? We call this the *Deadline-TSP* problem. Even more generally, what if each location v has a *time-window* $[R(v), D(v)]$ in which it must be visited? This is the classic *Vehicle Routing Problem with Time Windows*. In this paper, we give approximation guarantees for both of these problems in general metric spaces. Note that if the deadlines of all the nodes are the same, and their release dates are zero, the problem reduces to the well known *Orienteering* problem [3, 4, 11], for which the first constant factor approximation was recently given by [6].

The Vehicle Routing Problem with Time Windows, or "Time-Window problem" for short, has been studied extensively in the Operations Research literature (see [1, 10] for a survey). Various heuristics [9, 17, 19, 18], such as local search, Simulated Annealing and Genetic algorithms, as well as cutting plane and branch and bound methods [20, 16, 14] have been studied for solving this problem optimally. Optimal algorithms for stochastic inputs have also been proposed. In the approximation algorithms literature, there has been work on geometric versions of this problem. Several constant factor approximations [5, 21, 15] have been proposed for the case of points on a line. For general graphs, Chekuri and Kumar [8] give a constant-factor approximation when there are a constant number of different deadlines (or time windows). Our paper is the first to give approximation guarantees for the general case with arbitrary deadlines or arbitrary time-windows.

Another motivation for these problems comes from a classic scheduling problem known as scheduling with sequence dependent setup times [2, 23, 22, 7]. In this problem we are given a collection of jobs, each having a production duration p_j and a delivery date D_j. In addition, for each pair of jobs, there is a setup time s_{ij}, which is incurred when job j is undertaken following job i. The goal is to schedule jobs as efficiently as possible on a single machine. If we assume the s_{ij} are symmetric (e.g., they depend on some notion of similarity between the jobs) then this can be modeled as an undirected graph problem by assigning a length of s_{ij} to the edge between i and j. Production durations can also be incorporated by adding $p_j/2$ to every edge incident on the node j, and subtracting the same from the deadline D_j. The scheduling setting is fairly general and models many problems. For example, one objective may be to minimize the makespan, which is equivalent to the TSP problem. Likewise, the objective of completing as many jobs as possible before their specified deadline is equivalent to the Deadline-TSP. Interestingly, meeting target delivery dates has been declared as the most important scheduling objective for production planners by Wisner and Siferd [22]. Several heuristics [13, 12] have been proposed for this (and other related problems), however, no approximations were known prior to our work.

Our Results. We give an $O(\log n)$ approximation for the Deadline-TSP, based on a 3-approximation that we provide for an extension of the Orienteering problem, that we call "point-to-point orienteering". In point-to-point orienteering, both the start node and end node of the path are given, and the goal is to travel from the start to the end, traveling distance at most D, and visiting (approximately) as many points as possible. Note that we approximate the number of points, and not the distance traveled. This means that if we can break up the problem into several subparts, then we can hope to attack each piece individually and patch the solutions together without worrying that mistakes made earlier will affect performance down the line. Since point-to-point orienteering generalizes the standard version of this problem, this improves on the previous 4-approximation of [6]. Using the $O(\log n)$ approximation to Deadline-TSP, we develop an $O(\log^2 n)$ approximation for the Time-Window problem.

We also give an algorithm for the Time-Window problem, that achieves a bicriteria optimization — it achieves a constant factor approximation to the reward, while exceeding the deadlines by a small factor. In particular, for any given $\epsilon > 0$, our algorithm collects an $\Omega(\log^{-1}(\frac{1}{\epsilon}))$ fraction of the reward, while visiting each vertex v in the interval $[R(v), (1+\epsilon)D(v)]$, where $[R(v), D(v)]$ denotes the time-window of the vertex v. Note that unlike typical bicriteria results, our approximation is logarithmic in $1/\epsilon$ rather than linear. In particular, this implies an $O(\log D_{\max})$-approximation for the Time-Window problem, where $D_{\max}$ is the maximum deadline in the graph, by taking $\epsilon = 1/D_{\max}$. This gives an asymptotically better approximation than our $O(\log^2 n)$-approximation if all the deadlines in the graph are polynomially bounded in n.

The rest of this paper is organized as follows. We begin with some notation and definitions in Section 2. In Section 3, we give a 3-approximation algorithm for the point-to-point orienteering problem. Section 4 contains some constant factor approximations for special cases of the Time-Window problem, that we then use to achieve our bicriteria approximation. Section 5 contains an $O(\log^2 n)$-approximation algorithm. We conclude in Section 6.

2. NOTATION AND PRELIMINARIES

Let $G = (V, E)$ be a weighted graph, with a start node r, a prize (or reward) function $\Pi : V \to \mathcal{Z}^+$, deadlines $D : V \to \mathcal{Z}^+$, release dates $R : V \to \mathcal{Z}^+$, and a length function $\ell : E \to \mathcal{Z}^+$. We assume without loss of generality that the deadline of every node is larger than the release date of the node and the shortest path distance from the root to the node.

For any two nodes u and v, let $\ell(u, v)$ denote the shortest distance between u and v. Given a path P from u to v, let $t_P(u, v)$ denote the time taken (distance) to reach v from u along the path P. The excess along the path P is defined as $\epsilon_P(u, v) = t_P(u, v) - \ell(u, v)$. We abbreviate the time taken by a path P rooted at r to reach a node v, $t_P(r, v)$, by $t_P(v)$. We denote the optimal path by $\mathcal{O}$. Let $\mathcal{O}$ also denote the set of nodes visited by the optimal path within their time-windows. A path starting at root r collects the prize $\Pi(v)$ at node v, if it reaches v within the time interval $[R(v), D(v)]$. For a path P and set $S \subseteq V$, let $\Pi_P(S) = \sum_{v \in S: R(v) \le t_P(v) \le D(v)} \Pi(v)$ and $\Pi_P = \Pi_P(V)$. Note that a path can visit a node multiple times, but the prize at any node can be collected at most once. For any path P, the restriction of the path P to a set S of vertices, $P_{|S}$, denotes the path that visits nodes of S in the order that P visited them, and does not visit any other nodes. Note that, for all S and P, $t_{P_{|S}}(v) \le t_P(v)$ for all $v \in S$.

The goal in the Time-Window problem is to construct a path starting at r, that maximizes the total prize. We define the *Deadline-TSP* problem to be the special case where all the release dates $R(v)$ are 0. The Orienteering problem is a further special case of this problem in which all nodes $v \in V$ have $R(v) = 0$ and all have the same deadline $D(v) = D$. We define the *u-v orienteering problem* to be the orienteering problem with root node u that is required to end at node v. That is, the goal here is to find the path of length at most D that begins at u and ends at v that collects the maximum reward possible.

The minimum excess path problem is defined as follows. Given a graph with nodes u and v, and a prize quota k, the problem is to find a path P from u to v that collects at least k prize, and has the minimum possible excess $\epsilon_P(u, v)$. Note that in terms of exact solutions, this problem is the same as the k-TSP problem, in which the goal is to find the shortest path between u and v that collects at least k prize, but we are subtracting $\ell(u, v)$ from the objective, so the min-excess problem is more difficult in terms of approximation. Blum et al [6] give a $(2 + \epsilon)$-approximation for the excess problem (if the optimum path is $\mathcal{O}$, the algorithm finds a path that collects a prize at least k and has length at most $d(u, v) + (2 + \epsilon)\epsilon_{\mathcal{O}}(u, v))$, and then use this to get a 4-approximation to the orienteering problem.

We begin by showing how we can use the $(2+\epsilon)$-approximation to the minimum-excess path problem to find a path from u to v that has length at most $t_{\mathcal{O}}(u, v)$, and collects prize at least $k/3$. In other words, this gives a 3-approximation to the $u - v$ Orienteering problem, improving over [6] (in fact, this is stronger than the version of the problem solved in [6] because we get to specify the ending point of the path, and not just the starting point).

Input: Graph $G = (V, E)$; special nodes u and v; length bound D.
Output: Path P with $\Pi_P \geq \frac{1}{3}\Pi_{\mathcal{O}}$ and length at most D.

1. For every pair of nodes x and y, we will consider a path which proceeds from u to x, then visits some vertices while traveling from x to y, and then travels directly from y to v. The allowed excess of the path from x to y is $\epsilon_{xy} = D - \ell(u, x) - \ell(x, y) - \ell(y, v)$. Using the $2 + \epsilon$-approximation to the minimum excess problem from [6], we compute a path from x to y of excess at most ϵ_{xy} that visits at least as many vertices as the best path from x to y with excess $\epsilon_{xy}/3$.

2. We now pick the pair (x, y) that maximizes the total reward collected on the computed path. We then travel from u to x via a line, then x to y via the chosen path, then y to v.

Figure 1: Algorithm *P2P*— 3-approximation for $u - v$ Orienteering

We then show how we can use this as a subroutine to obtain a bicriteria approximation—for any $\epsilon > 0$, our algorithm outputs a path P with $\Pi^{\epsilon}_P \geq \Omega(\frac{1}{\log(1/\epsilon)})\Pi_{\mathcal{O}}$, where $\Pi^{\epsilon}_P = \sum_{v:R(v)\leq t_P(v)<(1+\epsilon)D(v)} \Pi(v)$. That is, for a constant ϵ, our algorithm obtains a constant fraction of the reward, while exceeding deadlines by a $1 + \epsilon$ factor. The same algorithm also gives us an $O(\log D_{\max})$ approximation without approximating deadlines, where $D_{\max} = \max_{v\in V} D(v)$ is the maximum deadline in the graph.

Our main result is a $3 \log n$ approximation for the Deadline-TSP problem, which we extend to a $3 \log^2 n$ approximation for the Time-Window problem. These algorithms use point-to-point orienteering as a subroutine.

We assume without loss of generality that prizes are in the range $\{1, \ldots, n\}$. In doing so, we only lose a factor of 2 in approximation. This is because, we can scale down prizes such that the maximum prize is exactly n (this guarantees that $\mathcal{O}$ gets at least n reward). Then we round down the prizes to the nearest integer, losing an additive amount of at most n. Note that since the new reward obtained by $\mathcal{O}$ is also at least n, we lose a factor of at most 2.

All our algorithms (as presented here) use as a subroutine a dynamic program that computes the maximum reward achievable between two specified end points for all possible path lengths. So, strictly speaking, the running time of these algorithms has a polynomial dependence on n and $D_{\max}$. This would be undesirable if $D_{\max}$ is exponential in n. However, we can use the following idea to reduce the running time to polynomial in n and $\log D_{\max}$. For all possible reward values (that are integers between 1 and n^2), we use binary search to find the smallest length for which that reward can be obtained. So we only need to compute and store polynomially many different lengths for each start and end pair. For ease of exposition, we ignore this detail while describing the algorithms.

Although all our algorithms run in polynomial time, the actual running times are fairly large. The bicriteria approximation (Algorithm $\mathcal{C}$, see Figure 5) requires $O(n^5 \log \frac{1}{\epsilon})$ applications of the Orienteering subroutine (Algorithm *P2P*, see Figure 1). The $O(\log n)$ approximation (Figure 7) requires $O(n^6)$ applications of Algorithm *P2P*. Algorithm *P2P* requires $O(n^2)$ applications of the algorithm for the min-excess problem. Note that these are worst case running times for the algorithms, and the actual running times may be much smaller.

3. POINT-TO-POINT ORIENTEERING

In this section we show how we can use a $(2+\epsilon)$-approximation algorithm for the minimum excess problem, to achieve a 3-approximation to the point-to-point orienteering problem.

We begin with some properties of excess. Let P be any $u-v$ path that visits nodes in the order $u_0 = u, u_1, \ldots, u_l = v$.

Fact 1 $\epsilon_P(u_0, u_i)$ is increasing in i.

Fact 2 The excess function is sub-additive. That is, for any nodes u_i, u_j, u_k with $0 \leq i < j < k \leq l$, $\epsilon_P(u_i, u_j) + \epsilon_P(u_j, u_k) \leq \epsilon_P(u_i, u_k)$.

PROOF. The first fact follows from the triangle inequality. The second follows by rewriting $\epsilon_P(u_i, u_j) + \epsilon_P(u_j, u_k)$ as $t_P(u_i, u_j) - \ell(u_i, u_j) + t_P(u_j, u_k) - \ell(u_j, u_k) = t_P(u_i, u_k) - \ell(u_i, u_j) - \ell(u_j, u_k)$ and observing that $\ell(u_i, u_j) + \ell(u_j, u_k) \geq \ell(u_i, u_k)$ by the triangle inequality. □

The algorithm *P2P* is given in Figure 1. Note that by design, the algorithm produces a path of length at most D. We just need to show that it visits enough points.

THEOREM 1. *The algorithm* P2P *is a 3-approximation to the point-to-point orienteering problem.*

PROOF. Consider the optimum path $\mathcal{O}$ from u to v. Break this path into three pieces, each having at least 1/3 of the total prize value, and let x and y be the endpoints of the portion such that $\epsilon_{\mathcal{O}}(x, y)$ is smallest. Consider now the path O' that travels directly from u to x, then follows $\mathcal{O}$ to y, and then travels directly from y to v. Because we chose x, y such that $\epsilon_{\mathcal{O}}(x, y)$ is the smallest of the three segments, and by the triangle inequality, it must be that by traveling along O' rather than $\mathcal{O}$, we save a total of at least $2\epsilon_{\mathcal{O}}(x, y)$; that is, $t_{\mathcal{O}}(u, v) - t_{O'}(u, v) \geq 2\epsilon_{\mathcal{O}}(x, y)$. This is enough to pay for the added length produced by applying the min-excess approximation from x to y. Formally, by definition, $\epsilon_{xy} = t_{\mathcal{O}}(u, v) - t_{O'}(x, y) + \epsilon_{\mathcal{O}}(x, y)$, and plugging in the above inequality, this is at least $3\epsilon_{\mathcal{O}}(x, y)$. Since the min-excess algorithm gets at least as much prize value as the best possible path from x to y with excess $\epsilon_{xy}/3$, it is guaranteed to get at least as much as this portion of the optimal path, which is at least 1/3 of the total prize of the optimal path. □

4. A BICRITERIA APPROXIMATION

In this section, we describe a bicriteria algorithm for the Time-Window problem that for any $\epsilon > 0$, obtains an $O(\log \frac{1}{\epsilon})$ fraction of the reward obtained by $\mathcal{O}$, if it is allowed to visit vertices in the window $[R(v), (1 + \epsilon)D(v)]$.

Input: Graph $G = (V, E)$ with deadlines $D(v)$; Constant ϵ; $f = \frac{1}{\sqrt{1+\epsilon}}$. Let $S_j = \{v : D(v) \in (f^j D_{\max}, f^{j-1} D_{\max}]\}$, $\mathcal{S}_{i\text{mod}3} = \cup_{j=0}^{\infty} S_{3j+i}$.
Output: Path P with $\Pi_P \geq \frac{1}{9}\Pi_{\mathcal{O}}(V_\epsilon)$ and $R(v) \leq t_P(v) \leq (1+\epsilon)^2 D(v)$ for all $v \in P$, where $V_\epsilon = \{v : D(v)/(1+\epsilon) \leq t_{\mathcal{O}}(v) \leq D(v)\}$.

1. For all j, for all $u, v \in S_j$, and for all integers $t \leq f^{j-1}D_{\max} - f^j D_{\max}$, use algorithm *P2P* to find a path of length t from u to v. Let this reward be $\pi(u, v, t)$.

2. For $k = 0, 1, 2$ do the following:

   (a) Let $\pi'(v, j, t)$ for $t \leq f^{j-1}D_{\max}$ denote the (approximately) maximum reward that can be collected from $\mathcal{S}_{k\text{mod}3}$ up to segment $S_j \subset \mathcal{S}_{k\text{mod}3}$ using a path of length t that ends at $v \in S_j$.

   (b) Compute $\pi'(v, j, t)$ using the following recurrence.

$$\pi'(v, j, t) = \max_{f^j D_{\max} \leq t' \leq t, u \in S_j, u' \in S_{j+3}} \{\pi'(u', j+3, t' - \ell(u', u)) + \pi(u, v, t - t')\}$$

3. Output the path corresponding to the maximum prize collected until the last segment, slowed-down by a factor of f^3.

Figure 2: Algorithm $\mathcal{A}$—Bicriteria approximation for the small margin case

We begin with a few special purpose algorithms, and then describe how to combine them for the general case. We first consider the case when $\mathcal{O}$ visits a large fraction of the nodes very close to their deadlines (the *small margin* case). Then we consider the case when $\mathcal{O}$ visits many nodes much before their deadlines (the *large margin* case). Towards the end of this section, we show that the bicriteria result also implies a $O(\log D_{\max})$-approximation to the Time-Window Problem.

4.1 The small margin case

At the heart of our bicriteria approximation is a procedure that obtains a constant fraction of the optimal reward while exceeding deadlines by a small factor, if $\mathcal{O}$ visits most nodes $v \in \mathcal{O}$ very close to their deadlines (within some multiplicative factor).

Let ϵ be a fixed constant. Consider the set of nodes $V_\epsilon = \{v : D(v)/(1+\epsilon) \leq t_{\mathcal{O}}(v) \leq D(v)\}$. We design an algorithm $\mathcal{A}$ that obtains a constant fraction of $\Pi_{\mathcal{O}}(V_\epsilon)$ as reward, while exceeding deadlines by a factor of $(1+\epsilon)^2$. Note that only deadlines are violated – nodes on the path output by algorithm $\mathcal{A}$ are visited after their release dates.

Let $f = \frac{1}{\sqrt{1+\epsilon}}$. We divide the nodes in the graph into segments as follows. Segment S_j consists of nodes that have deadlines in $(f^j D_{\max}, f^{j-1} D_{\max}]$ (Figure 3). The definition of f and V_ϵ are such that for any j, all vertices in $S_j \cap V_\epsilon$ are visited by $\mathcal{O}$ after all vertices in $S_{j+3} \cap V_\epsilon$ (see Lemma 2 and Figure 3).

Lemma 2. *For any j and any nodes $u \in S_j \cap V_\epsilon$ and $v \in S_{j+3} \cap V_\epsilon$, $t_{\mathcal{O}}(v) < t_{\mathcal{O}}(u)$.*

Proof. We have $D(v) \leq f^{j+2} D_{\max}$. So, $t_{\mathcal{O}}(v) \leq D(v) \leq f^{j+2} D_{\max}$. Likewise, $D_u > f^j D_{\max}$. So, by the definition of V_ϵ, $t_{\mathcal{O}}(u) \geq f^2 D_u > f^{j+2} D_{\max} \geq t_{\mathcal{O}}(v)$. $\square$

The above lemma suggests a natural strategy for approximating the reward collected by $\mathcal{O}$ in V_ϵ. Let $\mathcal{S}_{i\text{mod}3} = \cup_{j=0}^{\infty} S_{3j+i}$ for $i = 0, 1, 2$. Then, $\cup_i \mathcal{S}_{i\text{mod}3} = V_\epsilon$. Then, one of the three sets contains at least a third of the total reward in V_ϵ. Let this be $\mathcal{S}_{k\text{mod}3}$.

We approximate the reward obtained by $\mathcal{O}' = \mathcal{O}_{|\mathcal{S}_{k\text{mod}3}}$ as follows. We construct approximations to the optimal path in sets $S_{3j+k} \in \mathcal{S}_{k\text{mod}3}$, and join them by taking a shortcut across the intermediate sets S_{3j+k-1} and S_{3j+k-2}. In order

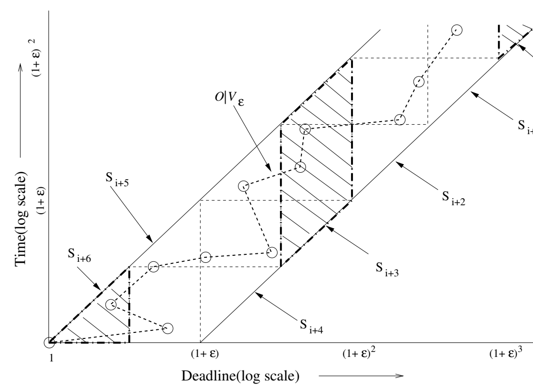


Figure 3: Division of set V_ϵ into segments S_j. Note that segments S_i, S_{i+3} and S_{i+6} are disjoint along the time axis.

to approximate the optimal path in some S_j, we guess the first and the last vertex that $\mathcal{O}'$ visits in this set, and the corresponding times at which it visits them. Then we use the *P2P* algorithm to construct a path of the guessed length. We append these subpaths, with an appropriate amount of "waiting-time" between consecutive segments, so that the resulting path visits segments in the same time interval as $\mathcal{O}$. Lemma 3 shows that the start time of this path is at least f^3 times the release date of any node in S_j, whereas the end time time is at most $\frac{1}{f}$ times the deadline of any node in S_j.

Lemma 3. *For any two nodes $v \in S_j \cap V_\epsilon$, $f^3 R(v) \leq t_{\mathcal{A}}(v) < f D(v)$.*

Proof. Note that by construction, both $\mathcal{A}$ and $\mathcal{O}$ visit vertices in $S_j \cap V_\epsilon$ after the time $f^{j+2} D_{\max}$ and before $f^{j-1} D_{\max}$. Therefore, for all $v \in S_j \cap V_\epsilon$, $R(v) \leq f^{j-1} D_{\max}$. This gives $t_{\mathcal{A}}(v) \geq f^{j+2} D_{\max} \geq f^3 R(v)$. Furthermore, by the definition of S_j, $D(v) < f^j D_{\max}$. Therefore, $t_{\mathcal{A}}(v) \leq f^{j-1} D_{\max} < f D(v)$. $\square$

Finally, we "slow-down" this path by a factor of f^3, that is, if the path output by the algorithm visits a vertex at time

Input: Graph $G = (V, E)$ with deadlines $D(v)$.
Output: Path Q with $\Pi_Q \geq \frac{1}{24}\Pi_{\mathcal{O}}(V_{1/4})$ and $R(v) \leq t_Q(v) \leq D(v)$ for all $v \in Q$, where $V_{1/4} = \{v : t_{\mathcal{O}}(v) \leq \frac{D(v)}{4}\}$.

1. For all i, use the algorithm *P2P* on graph $G(\{r\} \cup S_i)$ to construct a path P_i with length at most α^{i+1}. Let T_i be the corresponding tour of length $2\alpha^{i+1}$.
2. For all $j \in \{0, \cdots, \beta - 1\}$, let Q_j be the concatenation of $P_{\beta i+j}$ for all $i \geq 0$ and let $\pi_j = \sum_i \Pi_{P_{\beta i+j}}(S_{\beta i+j})$.
3. Return the path Q_j, slowed down by a factor of 2, corresponding to the maximum reward π_j over all j.

Figure 4: Algorithm $\mathcal{B}$—Approximation for the large margin case

t, we visit the vertex and then wait until time $\frac{t}{f^3}$ to move to the next vertex. In the worst case, the path constructed as above visits nodes in S_j at $\frac{1}{f^3} \times f^{j-1} D_{\max}$. By the definition of S_j, we know that this is at most $(1+\epsilon)^2$ times the deadline of any node in S_j. Furthermore, this slow-down ensures that the path visits every vertex after its release date. We lose a factor of 3 in reward by leaving out vertices in $\mathcal{S}_{i \bmod 3}, i \neq k$, and another factor of 3 by using algorithm *P2P*. Thus, we get the following theorem. Figure 2 describes the algorithm in detail.

THEOREM 4. *Algorithm $\mathcal{A}$ returns a path P with $\Pi_P^{\epsilon} \geq \frac{1}{9}\Pi_{\mathcal{O}}(V_\epsilon)$.*

4.2 The large margin case

Let $V_{1/4} = \{v : t_{\mathcal{O}}(v) \leq \frac{D(v)}{4}\}$. In this section we will describe an algorithm that collects a constant fraction of the reward $\Pi_{\mathcal{O}}(V_{1/4})$.

For all nodes v, define new deadlines $D'(v) = \frac{D(v)}{4}$. Note that for all $v \in V_{1/4}$, $\mathcal{O}$ visits v before its new deadline $D'(v)$. We divide nodes into subsets as follows. Let $\alpha = 1.2$. For $i \geq 0$, the set S_i contains nodes v that have $D'(v) \in [\alpha^i, \alpha^{i+1})$. Let $\beta = 8$. For $j \in \{0, \cdots, \beta - 1\}$, define $\mathcal{S}_{j \bmod \beta} = \cup_{i \geq 0} S_{\beta i+j}$. Then, $\cup_{j \leq \beta-1} \mathcal{S}_{j \bmod \beta} = V$.

Let P_i be a path returned by the *P2P* algorithm with parameter $D = \alpha^{i+1}$ when applied to the graph induced by $\{r\} \cup S_i$. Note that P_i collects at least $\frac{1}{3}\Pi_{\mathcal{O}}(S_i)$ reward. Let T_i be a tour that starts at the root, follows path P_i and then returns back to the root. For some $j < \beta$, consider the path constructed by appending all $T_{\beta i+j}$ for $i \geq 0$. Assume that the length of T_i is exactly $2\alpha^{i+1}$ (we can ensure this by waiting for an appropriate amount of time at the root between consecutive tours). Let this path be called Q_j.

LEMMA 5. *For any i and j and $v \in S_{\beta i+j}$, $\frac{1}{2}R(v) \leq t_{Q_j}(v) \leq \frac{1}{2}D(v)$.*

PROOF. The length of the path Q_j up to and including the set $S_{\beta i+j}$ is

$$\begin{aligned}
\textstyle\sum_{k<i} |T_{\beta k+j}| + |P_{\beta i+j}| &= \textstyle\sum_{k<i} 2|P_{\beta k+j}| + |P_{\beta i+j}| \\
&= 2\textstyle\sum_{k<i} \alpha^{\beta k+j+1} + \alpha^{\beta i+j+1} \\
&= 2\alpha^{j+1}\frac{\alpha^{\beta i}-1}{\alpha^{\beta}-1} + \alpha^{\beta i+j+1} \\
&\leq \alpha^{\beta i+j+1}\left(\frac{2}{\alpha^{\beta}-1} + 1\right)
\end{aligned}$$

The deadline of v is $D(v) = 4D'(v) \geq 4\alpha^{\beta i+j}$. Thus, $t_{Q_j}(v) \leq \frac{\alpha}{4}(\frac{2}{\alpha^{\beta}-1} + 1)D(v)$. Taking $\alpha = 1.2$ and $\beta = 8$, we get that $t_{Q_j}(v) \leq \frac{1}{2}D(v)$. Similarly, $t_{Q_j}(v) \geq \sum_{k<i} |T_{\beta k+j}| = \frac{2\alpha^{\beta i+j+1}}{\alpha^{\beta}-1}$. Using $\beta = 8$, and $D'(v) \leq \alpha^{\beta i+j+1}$, we get $t_{Q_j}(v) \geq \frac{1}{2}D'(v)$. However we also have $R(v) \leq D'(v)$, because $\mathcal{O}$ collects vertex v before time $D'(v)$. Therefore we get $t_{Q_j}(v) \geq \frac{1}{2}R(v)$. $\square$

Slowing down the path Q_j by a factor of 2 ensures that we cover all nodes within their time window $[R(v), D(v)]$. Note that the sets $\mathcal{S}_{j \bmod \beta}$ for $j \in \{0, \cdots, \beta - 1\}$ together cover all the nodes in $V_{1/4}$. Furthermore, we have $\Pi_{Q_j} \geq \frac{1}{3}\sum_i \Pi_{\mathcal{O}}(S_{\beta i+j})$. Thus, one of the paths Q_j gives a $3\beta = 24$ approximation to the reward collected by $\mathcal{O}$ in $V_{1/4}$. Our algorithm for finding the best Q_j is given in Figure 4.

THEOREM 6. *Algorithm $\mathcal{B}$ returns a path P with $\Pi_P \geq \frac{1}{24}\Pi_{\mathcal{O}}(V_{1/4})$.*

4.3 The general case

Now we will give an algorithm that produces a bicriteria approximation for the entire graph. In particular, given a parameter ϵ, we construct a path that obtains reward at least $\Omega(\log^{-1}\frac{1}{\epsilon})\Pi_{\mathcal{O}}$ if it is allowed to exceed the deadlines by a factor of $1 + \epsilon$. As before, let $f = \frac{1}{\sqrt{1+\epsilon}}$. Let s be defined as the smallest integer for which $f^{(1.5)^s} \leq \frac{1}{4}$. Then $s = O(\log\frac{1}{\epsilon})$.

Our algorithm proceeds as follows. Divide all the nodes into $s + 2$ groups as follows. Group i, $1 \leq i \leq s$, is given by $V_i = \{v : t_{\mathcal{O}}(v) \in (f^{(1.5)^i} D(v), f^{(1.5)^{i-1}} D(v)]\}$. Group 0 is given by $V_0 = \{v : t_{\mathcal{O}}(v) \in (fD(v), D(v)]\}$. Group $s + 1$ is defined as $V_{s+1} = \{v : t_{\mathcal{O}}(v) \in (0, D(v)/4]\}$. These groups together cover all the nodes in $\mathcal{O}$. So, one of the groups contains at least a $\frac{1}{s+2}$ fraction of the total reward collected by $\mathcal{O}$. Let V_i be such a group.

If $i = 0$, we can apply algorithm $\mathcal{A}$ right away and obtain a path P with $\Pi_P \geq \frac{1}{9}\Pi_{\mathcal{O}}(V_0)$ that visits its nodes within a factor of $(1 + \epsilon)$ of their deadlines.

Consider the case when $1 \leq i \leq s$. Scale all the deadlines down by a factor of $f^{(1.5)^{i-1}}$, that is, define $D'(v) = f^{(1.5)^{i-1}} D(v)$. Then the path $\mathcal{O}$ visits all nodes in V_i at time $t_{\mathcal{O}'}(v) \in (f^{0.5(1.5)^{i-1}} D'(v), D'(v)]$. Now apply algorithm $\mathcal{A}$ with parameter $f^{0.5(1.5)^{i-1}}$. Then, the obtained path collects reward $\pi_P \geq \frac{1}{9}\Pi_{\mathcal{O}}(V_i)$ and visits all nodes v before time $D'(v)f^{-(1.5)^{i-1}} = D(v)$.

Finally consider the case when $i = s + 1$. Note that this is the large margin case considered in a previous subsection. So we can use algorithm $\mathcal{B}$ to obtain a 24-approximation in this case.

Putting everything together, we get the following theorem. The algorithm is described in Figure 5.

Input: Graph $G = (V, E)$ with deadlines $D(v)$; parameter ϵ.
Output: Path P with $\Pi_P \geq \Omega(\frac{1}{\log \frac{1}{\epsilon}})\Pi_{\mathcal{O}}$ and $R(v) \leq t_P(v) \leq (1+\epsilon)D(v)$ for all $v \in P$.

1. Let $f = \frac{1}{\sqrt{1+\epsilon}}$ and s be the smallest integer for which $f^{(1.5)^s} \leq \frac{1}{4}$.
2. Apply algorithm $\mathcal{A}$ with parameter f to the graph, and let P_0 be the path obtained.
3. Apply algorithm $\mathcal{B}$ to the graph, and let P_{s+1} be the path obtained.
4. For all $i \in \{1, \cdots, s\}$, do the following:
   (a) For all $v \in V$, define $D'(v) = D(v) f^{(1.5)^{i-1}}$.
   (b) Apply algorithm $\mathcal{A}$ with parameter $f^{(1.5)^{i-1}}$ to the graph with the new deadlines D', and let P_i be the path obtained.
5. Among the paths constructed above, return the one with the maximum reward.

Figure 5: Algorithm $\mathcal{C}$—Bicriteria approximation for the general case

THEOREM 7. *Algorithm $\mathcal{C}$ returns a path P with $\Pi_P^\epsilon \geq \frac{1}{24(s+2)}\Pi_{\mathcal{O}} = \Omega(\frac{1}{\log \frac{1}{\epsilon}})\Pi_{\mathcal{O}}$.*

As a simple corollary of Theorem 7, we get an $O(\log D_{\max})$-approximation to the Time-Window problem without exceeding deadlines.

COROLLARY 8. *Algorithm $\mathcal{C}$ with $\epsilon = 1/D_{\max}$ gives an $O(\log D_{\max})$-approximation to the Time-Window problem.*

PROOF. From Theorem 7, we know that for $\epsilon = 1/D_{\max}$ the path P collects an $\Omega(1/\log D_{\max})$ fraction of the reward from nodes v visited before $(1+\epsilon)D(v)$. For $\epsilon = 1/D_{\max}$, $(1+\epsilon)D(v) < D(v)+1$. Since, all edge lengths and deadlines are integral by assumption, node v is visited by $D(v)$. $\square$

5. AN $O(\log^2 n)$ APPROXIMATION

In this section we give an $O(\log^2 n)$ approximation algorithm for the Time-Window problem. We begin by giving an $O(\log n)$ approximation for Deadline-TSP. Then we describe a general method of obtaining an α^2 approximation for Time-Windows based on an α-approximation for Deadline-TSP, that uses Orienteering as a subroutine. This gives us the desired approximation.

5.1 An $O(\log n)$ approximation for Deadline-TSP

The basic idea behind our algorithm is to prove that there is a segmentation of the graphs into sets $\{V_i\}$ with the following properties. The sets are characterized by deadlines $\{d_i\}$, such that $V_i = \{v \in V : D(v) \in (d_i, d_{i+1}]\}$. There is a path that for all pairs $i < j$ visits vertices in V_i before vertices in V_j, and obtains at least an $\Omega(1/\log n)$ fraction of the optimal reward. Furthermore, this path has the property, that among all vertices that it visits in the set V_i, the vertex with the smallest deadline is visited last. This allows us to use the Orienteering subroutine along with dynamic programming to find the best such path.

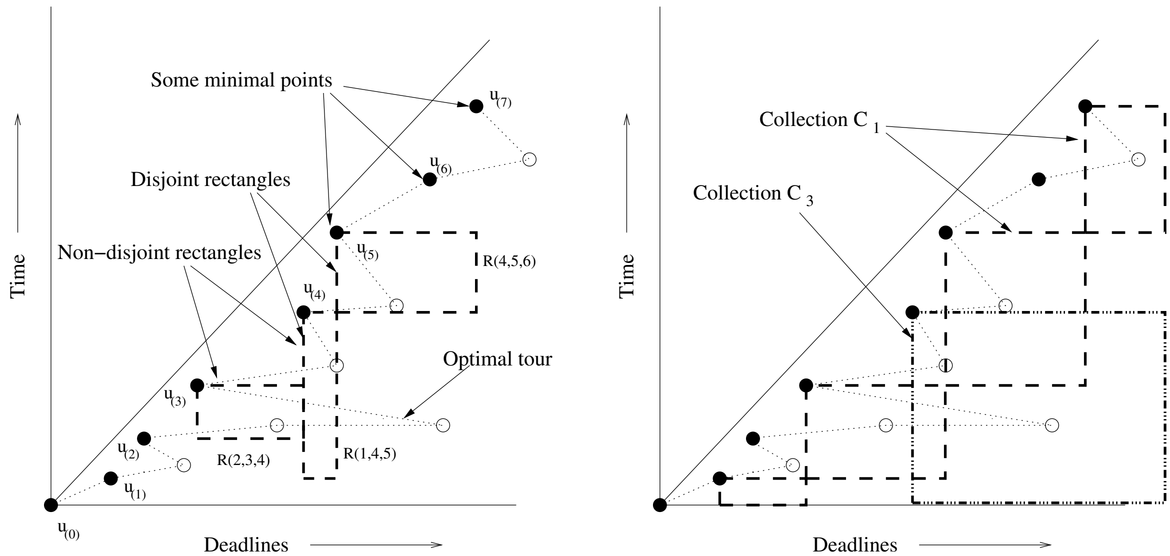
We begin with some notation. Let $r = u_0, u_1, \ldots, u_l$ denote the vertices visited by the optimal path $\mathcal{O}$. It is convenient to view the vertices in $\mathcal{O}$ as lying on a two dimensional plane, with the horizontal and vertical axes corresponding to deadlines and time respectively. That is, vertex u_i lies at the point $p_i = (D(u_i), t_{\mathcal{O}}(u_i))$ (See Figure 6(a)).

We call a vertex u_i *minimal* if for any other vertex u_j, either $t_{\mathcal{O}}(u_j) \leq t_{\mathcal{O}}(u_i)$ or $D(u_j) \geq D(u_i)$. Pictorially, these are vertices that form the upper left envelope of the points p_i. Let $\mathcal{M} = \{u_{(0)}, \ldots, u_{(m)}\}$ denote the set of minimal vertices ordered in increasing order of deadlines, with $|\mathcal{M}| = m+1$. Without loss of generality, we assume that $D(r) = 0$, so $r \in \mathcal{M}$. Similarly, for the purposes of analysis we assume that there is a dummy vertex u_{end} which is at distance $2\sum_{e \in E} \ell(e)$ from the root and has a deadline of $4\sum_{e \in E} \ell(e)$. Without loss of generality we can assume that u_{end} is always visited in the end by any path, and therefore, $u_{end} \in \mathcal{M}$.

For any three minimal vertices $u_{(h)}, u_{(j)}, u_{(k)} \in \mathcal{M}$ with $h \leq j \leq k$, let $\mathcal{R}(h, j, k)$ denote the set of vertices u in $\mathcal{O}$ such that $D(u_{(j)}) \leq D(u) \leq D(u_{(k)})$ and $t_{\mathcal{O}}(u_{(h)}) \leq t_{\mathcal{O}}(u) \leq t_{\mathcal{O}}(u_{(j)})$. Thus, $\mathcal{R}(h, j, k)$ is the set of points lying in a rectangular region defined by $u_{(h)}, u_{(j)}$ and $u_{(k)}$ as shown in Figure 6(a). Note that among all points visited by $\mathcal{O}$ in $\mathcal{R}(h, j, k)$, $u_{(j)}$ is visited last. So all vertices in this rectangle are visited by time $D(u_{(j)})$. This fact will allow us to apply orienteering to points in the rectangle.

Two rectangles $\mathcal{R}_1 = \mathcal{R}(h_1, j_1, k_1)$ and $\mathcal{R}_2 = \mathcal{R}(h_2, j_2, k_2)$ are called *disjoint* if $h_2 \geq j_1$ and $j_2 \geq k_1$, (or equivalently if $h_1 \geq j_2$ and $j_1 \geq k_2$). Pictorially, $\mathcal{R}_1$ and $\mathcal{R}_2$ are disjoint if they are disjoint along *both* the time and deadline axes. Observe that no vertex can lie in two disjoint rectangles. Finally, a collection of rectangles $\mathcal{C} = \{\mathcal{R}_1, \ldots, \mathcal{R}_r\}$ is called disjoint if for all pairs $\mathcal{R}_i, \mathcal{R}_j \in \mathcal{C}$, $\mathcal{R}_i$ and $\mathcal{R}_j$ are disjoint. This notion of disjointness allows us to approximate $\mathcal{O}$ in several rectangles simultaneously and patch the paths—disjointness along deadlines allows us to segment the nodes without double-counting them, while disjointness along the time axis ensures that we do not exceed the length bound.

The main idea behind our $O(\log n)$ approximation is the following. First, we show that there exist $\log n$ collections of disjoint rectangles $\{\mathcal{C}_1, \ldots, \mathcal{C}_{\log n}\}$ such that each vertex $u_i \in \mathcal{O}$ lies in a rectangle contained in at least one $\mathcal{C}_i$. This implies that the total reward contained in this family of collections is at least $\Pi_{\mathcal{O}}$. Therefore, there is some collection $\mathcal{C}_i$ that has at least a $1/\log n$ fraction of this reward. Second, we will give a polynomial time procedure to compute a path that collects at least an $O(1)$ fraction of the reward contained in the best disjoint collection of rectangles. Together, these will imply an $O(\log n)$ approximation.



(a) The 2-dimensional layout and rectangles

(b) Family of disjoint collections of rectangles

Figure 6: An example illustrating the construction of the rectangles $R(h, j, k)$.

We first describe the family $\mathcal{C}_1, \ldots, \mathcal{C}_{\log n}$. Let

$$S_i = \{n_{i,j} = j2^i + 2^{i-1} : 0 \le j \le 2^{\log m - i} - 1\} \cup \{0, m\},$$

where $n_{i,-1} = 0$ and $n_{i,2^{\log m - i}} = m$, by definition. Let

$$\mathcal{C}_i = \{\mathcal{R}(n_{i,j}, n_{i,j+1}, n_{i,j+2})\}_{-1 \le j \le 2^{\log m - i}}$$

So, $\mathcal{C}_1 = \{\mathcal{R}(2j-1, 2j+1, 2j+3) : j = 0, \ldots, m/2-1\}$, $\mathcal{C}_2 = \{\mathcal{R}(4j-2, 4j+2, 4j+6) : j = 0, \ldots, m/4-1\}$ and so on (see Figure 6 (b)). Thus we have a family $\mathcal{F} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{\log m}\}$ consisting of $\log m \le \log n$ collections. Note that for all $j \le m$, at least one set S_i contains j.

LEMMA 9. *For each vertex u visited in the optimum path $\mathcal{O}$, there is at least one collection $\mathcal{C}_i \in \mathcal{F}$, such that u lies in some rectangle in $\mathcal{C}_i$.*

PROOF. Consider a vertex $u \in \mathcal{O}$ and let $u_{(i)}$ be the minimal vertex for which $D(u_{(i)}) \le D(u) < D(u_{(i+1)})$. Likewise, let $u_{(j)}$ be the minimal vertex such that $t_{\mathcal{O}}(u_{(j-1)}) < t_{\mathcal{O}}(u) \le t_{\mathcal{O}}(u_{(j)})$. Observe that $i \ge j$. If $i = j$, then u is the minimal vertex $u_{(i)}$ and in this case, u belongs to the collection $\mathcal{C}_b$ corresponding to an S_b that contains i.

If $i \ne j$, observe that if for some b, S_b contains exactly one number between i and j, say k, then the point u would lie in some rectangle in $\mathcal{C}_b$, in particular, the one corresponding to the number k. Let b be such that $2^b \le i - j < 2^{b+1}$. For this choice of b, either $|S_b \cap [i, j]| = 1$ or $|S_b \cap [i, j]| = 2$. In the first case, we are already done. In the second case, let x and y be the two points in $S_b \cap [i, j]$, then observe that $(x + y)/2 \in S_{b+1}$ and since $i - j < 2^{b+1}$, exactly one point from S_{b+1} lies in the range $[i, j]$. $\square$

Approximating the reward in the best collection. We now show how we can find a path such that the total reward collected in that path is a constant factor of the reward in the best collection.

The idea is the following: Consider the collection $\mathcal{C} \in \mathcal{F}$ that has the maximum reward. Let us consider the optimum path restricted to the vertices in $\mathcal{C}$. Then if we reduce deadlines of nodes in $\mathcal{C}$ as follows — for each vertex v in $\mathcal{R}(i, j, k) \in \mathcal{C}$ we assign a deadline $D'(v) = D(u_{(j)}) \le D(v)$ — then $\mathcal{O}_{|\mathcal{C}}$ still meets all the new deadlines. So, if we could run the point-to-point orienteering subroutine with parameter $D(u_{(j)})$ on the instance restricted to vertices in $\mathcal{R}(i, j, k)$, we would be guaranteed a constant fraction of the reward that $\mathcal{O}$ collects in the rectangle $\mathcal{R}(i, j, k)$. However, we do not know which vertices are minimal, and therefore do not know which vertices lie in $\mathcal{R}(i, j, k)$ (the definition of a minimal vertex depends on when $\mathcal{O}$ visits that vertex). To get around this problem, we can do the following: Observe that the vertices $u_{(j)}$ in $\mathcal{M}$ have increasing deadlines. This allows to write dynamic program in a natural way. We arrange all the vertices of G in the increasing order of deadlines as $v_1, \ldots, v_n$. For every $1 \le j \le k \le n$, we consider the graph $G_{j,k}$ restricted to vertices v_l such that $j \le l < k$, and solve a point to point orienteering instance (assuming some start time to account for distance traveled before considering this interval) such that each vertex is visited before the deadline of v_j. We now give the details.

LEMMA 10. *We can compute in polynomial time a feasible path that collects at least a third of the the reward collected by the best collection $\mathcal{C}_i$.*

PROOF. Let $v_1, \ldots, v_n$ denote the vertices in G in the increasing order of their deadlines. We compute the following quantity: For every pair of vertices v_j, v_k such that $j \le k$, for every vertex v_g and v_h such that $D(v_g), D(v_h) \in [D(v_j), D(v_k))$, and start and finish times t_1 and t_2, such that $t_1 \le t_2 \le D(v_j)$, let $\pi(j, k, g, h, t_1, t_2)$ be the reward collected by the optimum path that begins at v_g at time t_1, finishes at v_h at time t_2 and only visits vertices such that their deadlines are in the interval $[D(v_j), D(v_k))$. To

Input: Graph $G = (V, E)$ with deadlines $D(v)$.
Output: Path P with $\Pi_P \geq \frac{1}{3 \log n} \Pi_{\mathcal{O}}$.

1. Let V_{jk} denote the set of vertices with deadlines between $D(v_j)$ and $D(v_k)$. For all $j \neq k \leq n$, for all $t \leq D(v_j)$, and for all $v_g, v_h \in V_{j,k}$, apply algorithm *P2P* to the graph restricted to V_{jk} with distance bound t, and let $\pi(j, k, g, h, t)$ denote the reward obtained.

2. Let $\pi'(k, t)$ with $t \leq D(v_k)$ denote the (approximately) maximum reward that can be obtained by a path of length t visiting nodes with deadline at most $D(v_k)$.

3. For all $k \leq n$ and $t \leq D(v_k)$, compute $\pi'(k, t)$ and the corresponding path by the following recurrence
$$\pi'(k, t) = \max_{j \leq g, h \leq k, \text{ with } D(v_j) \geq t, t' \leq t} \{\pi'(j, t') + \pi(j, k, g, h, t - t')\}$$

4. Return the path corresponding to maximum reward computed in the last step.

Figure 7: Algorithm $\mathcal{D}$—A $3 \log n$-approximation for Deadline-TSP

compute this approximately (up to a factor of 3), we run the point to point orienteering subroutine on the graph restricted to nodes v such that for $D(v_j) \leq D(v) < D(v_k)$.

Having obtained these $O(n^4 D_{\max}^2)$ quantities, we use a dynamic program to find the maximum reward path obtained by patching the paths corresponding to disjoint intervals $[v_j, v_k]$, computed above. Note that since the vertices are ordered by deadlines, no vertex is double counted in the reward. Moreover the path obtained is feasible, since every vertex v in an interval $[v_j, v_k]$ is visited before $D(v_j)$ and hence before $D(v)$. Finally, observe that any path corresponding to a collection $\mathcal{C}_i$ would be considered by this dynamic program, as this corresponds to patching intervals corresponding to the disjoint rectangles in $\mathcal{C}_i$. Hence modulo the factor 3 that we lose in the point to point orienteering subroutine, we can obtain at least the reward contained in the optimum collection $\mathcal{C}_i$. $\square$

The algorithm is given in Figure 7. By Lemma 9 and 10 we have that,

Theorem 11. *Algorithm $\mathcal{D}$ described in Figure 7 is an $O(\log n)$ approximation algorithm for the Deadline-TSP problem.*

5.2 From Deadlines to Time-Windows

Note that Algorithm $\mathcal{D}$ can be easily modified to return a path of length exactly T for some parameter T — Reduce the deadlines of all nodes with $D(v) > T$ to T and run the algorithm; If the resulting path is shorter than T, wait at the last node for an appropriate amount of time. Likewise, we can modify the algorithm so that the returned path must end at a pre-specified vertex t.

We now show how to use Algorithm $\mathcal{D}$ to solve the Time-Window problem in the special case when all the nodes have a deadline of ∞, but there is a fixed limit T on the length of the path and the path must start from node s and end at node t. We call this problem *Orienteering with Release Dates*.

Given an instance G of Orienteering with Release Dates, we convert the graph G into its "complimentary" graph $G' = (V, E)$, with each $v \in V$ having a time window $[0, T - R(v)]$, where $R(v)$ is the release date of v in G. For any "legal" path P of length T in G that starts at s, ends at t and visits vertices v within their time-windows $[R(v), \infty]$, we can define a legal path P^R in G' that starts at t, ends at s, follows path P in reverse, and visits nodes v within $[0, T - R(v)]$. Therefore, any solution to Orienteering with Release Dates on G can be converted into a solution with the same reward to the Deadline-TSP on G', and vice versa. Therefore, using an α-approximation to the Deadline-TSP, we can obtain an α-approximation to Orienteering with Release Dates. We get the following theorem:

Theorem 12. *Algorithm $\mathcal{D}$ gives a $3 \log n$ approximation to Orienteering with Release Dates, when applied to the complimentary graph.*

Based on the above theorem, our algorithm for the Time-Window problem is simple—run the algorithm $\mathcal{D}$ (Figure 7), replacing the *P2P* subroutine in step 1 by the algorithm for Orienteering with Release Dates. This gives a $3 \log^2 n$-approximation to the Time-Window problem. In fact, an α-approximation to the Deadline-TSP, that uses Orienteering as a subroutine, can be converted to an α^2 approximation to the Time-Window problem using the same technique.

Theorem 13. *The algorithm described above gives a $3 \log^2 n$ approximation to the Time-Window Problem.*

6. CONCLUSION

We present an $O(\log n)$-approximation algorithm for the Deadline-TSP problem and an $O(\log^2 n)$-approximation for the Vehicle Routing Problems with Time Windows, based on an approximation to the point-to-point orienteering problem. We also give a bicriteria approximation for the Deadline-TSP and Time-Windows problems, as well as an improved 3-approximation to the orienteering problem. The main open question is whether it is possible to achieve a constant-factor approximation for either the Deadline-TSP or Time-Window problems. The fact that our bicriteria result has a logarithmic rather than linear dependence on $1/\epsilon$ is promising, but we know of no way to convert it to a constant-factor approximation, and in fact we have a counterexample for our specific algorithm. Interestingly, obtaining a constant factor approximation to the unrooted version of the problem (where the path can begin at any node) is as hard as for the rooted version. Another interesting problem would be to reduce the running time of our algorithms without increasing the approximation factor.

Acknowledgements

This work was supported in part by NSF grants CCR-0105488 and NSF-ITR CCR-0122581.

7. REFERENCES

[1] Bibliography on vehicle routing, http://people.freenet.de/Emden-Weinert/VRP.html.

[2] A. Allahverdi, J. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega, Int. Journal of Management Science*, 27:219–239, 1999.

[3] E. M. Arkin, J. S. B. Mitchell, and G. Narasimhan. Resource-constrained geometric network optimization. In *Symposium on Computational Geometry*, pages 307–316, 1998.

[4] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *Siam J. Computing*, 28(1):254–262, 1999.

[5] R. Bar-Yehuda, G. Even, and S. Shahar. On approximating a geometric prize-collecting traveling salesman. In *Proceedings of the European Symposium on Algorithms*, 2003.

[6] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. In *Proceedings of the 44th Foundations of Computer Science*, 2003.

[7] J. Bruno and P. Downey. Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM Journal on Computing*, 7(4):393–404, 1978.

[8] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications, Manuscript, 2004.

[9] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354, 1992.

[10] M. Desrochers, J. Lenstra, M. Savelsbergh, and F. Soumis. Vehicle routing with time windows: optimization and approximation. In *B.L. Golden, A.A. Assad (eds.). Vehicle Routing: Methods and Studies, North-Holland, Amsterdam*, pages 65–84, 1988.

[11] B. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[12] M. Gravel, W. Price, and C. Gagn. Scheduling jobs in a alcan aluminium factory using a genetic algorithm. *International Journal of Production Research*, 38(13):3031–3041, 2000.

[13] C. Jordan. A two-phase genetic algorithm to solve variants of the batch sequencing problem. *International Journal of Production Research (UK)*, 36(3):745–760, 1998.

[14] M. Kantor and M. Rosenwein. The orienteering problem with time windows. *Journal of the Operational Research Society*, 43:629–635, 1992.

[15] Y. Karuno and H. Nagamochi. A 2-approximation algorithm for the multi-vehicle scheduling problem on a path with release and handling times. In *Proceedings of the European Symposium on Algorithms*, pages 218–229, 2001.

[16] A. Kolen, A. R. Kan, and H. Trienekens. Vehicle routing with time windows. *Operations Research*, 35:266–273, 1987.

[17] M. Savelsbergh. Local search for routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[18] K. Tan, L. Lee, and K. Zhu. Heuristic methods for vehicle routing problem with time windows. In *Proceedings of the 6th AI and Math*, 2000.

[19] S. Thangiah. *Vehicle Routing with Time Windows using Genetic Algorithms, Application Handbook of Genetic Algorithms: New Frontiers, Volume II. Lance Chambers (Ed.).* CRC Press, 1995.

[20] S. R. Thangiah, I. H. Osman, R. Vinayagamoorthy, and T. Sun. Algorithms for the vehicle routing problems with time deadlines. *American Journal of Mathematical and Management Sciences*, 13(3& 4):323–355, 1993.

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

[22] J. Wisner and S. Siferd. A survey of u.s. manufacturing practices in make-to-order machine shops. *Production and Inventory Management Journal*, 1:1–7, 1995.

[23] W. Yang and C. Liao. Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155, 1999.