

Approximation Algorithms for Orienteering with Time Windows

Chandra Chekuri*

Nitish Korula†

February 16, 2008

Abstract

Orienteering is the following optimization problem: given an edge-weighted graph (directed or undirected), two nodes s, t and a time limit T , find an s - t walk of total length at most T that maximizes the number of *distinct* nodes visited by the walk. One obtains a generalization, namely orienteering with *time-windows* (also referred to as TSP with time-windows), if each node v has a specified time-window $[R(v), D(v)]$ and a node v is counted as visited by the walk only if v is visited during its time-window. For the time-window problem, an $O(\log \text{OPT})$ approximation can be achieved even for directed graphs if the algorithm is allowed *quasi-polynomial* time. However, the best known polynomial time approximation ratios are $O(\log^2 \text{OPT})$ for undirected graphs and $O(\log^4 \text{OPT})$ in directed graphs. In this paper we make some progress towards closing this discrepancy, and in the process obtain improved approximation ratios in several natural settings.

Let $L(v) = D(v) - R(v)$ denote the length of the time-window for v and let $L_{\max} = \max_v L(v)$ and $L_{\min} = \min_v L(v)$. Our results are given below with α denoting the known approximation ratio for orienteering (without time-windows). Currently $\alpha = (2 + \epsilon)$ for undirected graphs and $\alpha = O(\log^2 \text{OPT})$ in directed graphs.

- An $O(\alpha \log L_{\max})$ approximation when $R(v)$ and $D(v)$ are integer valued for each v .
- An $O(\alpha \max\{\log \text{OPT}, \log \frac{L_{\max}}{L_{\min}}\})$ approximation.
- An $O(\alpha \log \frac{L_{\max}}{L_{\min}})$ approximation when no start and end points are specified.

In particular, if $\frac{L_{\max}}{L_{\min}}$ is poly-bounded, we obtain an $O(\log n)$ approximation for the time-window problem in undirected graphs.

1 Introduction

In the orienteering problem, we are given an edge-weighted graph $G(V, E)$, two vertices $s, t \in V$, and a time limit T . The goal is to find a walk that begins at s at time 0, reaches t before time T , and visits as many vertices as possible. (The weight or length of an edge denotes the time taken to travel from one endpoint to the other.) Note that a vertex may be visited many times, but is only counted once in the objective function. In other words, the goal is to find an s - t walk of total length at most T that maximizes the number of distinct vertices visited by the walk. This problem is also referred to as the *point-to-point* orienteering problem to distinguish it from two special cases: only the start vertex s is specified, or neither s nor t are specified. Here we consider a more general problem, namely orienteering with time-windows. In this problem, we are additionally given a time-window (or interval) $[R(v), D(v)]$ for each vertex v . A vertex is counted as visited only if the walk visits v at some time $t \in [R(v), D(v)]$. (If a vertex v is reached before $R(v)$, we may choose to “wait” at v

*Dept. of Computer Science, University of Illinois, Urbana, IL 61801. Partially supported by NSF grants CCF 07-28782 and CNS-0721899, and a US-Israeli BSF grant 2002276. chekuri@cs.uiuc.edu

†Dept. of Computer Science, University of Illinois, Urbana, IL 61801. Partially supported by NSF grant CCF 07-28782. nkorula2@uiuc.edu

	Points in Euclidean Space	Undirected Graphs	Directed Graphs
Orienteering	$(1 + \epsilon)$ [9]	$(2 + \epsilon)$ [6]	$O(\log^2 \text{OPT})$ [14, 6]
ORIENT-DEADLINE	$O(\log \text{OPT})$	$O(\log \text{OPT})$ [2]	$O(\log^3 \text{OPT})$
ORIENT-TW	$O(\log^2 \text{OPT})$	$O(\log^2 \text{OPT})$ [2]	$O(\log^4 \text{OPT})$

Table 1: Known approximation ratios for orienteering and orienteering with time-windows. Entries without a citation come from combining results for orienteering with the results from [2] in a black-box fashion.

until $R(v)$, so the walk can obtain credit for v , and then resume the walk. The time spent “waiting” is included in the length of the walk.) For ease of notation, we use ORIENT-TW to refer to the problem of orienteering with time-windows. A problem of intermediate complexity is the one in which $R(v) = 0$ for all v . We refer to this problem as orienteering with deadlines (ORIENT-DEADLINE); it has also been called the Deadline-TSP problem in [2]. The problem where vertices have release times but not deadlines (that is, $D(v) = \infty$ for all v) is entirely equivalent to ORIENT-DEADLINE.

The orienteering problem and the time-window generalization are intuitively appealing variants of TSP. They also arise in practical applications of vehicle routing and scheduling [15]. Even in undirected graphs these problems are NP-Hard, and also APX-hard [4]. In fact, ORIENT-TW is NP-hard even on the line [16]. Although these problems are natural and simple to state, the first non-trivial approximation algorithm for undirected graphs appeared only a few years ago [4], and the first polynomial time approximation algorithms for directed graphs appeared within the last year [14, 6]; earlier, a constant factor approximation was known for points in the Euclidean plane [1]. Table 1 summarizes the best known approximation ratios.

In [8], a recursive greedy algorithm is given for the orienteering problem when the reward function is a given monotone submodular set function¹ f on V , and the objective is to maximize $f(S)$ where S is the set of vertices visited by the walk. Several non-trivial problems, including ORIENT-TW, can be captured by using different submodular functions. The algorithm from [8] provides an $O(\log \text{OPT})$ approximation in directed graphs, but it runs in *quasi-polynomial* time. Thus, we make the following natural conjecture:

Conjecture 1 *There is a polynomial time $O(\log \text{OPT})$ approximation for orienteering with time-windows in directed (and undirected) graphs.*

As can be seen from Table 1, even in undirected graphs the current best ratio is $O(\log^2 \text{OPT})$. Our primary motivation is to close the gap between the ratios achievable in polynomial and quasi-polynomial time respectively. We remark that the quasi-polynomial time algorithm in [8] is quite different from all the other polynomial time algorithms, and it does not appear easy to find a polynomial time equivalent. In this paper we make some progress in closing the gap, while also obtaining some new insights. An important aspect of our approach is to understand the complexity of the problem in terms of the maximum and minimum time-window lengths. Let $L(v) = D(v) - R(v)$ be the length of the time-window of v . Let $L_{\max} = \max_v L(v)$ and $L_{\min} = \min_v L(v)$. Our results depend on the ratio $L = L_{\max}/L_{\min}$.² We define this parameter following the work of Frederickson and Wittman [10]; they showed that a constant factor approximation is achievable in undirected graphs if all time-windows are of the same length (that is, $L = 1$) and the end points of the walk are not specified. We believe this is a natural parameter to consider in the context of time-windows. In many practical settings L is likely to be small, and hence, algorithms whose performance depends on L may be better than those that depend on other parameters. In [2], an $O(\log D_{\max})$ approximation is given for ORIENT-TW in

¹A function $f : 2^V \rightarrow \mathcal{R}^+$ is a monotone submodular set function if f satisfies the following properties: (i) $f(\emptyset) = 0$, $f(A) \geq f(B)$ for all $A \subseteq B$ and (ii) $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$.

²If $L_{\min} = 0$, consider the set of vertices which have zero-length time-windows. If this includes a significant fraction of the vertices of an optimal solution, use dynamic programming to get a $O(1)$ -approximation. Otherwise, we can ignore these vertices and assume $L_{\min} > 0$ without losing a significant fraction of the optimal reward.

undirected graphs where $D_{\max} = \max_v D(v)$ and only the start vertex s is specified (here it is assumed that all the input is integer valued). We believe that L_{\max} is a better measure than D_{\max} for ORIENT-TW; note that $L_{\max} \leq D_{\max}$ for all instances.

Results: We obtain results for both undirected and directed graphs. Our results are for ORIENT-TW and use an algorithm for the point-to-point orienteering problem as a black box. Letting α denote the approximation ratio for the orienteering problem, we have the following results.

- An $O(\alpha \log L_{\max})$ approximation when $R(v)$ and $D(v)$ are integer valued for each v .
- An $O(\alpha \max\{\log \text{OPT}, \log L\})$ approximation.
- An $O(\alpha \log L)$ approximation when no start and end points are specified.

We briefly compare our results to previous results to put the improvements in a proper context. We focus on undirected graphs where $\alpha = O(1)$. The $O(\log L_{\max})$ approximation improves on the $O(\log D_{\max})$ approximation from [2] in several ways. First, $L_{\max} \leq D_{\max}$ in all instances and is considerably smaller in many instances. Second, our algorithm applies to directed graphs while the algorithm in [2] is applicable only for undirected graphs. Third, our algorithm is for the point-to-point version while the one in [2] does not guarantee that the walk ends at t . The $O(\max\{\log \text{OPT}, \log L\})$ ratio improves the $O(\log^2 \text{OPT})$ approximation in [2] on instances where L is not too large; in particular if L is poly-bounded then the ratio we obtain is $O(\log n)$ while the previous guarantee is $O(\log^2 n)$ when expressed as a function of n , the number of vertices in G . Finally, our bound of $O(\log L)$ strictly generalizes the result of [10], which considers only the case of $L = 1$.

Our results are obtained using relatively simple ideas. Nevertheless, we believe that they are interesting, useful, and shed more light on the complexity of the problem. In particular we are optimistic that some of these ideas will lead to an $O(\log n)$ approximation for the time-window problem in undirected graphs even when L is not poly-bounded.

Related Work: Table 1 refers to a good portion of the recent work on approximation algorithms for orienteering and related problems. The first non-trivial approximation algorithm for orienteering was a $(2 + \epsilon)$ -approximation in the Euclidean plane [1]. In [4], the authors showed how one can use an approximation for the k -stroll problem (here, the goal is to find a minimum length s - t walk that visits k vertices) to obtain an approximation for orienteering. In undirected graphs an approximation for k -stroll can be obtained using an approximation for the more well-studied k -MST problem although one can obtain improved ratios for k -stroll using related ideas [5]. In [2], orienteering is used as a black box for ORIENT-DEADLINE and ORIENT-TW. For directed graphs a bi-criteria approximation for k -stroll was only recently obtained [14, 6] and this led to the first poly-logarithmic approximation for orienteering and ORIENT-TW. The recursive greedy algorithm from [8], as discussed before, is based on a different approach. Other special cases have been considered in the literature. For points on a line, an $O(\min\{\log n, \log L\})$ approximation is given in [3]. When the number of distinct time-windows is a fixed constant, [7] gives an $O(\alpha)$ approximation; here α is the approximation ratio for orienteering. As we already mentioned, [10] considered the case of equal length time-windows.

2 Preliminaries and General Techniques

Much of the prior work on orienteering with time-windows, following [2], has used the same general technique or can be cast in this framework: Use combinatorial methods to reduce the problem to a collection of sub-problems where the time-windows can be ignored. Each sub-problem has a subset of vertices V' , start and end vertices $s', t' \in V'$, and a time-interval I in which we must travel from s' to t' , visiting as many vertices of V' within their time windows as possible. However, the sub-problem is constructed such that the time-window for every vertex in V' entirely contains the interval I . Therefore, the sub-problem is really an instance of the

orienteering problem (without time-windows). An approximation algorithm for orienteering can be used to solve each sub-problem, and these solutions can be pasted together using dynamic programming. The next two sub-sections describe this framework.

One consequence of using this general method is that the techniques we develop apply to both directed and undirected graphs; while solving a sub-problem we use either the algorithm for orienteering on directed graphs, or the algorithm for undirected graphs. Better algorithms for either of these problems would immediately translate into better algorithms for orienteering with time-windows.

Subsequently, we mainly study undirected graphs, and state our results in that context. The corresponding approximation ratios for directed graphs are a factor of $O(\log^2 \text{OPT})$ higher; this is simply because $O(\log^2 \text{OPT})$ is the ratio between the current best approximations for orienteering (without time-windows) in directed and undirected graphs.

Recall that L_{\max} and L_{\min} are the lengths of the longest and shortest time windows respectively, and L is the ratio $\frac{L_{\max}}{L_{\min}}$. We first provide two algorithms with the following guarantees:

- $O(\log L_{\max})$, if the release time and deadline of every vertex are integers.
- $O(\log n)$, if $L \leq 2$.

The second algorithm immediately leads to a $O(\log n \times \log L)$ -approximation for the general time-window problem, which is already an improvement on $O(\log^2 n)$ when the ratio L is small. However, we can combine the first and second algorithms to obtain a $\max\{O(\log n), O(\log L)\}$ -approximation for orienteering with time-windows.

Throughout this paper, we use $R(v)$ and $D(v)$ to denote (respectively) the release time and deadline of a vertex v . We also use the word *interval* to denote a time window; $I(v)$ denotes the interval $[R(v), D(v)]$. Typically, we use ‘time-window’ when we are interested in the start and end points of a window, and ‘interval’ when we think of a window as an interval along the ‘time axis’.

For any instance X of ORIENT-TW, we let $\text{OPT}(X)$ denote the reward collected by an optimal solution for X . When the instance is clear from context, we use OPT to denote this optimal reward.

2.1 The General Framework

As described at the beginning of section 2, our general method to solve ORIENT-TW is to reduce the problem to a set of sub-problems without time-windows. Given an instance of ORIENT-TW on a graph $G(V, E)$, suppose V_1, V_2, \dots, V_m partition V , and we can associate times R_i and D_i with each V_i such that each of the following conditions holds:

- For each $v \in V_i$, $R(v) \leq R_i$ and $D(v) \geq D_i$.
- For $1 \leq i < m$, $D_i < R_{i+1}$.
- An optimal solution visits any vertex in V_i during $[R_i, D_i]$.

Then, we can solve an instance of the orienteering problem in each V_i separately, and combine the solutions using dynamic programming. The approximation ratio for such ‘‘composite’’ solutions would be the same as the approximation ratio for the orienteering problem. We refer to an instance of ORIENT-TW in which we can construct such a partition of the vertex set (and solve the sub-problems separately) as a *modular instance*. Subsection 2.2 describes a dynamic program that can solve modular instances.

Unfortunately, given an arbitrary instance of ORIENT-TW, it is unlikely to be a modular instance. Therefore, we define restricted versions of a given instance:

Definition 2.1 Let A and B be instances of the time-window problem on the same underlying graph (with the same edge-lengths), and let $I_A(v)$ and $I_B(v)$ denote the intervals for vertex v in instances A and B respectively. We say that B is a restricted version of A if, for every vertex v , $I_B(v)$ is a sub-interval of $I_A(v)$.

Clearly, a walk that gathers a certain reward in a restricted version of an instance will gather at least that reward in the original instance. We attempt to solve ORIENT-TW by constructing a set of restricted versions that are easier to work with. Typically, the construction is such that the reward of an optimal solution in at least one of the restricted versions is a significant fraction of the reward of an optimal solution in the original instance. Hence, an approximation to the optimal solution in the ‘best’ restricted version leads us to an approximation for the original instance.

This idea leads us to the next proposition, the proof of which is straightforward, and hence omitted.

Proposition 2.2 Let A be an instance of ORIENT-TW on a graph $G(V, E)$. If B_1, B_2, \dots, B_β are restricted versions of A , and for all vertices $v \in V$, $I_A(v) = \bigcup_{1 \leq i \leq \beta} I_{B_i}(v)$, there is some B_j such that $\text{OPT}(B_j) \geq \frac{\text{OPT}(A)}{\beta}$.

The restricted versions we construct will usually be modular instances of ORIENT-TW. Therefore, the general algorithm for ORIENT-TW is:

1. Construct a set of β restricted versions of the given instance; each restricted version is a modular instance.
2. Pick the best restricted version (enumerate over all choices), find an appropriate partition, and use an α -approximation for orienteering together with dynamic programming to solve that instance.

It follows from the previous discussion that this gives a $(\alpha \times \beta)$ -approximation for ORIENT-TW. We next describe how to solve modular instances of ORIENT-TW.

2.2 A dynamic program for modular instances

Recall that a modular instance is an instance of ORIENT-TW on a graph $G(V, E)$ in which the vertex set V can be partitioned into V_1, V_2, \dots, V_m , such that an optimal solution visits vertices of V_i after time R_i and before D_i . For any vertex $v \in V_i$, $R(v) \leq R_i$ and $D(v) \geq D_i$. Further, vertices of V_i are visited before vertices of V_j , for all $j > i$.

To solve a modular instance, for each V_i we could ‘guess’ the first and last vertex visited by an optimal solution, and guess the times at which this solution visits the first and last vertex. If α is the approximation ratio of an algorithm for orienteering, we find a path in each V_i that collects an α -fraction of the optimal reward, and combine these solutions.

More formally, we use the following dynamic program: For any $u, v \in V_i$, consider the graph induced by V_i , and let $\text{OPT}(u, v, t)$ denote the optimal reward collected by any walk from u to v of length at most t (ignoring time-windows). Now, define $\Pi_i(v, T)$ for $v \in V_i, R_i \leq T \leq D_i$ as the optimal reward collected by any walk in G that begins at s at time 0, and ends at v at time T . Given $\text{OPT}(u, v, t)$, the following recurrence allows us to easily compute $\Pi_i(v, T)$:

$$\Pi_i(v, T) = \max_{u \in V_i, w \in V_{i-1}, t \leq T - R_i} \text{OPT}(u, v, t) + \Pi_{i-1}(w, T - t - d(w, u)).$$

Of course, we cannot exactly compute $\text{OPT}(u, v, t)$; instead, we use an α -approximation algorithm for orienteering to compute an approximation to $\text{OPT}(u, v, t)$ for all $u, v \in V_i, t \leq D_i - R_i$. This gives an α -approximation to $\Pi_i(v, T)$ using the recurrence above.

Unfortunately, the running time of this algorithm is polynomial in T ; this leads to a pseudo-polynomial algorithm. To obtain a polynomial-time algorithm, we use a standard technique of dynamic programming based on reward instead of time (see [4], [8]). Using standard scaling tricks for maximization problems, one can reduce the problem with arbitrary rewards on the vertices to the problem where the reward on each vertex is 1; the resulting loss in approximation can be made $(1 + o(1))$.

To construct a dynamic program based on reward instead of time, we “guess” the reward k_i collected by an optimal solution in each V_i . However, it is important that we do not try to find an (approximately) shortest path in each V_i that collects reward k_i , since taking slightly too much time early on can have bad consequences for later V_i s. Instead, we use binary search to compute the shortest walk we can find that collects reward at least k_i/α ; this walk is guaranteed to be no longer than the optimal walk that collects reward k_i from V_i . We then combine the solutions from each V_i using a dynamic program very similar to the one described above for times.³ We omit details in this version.

3 The Algorithms

In this section, we use the techniques described above to develop algorithms which achieve approximation ratios depending on the lengths of the time-windows. We first consider instances where all time-windows have integral end-points, and then instances for which the ratio $L = \frac{L_{\max}}{L_{\min}}$ is bounded. Finally, we combine these ideas to obtain a $\max\{O(\log n), O(\log L)\}$ -approximation for all instances of ORIENT-TW.

3.1 An $O(\log L_{\max})$ -approximation

We now focus on instances of ORIENT-TW in which, for all vertices v , $R(v)$ and $D(v)$ are integers. Our algorithm requires the following simple lemma:

Lemma 3.1 *Any interval of length $M > 1$ with integral endpoints can be partitioned into at most $2 \log M$ disjoint sub-intervals, such that the length of any sub-interval is a power of 2, and any sub-interval of length 2^i begins at a multiple of 2^i . Further, there are at most 2 sub-intervals of each length.*

Proof: Use induction on the length of the interval. The lemma is clearly true for intervals of length 2 or 3. Otherwise, use at most 2 sub-intervals of length 1 at the beginning and end of the given interval, so that the residual interval (after the sub-intervals of size 1 are deleted) begins and ends at an even integer. To cover the residual interval, divide all integers in the (residual) problem by 2, and apply the induction hypothesis; we use at most $2 + (2 \log M/2) \leq 2 \log M$ sub-intervals in total. It is easy to see that we use at most 2 sub-intervals of each length; intervals of length 2^i are used at the $(i + 1)$ th level of recursion. \square

For ease of notation, we let ℓ denote $\log L_{\max}$ for the rest of this sub-section. Given an instance of ORIENT-TW, for each vertex v with interval $I(v)$, we use Lemma 3.1 to partition $I(v)$ into at most 2ℓ sub-intervals. We label the sub-intervals of $I(v)$ as follows: For each $1 \leq i \leq \ell$, the first sub-interval of length 2^i is labeled $I_i^1(v)$ and the second sub-interval $I_i^2(v)$. (Note that there may be no sub-intervals of length 2^i .)

We now construct a set of at most 2ℓ restricted versions of the given instance. We call these restricted versions $B_1^1, B_2^1, \dots, B_\ell^1$ and $B_1^2, B_2^2, \dots, B_\ell^2$, such that the interval for vertex v in B_i^b is $I_i^b(v)$. If $I_i^b(v)$ was not an interval used in the partition of $I(v)$, v is not present in the restricted version. (Equivalently, it has reward 0 or an empty time-window.)

Consider an arbitrary restricted instance B_i^b . All vertices in this instance of ORIENT-TW have intervals of length 2^i , and all time-windows begin at an integer that is a multiple of 2^i . Hence, any 2 vertices either have time-windows that are identical, or entirely disjoint. This means that B_i^b is a modular instance, so we can

³Recall that a walk is allowed to spend time “waiting” at a vertex v ; without this ability, we cannot combine solutions from each V_i using this reward-based dynamic program.

break it into sub-problems, and use a $(2 + \epsilon)$ -approximation to orienteering in the sub-problems to obtain a $(2 + \epsilon)$ -approximation for the restricted instance.

By Proposition 2.2, one of the restricted versions has an optimal solution that collects reward at least $\frac{\text{OPT}}{2\ell}$. Using a $(2 + \epsilon)$ -approximation for this restricted version gives us a $(2 + \epsilon) \times 2\ell = O(\log L_{max})$ -approximation for ORIENT-TW when all interval endpoints are integers.

3.2 A $O(\log n)$ -approximation when $L \leq 2$

For an instance of ORIENT-TW when $L = \frac{L_{max}}{L_{min}} \leq 2$, we begin by scaling all times so that $L_{min} = 1$ (and so $L_{max} \leq 2$). Note that even if all release times and deadlines were integral prior to scaling, they may not be integral in the scaled version.

For each vertex v , we partition $I(v) = [R(v), D(v)]$ into 3 sub-intervals: $I_1(v) = [R(v), a]$, $I_2(v) = [a, b]$, and $I_3(v) = [b, D(v)]$, where $a = \lfloor R(v) + 1 \rfloor$ (that is, the next integer strictly greater than the release time) and $b = \lceil D(v) - 1 \rceil$ (the greatest integer strictly less than the deadline). The figure below illustrates the partitioning of intervals. Note that $I_2(v)$ may be a point, and in this case, we ignore such a sub-interval.

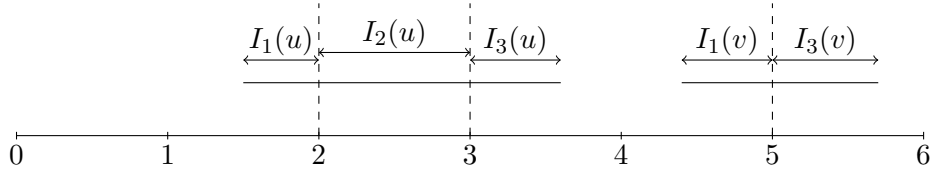


Figure 1: We illustrate the partitioning of 2 intervals into sub-intervals. Note that on the right, $I_2(v)$ is empty.

We now construct 3 restricted versions of the given instance — B_1 , B_2 , and B_3 — such that the interval for any vertex v in B_i is simply $I_i(v)$. By Proposition 2.2, one of these has an optimal solution that collects at least a third of the reward collected by an optimal solution to the original instance. Suppose this is B_2 . All time-windows have length exactly 1, and start and end-points are integers. Therefore, B_2 is a modular instance, and we can get a $(2 + \epsilon)$ -approximation to the optimal solution in B_2 ; this gives a $(6 + \epsilon)$ -approximation to the original instance.

Dealing with B_1 and B_3 is not nearly as easy; they are not quite modular. Every interval in B_1 has length at most 1, and ends at an integer; for B_3 , intervals have length at most 1 and start at an integer. We illustrate how to approximate a solution for B_3 within a factor of $O(\log n)$; the algorithm for B_1 is identical except that release times and deadlines are to be interchanged.

For B_3 , we can partition the vertex set into V_1, V_2, \dots, V_m , such that all vertices in V_i have the same (integral) release time, and any vertex in V_i is visited before any vertex in V_j for $j > i$. Figure 2 shows such a partition. The deadlines for vertices in V_i may be all distinct. However, we can solve an instance of ORIENT-DEADLINE in each V_i separately, and paste the solutions together using dynamic programming. The solution we obtain will collect at least $\Omega(1/\log n)$ of the reward of an optimal solution for B_3 , since there is a $O(\log n)$ -approximation for ORIENT-DEADLINE ([2]). Therefore, this gives us a $3 \times O(\log n) = O(\log n)$ -approximation to the original instance.

Similarly, we can obtain a $O(\log n)$ -approximation for B_1 using the $O(\log n)$ -approximation algorithm for orienteering with release times. Therefore, when $L \leq 2$, we have a $O(\log n)$ -approximation for ORIENT-TW.

3.3 Putting the pieces together

An arbitrary instance of ORIENT-TW may have $L > 2$, and interval end-points may not be integers. However, we can combine the algorithms from the two preceding sections to deal with such instances. We begin by

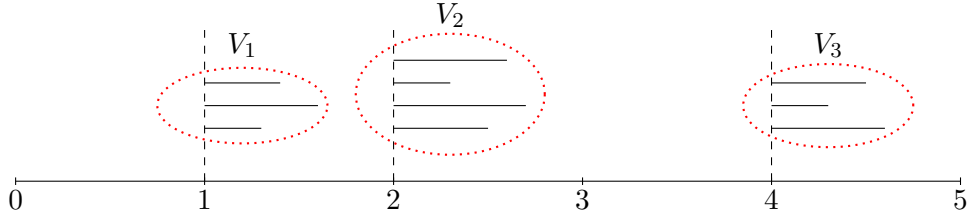


Figure 2: In B_3 , all time-windows start at an integer and have length at most 1. Each set of vertices whose windows have a common beginning corresponds to a sub-problem that is an instance of orienteering with deadlines.

scaling all times such that the shortest interval has length 1; the longest interval now has length $L = \frac{L_{\max}}{L_{\min}}$, where L_{\max} and L_{\min} are the lengths of the longest and shortest intervals in the original instance.

We now construct 3 restricted versions of the scaled instance: B_1 , B_2 , and B_3 . For any vertex v with interval $[R(v), D(v)]$ in the scaled instance, we construct 3 sub-intervals. $I_1(v) = [R(v), a]$, $I_2(v) = [a, b]$, and $I_3(v) = [b, D(v)]$, where $a = \lceil R(v) + 1 \rceil$ and $b = \lfloor D(v) - 1 \rfloor$. As before, the interval for v in the instance B_i is $I_i(v)$.

One of the restricted versions collects at least a third of the reward of the original instance. Suppose this is B_1 or B_3 . All intervals in B_1 and B_3 have length between 1 and 2 by our construction. Therefore, we can use the $O(\log n)$ -approximation algorithm from section 3.2 to collect at least $\Omega(1/\log n)$ of the reward of an optimal solution to the original instance. It now remains only to consider the case that B_2 collects more than a third of the reward. In B_2 , the end-points of all time-windows are integral, and the longest interval has length less than L . We can now use the algorithm of section 3.1 to obtain a $O(\log L)$ -approximation.

Therefore, our combined algorithm is a $\max\{O(\log n), O(\log L)\}$ -approximation for ORIENT-TW.

3.4 Towards a better approximation, and arbitrary endpoints

In the previous sub-section, we obtained an approximation ratio of $\max\{O(\log n), O(\log L)\}$; we would like to improve this ratio to $O(\log n)$. Unfortunately, it does not seem easy to do this directly. A natural question, then, would be to obtain a ratio of $O(\log L)$; this is equivalent to a constant-factor approximation for the case when $L \leq 2$. However, this is no easier than finding a $O(\log n)$ -approximation for arbitrary instances of ORIENT-TW, as we show in the next proposition.

Proposition 3.2 *A constant-factor approximation algorithm for ORIENT-TW with $L \leq 2$ implies a $O(\log n)$ -approximation for arbitrary time-windows.*

Proof: We show that a constant-factor approximation when $L \leq 2$ implies a constant-factor approximation for ORIENT-DEADLINE. It follows from an algorithm of [2] that we can then obtain a $O(\log n)$ -approximation for ORIENT-TW.

Given an arbitrary instance of ORIENT-DEADLINE on graph $G(V, E)$, we add a new start vertex s' to G . Connect s' to s with an edge of length $D_{\max} = \max_v D(v)$. The release time of every vertex is 0, but all deadlines are increased by D_{\max} . Observe that all vertices have time-windows of length between D_{\max} and $2D_{\max}$, so $L \leq 2$. It is easy to see that given any walk beginning at s in the original instance, we can find an equivalent walk beginning at s' in the modified instance that visits a vertex in its time-window iff the original walk visited a vertex before its deadline in the given instance, and vice versa. Therefore, a constant-factor approximation for the modified instance of ORIENT-TW gives a constant-factor approximation for the original instance of ORIENT-DEADLINE. \square

We *can*, however, obtain a constant-factor approximation for ORIENT-TW when $L \leq 2$ if we remove the restriction that the walk must start and end at s and t , the specified endpoints. The algorithm of Frederickson and Wittman [10] for the case of $L = 1$ can be adapted relatively easily to give a constant-factor approximation for $L \leq 2$. For completeness, we sketch the algorithm here.

We construct 5 restricted versions B_1, \dots, B_5 , of a given instance A . For every vertex v , we create at most 5 sub-intervals of $I(v)$ by breaking it at every multiple of 0.5. (For instance $[3.7, 5.6]$ would be broken up into $[3.7, 4]$, $[4, 4.5]$, $[4.5, 5]$, $[5, 5.5]$, $[5.5, 5.6]$. Note that some intervals may have fewer than 5 sub-intervals.) The interval for v in $B_1(v)$ is the first sub-interval, and the interval in $B_5(v)$ is the last sub-interval, regardless of whether $I(v)$ has 5 sub-intervals. B_2, B_3 , and B_4 each use one of any remaining sub-intervals.

B_2, B_3 , and B_4 are modular instances, so if one of them is the best restricted version of A , we can use a $(2 + \epsilon)$ -approximation for orienteering to get reward at least $\frac{\text{OPT}(A)}{10+\epsilon}$. Exactly as in subsection 3.2, B_1 and B_5 are not quite modular instances; in B_1 , all deadlines are half-integral but release times are arbitrary, and in B_5 , all release times are half-integral, but deadlines are arbitrary.

Suppose that B_1 is the best restricted version. The key insight is that if the optimal walk in B_1 collects a certain reward starting at s at time 0, there is a walk in B_2 starting at s at time 0.5 that collects *the same* reward. (This is the substance of Theorem 1 of [10].) Therefore, if B_1 is the best restricted version, we find a $(2 + \epsilon)$ -approximation to the best walk in B_2 starting at s at time 0.5; we are guaranteed that this walk collects reward at least $\frac{\text{OPT}(A)}{10+\epsilon}$. Note that this walk may not reach the destination vertex t by the time limit, since we start 0.5 time units late. Similarly, if B_5 is the best restricted version, we can find a walk in B_4 that collects reward $\frac{\text{OPT}(A)}{10+\epsilon}$ while beginning at s at time -0.5 . (To avoid negative times, we can begin the walk at s' at time 0, where s' is the first vertex visited by the original walk after time 0.) This walk is guaranteed to reach t by the time limit, but does not necessarily begin at s .

Therefore, this algorithm is a $O(1)$ -approximation when $L \leq 2$, or a $O(\log L)$ -approximation for general instances of ORIENT-TW with arbitrary endpoints. We note that one *cannot* use this with Proposition 3.2 to get a $O(\log n)$ -approximation for ORIENT-TW with arbitrary endpoints. The dynamic program for modular instances crucially uses the fact that we can specify both endpoints for the sub-problems.

4 Conclusions

We conclude with some open problems:

- Can we obtain an improved approximation ratio for the general time-window problem that does not depend on the lengths of the intervals? In particular, is there an $O(\log n)$ (or even an $O(\log \text{OPT})$) approximation for undirected graphs?
- The current algorithms for ORIENT-TW use the orienteering algorithm in a black-box fashion. For directed graphs the current ratio for orienteering is $O(\log^2 \text{OPT})$, and hence the ratio for ORIENT-TW is worse by additional logarithmic factors. Can we avoid using orienteering in a black-box fashion? We note that the quasi-polynomial time algorithm of [8] has the same approximation ratio of $O(\log \text{OPT})$ for both the basic orienteering and time-window problems in directed graphs. In fact, the algorithm gives the same approximation ratio even for the more general problem where each vertex has multiple disjoint time windows in which it can be visited.
- For many applications, each vertex has multiple disjoint time-windows, and we receive credit for a vertex if we visit it within any of its windows. If each vertex has at most k windows, a naïve algorithm loses an extra factor of k beyond the ratio for ORIENT-TW, but no better approximation is known. Any non-trivial result would be of interest.

Acknowledgments: We thank Pratik Worah for several discussions of algorithms and hardness results for orienteering with time-windows and other variants.

References

- [1] E. Arkin, J. Mitchell, and G. Narasimhan. Resource-constrained geometric network optimization. *Proc. of ACM SoCG*, 307–316, 1998.
- [2] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows. *Proc. of ACM STOC*, 166–174, 2004.
- [3] R. Bar-Yehuda, G. Even and S. Shahar. On Approximating a Geometric Prize-Collecting Traveling Salesman Problem with Time Windows. *J. of Algorithms*, 55(1):76–92, 2005. Preliminary version in *Proc. of ESA*, 55–66, 2003.
- [4] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation Algorithms for Orienteering and Discounted-Reward TSP. *SIAM J. on Computing*, 37(2):653–670, 2007. Preliminary version in *Proc. of IEEE FOCS*, 46–55, 2003.
- [5] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. *Proc. of IEEE FOCS*, 36–45, 2003.
- [6] C. Chekuri, N. Korula, and M. Pál. Improved Algorithms for Orienteering and Related Problems. *Proc. of ACM-SIAM SODA*, 661–670, 2008.
- [7] C. Chekuri and A. Kumar. Maximum Coverage Problem with Group Budget Constraints and Applications. *Proc. of APPROX-RANDOM*, LNCS, 72–83, 2004.
- [8] C. Chekuri and M. Pal. A Recursive Greedy Algorithm for Walks in Directed Graphs. *Proc. of IEEE FOCS*, 245–253, 2005.
- [9] K. Chen and S. Har-Peled. The Orienteering Problem in the Plane Revisited. *Proc. of ACM SoCG*, 247–254, 2006.
- [10] G. Frederickson and B. Wittman. Approximation algorithms for the traveling repairman and speeding deliveryman problems. *Proc. of APPROX*, 119-133, 2007.
- [11] B. Golden, L. Levy and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.
- [12] G. Gutin and A. P. Punnen (Eds.). *Traveling Salesman Problem and Its Variations*. Springer, Berlin, 2002.
- [13] E. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. Shmoys (Eds.). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons Ltd., 1985.
- [14] V. Nagarajan and R. Ravi. Poly-logarithmic approximation algorithms for Directed Vehicle Routing Problems. *Proc. of APPROX*, 257–270, 2007.
- [15] *The Vehicle Routing Problem*. P. Toth, D. Vigo eds, SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.
- [16] J. Tsitsiklis. Special Cases of Traveling Salesman and Repairman Problems with Time Windows. *Networks*, vol 22, 263–282, 1992.