Approximation Algorithms for Orienteering and Discounted-Reward TSP

Avrim Blum* Shuchi Chawla* David R. Karger† Terran Lane‡ Adam Meyerson§
Maria Minkoff¶

Abstract

In this paper, we give the first constant-factor approximation algorithm for the rooted Orienteering problem, as well as a new problem that we call the Discounted-Reward TSP, motivated by robot navigation. In both problems, we are given a graph with lengths on edges and prizes (rewards) on nodes, and a start node s. In the *Orienteering Problem*, the goal is to find a path that maximizes the reward collected, subject to a hard limit on the total length of the path. In the *Discounted-Reward TSP*, instead of a length limit we are given a discount factor γ, and the goal is to maximize total discounted reward collected, where reward for a node reached at time t is discounted by γ^t. This problem is motivated by an approximation to a planning problem in the Markov decision process (MDP) framework under the commonly employed infinite horizon discounted reward optimality criterion. The approximation arises from a need to deal with exponentially large state spaces that emerge when trying to model one-time events and non-repeatable rewards (such as for package deliveries). We also consider tree and multiple-path variants of these problems and provide approximations for those as well. Although the unrooted orienteering problem, where there is no fixed start node s, has been known to be approximable using algorithms for related problems such as k-TSP (in which the amount of reward to be collected is fixed and the total length is approximately minimized), ours is the first to approximate the rooted question, solving an open problem [3, 1]. We complement our approximation result for Orienteering by showing that the problem is APX-hard.

*Computer Science Department, Carnegie Mellon University. Research supported by NSF Grants CCR-0105488, IIS-0121678, and CCR-0122581. email: {avrim,shuchi}@cs.cmu.edu

†MIT Computer Science and Artifi cial Intelligence Laboratory (CSAIL). email: karger@theory.csail.mit.edu

‡Department of Computer Science, University of New Mexico. email: terran@cs.unm.edu

§Computer Science Department, University of California, Los Angeles. This work was done while the author was visiting Carnegie Mellon University. email: awm@cs.ucla.edu

¶Max-Planck-Institut fr Informatik. This work was done while the author was a graduate student at MIT. email: mariam@theory.csail.mit.edu

1 Introduction

Consider a robot with a map of its environment, that needs to visit a number of sites in order to drop off packages, collect samples, search for a lost item, etc. One classic model of such a scenario is the *Traveling Salesman Problem*, in which we ask for the tour that visits all the sites and whose length is as short as possible. However, what if this robot cannot visit everything? For example, it might have a limited supply of battery power. In that case, a natural question to ask is for the tour that visits the maximum total reward of sites (where reward might correspond to the value of a package being delivered or the probability that some lost item we are searching for is located there), subject to a constraint that the total length is at most some given bound B. This is called the (rooted) *Orienteering Problem* ("rooted", because we are fixing the starting location of the robot). Interestingly, while there have been a number of algorithms that given a desired reward can approximately minimize the distance traveled (which yield approximations to the unrooted orienteering problem), approximating the reward for the case of a ***fixed*** starting location and ***fixed*** hard length limit has been an open problem.

Alternatively, suppose that battery power is not the limiting consideration, but we simply want to give the robot a penalty for taking too long to visit high-value sites. For example, if we are searching for a lost item, and at each time step there is some possibility the item will be taken (or, if we are searching for a trapped individual in a dangerous environment, and at each time step there is some probability the individual might die), then we would want to discount the reward for a site reached at time t by γ^t, where γ is a known discount factor. We call this the *Discounted-Reward TSP*. This problem is motivated by an approximation to a planning problem in the *Markov decision process* (MDP) framework [21, 20] under the commonly employed *infinite horizon discounted reward* optimality criterion. The approximation arises from a need to deal with exponentially large state spaces that emerge when trying to model one-time events and non-repeatable rewards (such as for package deliveries).

In this paper, we provide the first constant-factor approximations to both the (rooted) Orienteering and the Discounted-Reward TSP problems, and well as a number of variants that we discuss below. We also prove that Orienteering is APX-hard, or NP-hard to approximate within an arbitrarily small constant factor.

1.1 Motivation and Background

Robot navigation and path planning problems can be modeled in many ways. In the Theoretical Computer Science and Optimization communities, these are typically modeled as kinds of Prize-Collecting Traveling Salesman Problems [14, 4, 12, 3]. In the Artificial Intelligence community, problems of this sort are often modeled as Markov decision processes [6, 7, 16, 20, 21]. Below we give some background and motivation for our work from each perspective.

1.1.1 Markov decision process motivation

A Markov decision process (MDP) consists of a state space S, a set of actions A, a probabilistic transition function T, and a reward function R. For this work, it is sufficient to consider discrete, finite S and A. At any given time step, an agent (such as a robot) acting in an MDP will be located at some state $s \in S$, where it can choose an action $a \in A$. The agent is subsequently relocated to a new state s' drawn from the transition probability distribution $T(s'|s,a) \equiv \Pr[q_{t+1} = s'|q_t = s, a]$, where q_t is a random variable indicating the agent's state at time step t. The transition function captures both the agent's stochastic dynamics (e.g., unreliable actuators) and structure and characteristics of the environment such as walls, pits, friction of the surface, etc. Associated with each state is a real-valued reward, given by the function $R(s)$, which the agent

receives upon entering state s.[1] For example, a package-delivery robot might get a reward every time it correctly delivers a package.

The goal of planning in an MDP framework is to formulate a policy, $\psi : S \rightarrow A$, that guides the agent to optimal long-term aggregate reward. In order to encourage the agent to perform the tasks that we want, and to do so in a timely manner, a commonly employed aggregate reward objective function is the *infinite horizon discounted reward* [16, 20, 21]. Specifically, for a given *discount factor* $\gamma \in (0, 1)$, the value of reward collected at time t is discounted by a factor γ^t. Thus the total discounted reward, which we aim to maximize, is $R_{tot} = \sum_{t=0}^{\infty} R(s_t)\gamma^t$. Because the agent's actions are stochastic, in practice we must settle for optimizing the expected value of this quantity, $V^{\psi}(s) = \mathbf{E}_{\psi}[R_{tot}|q_0 = s]$, where the expectation is taken with respect to all possible trajectories through the state space rooted at state s, weighted by their probability of occurring under policy ψ. Note that, because a fixed (s, a) pair yields a fixed probability distribution over next states, the combination of an MDP with a fixed policy produces a Markov chain over S. The expectation, therefore, is simply the expected discounted reward accumulated by a random walk on the corresponding Markov chain. This optimality criterion guides the agent to accumulate as much reward as possible as early as possible, and produces what in practice turns out to be good behavior.

One can also motivate exponential discounting by imagining that, at each time step, there is some fixed probability the game will end (the robot loses power, a catastrophic failure occurs, the objectives change, etc.) Exponential discounting also has the nice mathematical property that it is *time-independent*, meaning that an optimal strategy is *stationary* and can be completely described by the mapping from states to actions given by ψ.[2] The overall goal of planning, then, is to locate ψ^*, the policy that maximizes $V^{\psi}(s)$, the expected infinte horizon discounted reward. A fundamental theorem of MDP planning states that for this optimality criterion, there is guaranteed to be a stationary $\psi*$ that dominates all other policies at all states: $V^{\psi^*}(s) \geq V^{\psi}(s)$ for all $s \in S$ and all ψ [20].

There are well-known algorithms for solving MDPs in time polynomial in the cardinality of the state space [6, 20, 21]. However, one drawback of the MDP model is that the agent receives $R(s)$ every time that state s is visited. Thus, in order to model a package-delivery or search-and-rescue robot, one would need a state representing not only the current location of the robot, but also a record of all packages (victims) it has already delivered (rescued). For example, one could write $S = L \times 2^d$, where L is a set of discrete locations that the robot could occupy, and the list of d bits tracks whether the agent has achieved each of d sub-goals (packages or rescues). Then the reward function can be $R(\langle l, b_1, \ldots, b_d \rangle) = 1$ iff l is a location containing sub-goal i and $b_i = 0$, or $R(s) = 0$ otherwise. When the robot reaches the location containing sub-goal i, b_i is set to 1 and remains so thereafter. This formulation yields an exponential increase in the size of the state space over the raw cardinality of L and prevents direct, exact solution of the MDP. Thus, it would be preferable to directly model the case of rewards that are given only the *first* time a state is visited [17, 18].

As a first step towards tackling this general problem, we abandon the stochastic element and restrict the model to deterministic, reversible actions. This model is a reasonable approximation to many robot-navigation style MDP domains, in which we can formulate sub-policies for navigating between pairs of locations in the environment. Often, such sub-policies, or macros, can be "nearly deterministic" (failing with probability $\leq \epsilon$) because they average out the stochasticity of atomic actions over many steps [18]. We can to a good approximation, therefore, treat such a domain as a deterministic planning problem over the set of sub-goal locations (nodes) and location-to-location macros (arcs). This leads us to study the *Discounted-Reward Traveling Salesman Problem*, in which we assume we have an undirected weighted graph (edge

[1] It is also possible to model rewards associated with actions or transitions by writing more general reward functions such as $R(s, a)$ or $R(s, a, s')$, but such extensions do not fundamentally change the nature of the MDP. Any such functions can be rewritten into a model of the form we give here with an appropriate modification to the state and action sets.

[2] Under other objective functions, an optimal policy could require dependence on the number of steps remaining in the game or other functions of the history of states encountered to date.

weights represent the time to traverse a given edge), with a prize (reward) value π_v on each vertex v, and our goal is to find a path visiting each vertex v at time t_v so as to maximize $\sum \pi_v \gamma^{t_v}$.

1.1.2 PC-TSP and Orienteering problems

A different way to model the goal of collecting as much reward as possible as early as possible is to impose a hard deadline on the time the robot may spend delivering its packages. The robot gets a reward equal to the value (prize) of the package on a delivery, but only if the delivery is made before a deadline D. If the deadline is exceeded, he gets no reward. This problem has been studied previously as the Orienteering [14] or Bank Robber's [3] Problem.

Orienteering belongs to the family of the Prize-Collecting Traveling Salesman problems (PC-TSP). Given a set of cities with non-negative prize values associated with them and a table of pairwise distances, a salesman needs to pick a subset of the cities to visit so as to minimize the total distance traveled while maximizing the total amount of prize collected. Note that there is a tradeoff between the cost of a tour and how much prize it spans. The original version of the PC-TSP introduced by Balas [4] deals with these two conflicting objectives by combining them: one seeks a tour that minimizes the *sum* of the total distance traveled and the penalties (prizes) on cities skipped, while collecting at least a given quota amount of prize. Goemans and Williamson subsequently focused on a special case of this problem in which the quota requirement is dropped, and provided a primal-dual 2-approximation algorithm for it [13].

An alternative approach to the bicriterion optimization is to optimize just one of the objectives while enforcing a fixed bound on the other. For example, in a quota-driven version of the PC-TSP, called k-TSP, every node has a prize of one unit and the goal is to minimize the total length of the tour, while visiting at least k nodes. Similarly, Orienteering can be viewed as a budget-driven version of the PC-TSP, since we are maximizing total amount of prize collected, while keeping the distance traveled below a certain threshold.[3]

There are several constant-factor approximations known for the k-TSP problem [2, 11, 8, 3], the best being a $(2+\epsilon)$-approximation due to Arora and Karakostas [2]. Most of these results are based on a classic primal-dual algorithm for PC-TSP due to Goemans and Williamson [13] (mentioned above).

The algorithms for k-TSP extend easily to the *unrooted* version of the Orienteering problem in which we do not fix the starting location [3]. In particular, given a tour (cycle) of value Π whose length is cD for some $c > 1$, we can just break the cycle into c pieces of length at most D, and then take the best one, whose total value will be at least Π/c. Noting that an optimal cycle of length $2D$ must span at least as much prize as an optimal path of length D (since one could just traverse the path forward and then back), we get a $2c$-approximation guarantee on the amount of prize contained in a segment we pick. However, this does not work for the rooted problem because the "best piece" in the above reduction might be far from the start. In contrast, there is no previously known $O(1)$ approximation algorithm for the rooted Orienteering Problem in general graphs. Arkin et al. [1] give a constant-factor approximation to the rooted Orienteering problem for the special case of points in the plane.

1.2 Summary of Results

In this paper, we give constant factor approximation algorithms for both the above problems. To do this, we devise a *min-excess* approximation algorithm that, given two endpoints s and t, approximates to within a constant factor the optimum *difference* between the length of a prize-collecting s-t path and the length of the shortest path between the two endpoints. Note that this is a strictly better guarantee than what can

[3] Strictly speaking, a budget-driven version of the PC-TSP would require a tour, e.g. a path that ends at the start node, whereas the Orienteering problem is content with a path that ends at an arbitrary node. We consider both versions of the problem.

be obtained by using an algorithm for k-TSP, which would return a path that has length at most a constant multiple times the *total* optimal length from s to t.

Using an approximation of α_{CP} for a variant of k-TSP, the min-cost s-t path problem (k-path problem in [9]), as a subroutine, we get an $\alpha_{EP} = \frac{3}{2}\alpha_{CP} - \frac{1}{2}$ approximation for the min-excess (s,t)-path problem. This further implies a $1 + \lceil \alpha_{EP} \rceil$ approximation for Orienteering, and a roughly $e(\alpha_{EP} + 1)$ approximation for Discounted-Reward TSP. Our final approximation factors for these problems are $2 + \epsilon$, 4, and $6.75 + \epsilon$ respectively, based on an improved analysis of a $(2 + \epsilon)$-approximation for min-cost s-t path due to Chaudhuri et al. [9].

Finally, using the APX-hardness of TSP on bounded metrics [10], we prove that the min-excess path problem and Orienteering are APX-hard.

1.3 Subsequent Work

Following the initial publication of our work, Bansal et al. [5] obtained a 3-approximation for a stronger version of Orienteering called "point-to-point Orienteering", in which the starting location s as well as the terminal location t are fixed. They also consider the Vehicle Routing Problem with Time-Windows, a generalization of Orienteering in which each reward has a time-window ("release time" and "deadline") associated with it, and reward is earned only if the location is visited within the corresponding time-window. Bansal et al. obtain an $O(\log^2 n)$ approximation for this problem, as well as an $O(\log n)$-approximation when all the release times are zero.

Organization. The rest of this paper is organized as follows. We begin with some definitions in Section 2. Then we give an algorithm for min-excess path in Section 3, followed by algorithms for Discounted PC-TSP and Orienteering in sections 4 and 5 respectively. In Section 6 we extend some of the algorithms to tree and multiple-path versions of the problems. We conclude in Section 8.

2 Notation and Definitions

Our work encompasses a variety of problems. In this section we introduce the notation to be used throughout the paper, provide formal problem statements and describe a uniform naming scheme for them.

Let $G = (V, E)$ be a weighted undirected graph, with a distance function on edges, $d : E \to \Re^+$, and a *prize* or *reward* function on nodes, $\pi : V \to \Re^+$. Let $\pi_v = \pi(v)$ be the reward on node v. Let $s \in V$ denote a special node called the *start* or *root*.

For a path P visiting u before v, let $d^P(u, v)$ denote the length along P from u to v. Let $d(u, v)$ denote the length of the *shortest* path from node u to node v. For ease of notation, let $d_v = d(s, v)$ and $d^P(v) = d^P(s, v)$. For a set of nodes $V' \subseteq V$, let $\Pi(V') = \sum_{v \in V'} \pi_v$. For a set of edges $E' \subseteq E$, let $d(E') = \sum_{e \in E'} d(e)$.

Our problems aim to construct a certain subgraph—a path, tree, or cycle, possibly with additional constraints. Most of the problems attempt a trade-off between two objective functions: the *cost* (distance) of the path (or tree, or cycle), and the *total prize* spanned by it. From the point of view of exact algorithms, we need simply to specify the cost we are willing to tolerate and the prize we wish to span. Most variants of this problem, however, are $\mathcal{NP}$-hard, so we focus on approximation algorithms. We must then specify our willingness to approximate the two distinct objectives. We refer to a *min-cost* problem when our goal is to *approximately* minimize the cost of our objective subject to a fixed lower bound on prize (thus, prize is a feasibility constraint while our approximated objective is cost). Conversely, we refer to a *max-prize* problem when our goal is to *approximately* maximize the prize collected subject to a fixed upper bound on cost (thus, cost is a feasibility constraint while our approximated objective is prize). For example, the min-cost tree

problem is the traditional k-MST: it requires spanning k prize and aims to minimize the cost of doing so. Both the rooted and unrooted min-cost tree problems have constant-factor approximations[15, 2, 11, 8, 3]. The max-prize path problem, which aims to find a path of length at most D from the start node s that visits a maximum amount of prize, is the orienteering problem.

The main subroutine in our algorithms requires also introducing a variation on approximate cost. Define the *excess* of a path P from s to t to be $d^P(s,t) - d(s,t)$, that is, the difference between that path's length and the distance between s and t in the graph. Obviously, the minimum-excess path of total prize Π is also the minimum-*cost* path of total prize Π; however, a path of a constant factor times minimum cost may have more than a constant-factor times the minimum excess. We therefore consider separately the *minimum excess path* problem. Note that an (s,t) path approximating the optimum excess ϵ by a factor α will have length $d(s,t) + \alpha\epsilon \leq \alpha(d(s,t) + \epsilon)$ and therefore approximates the minimum cost path by a factor α as well. Achieving a good approximation to this min-excess path problem will turn out to be a key ingredient in our approximation algorithms.

Finally, as discussed earlier, we consider a different means of combining length and cost motivated by applications of Markov decision processes. We introduce a *discount factor* $\gamma < 1$. Given a path P rooted at s, let the *discounted reward* collected at node v by path P be defined as $\rho_v^P = \pi_v \gamma^{d^P(s,v)}$. That is, the prize gets discounted exponentially by the amount of time it takes for the path to reach node v. The *max-discounted-reward* problem is to find a path P rooted at s, that maximizes $\rho^P = \sum_{v \in P} \rho_v^P$. We call this the *discounted-reward TSP*. Note that the length of the path is not specifically bounded in this problem, though of course shorter paths produce less discounting.

Problem	Best approx.	Source/Reduction	Hardness of approx.
min-cost s-t path (α_{CP})	$2+\epsilon$	[9]	$220/219$
min-excess path (α_{EP})	$2.5+\epsilon$	$\frac{3}{2}(\alpha_{CP}) - \frac{1}{2}$	$220/219$
	$2+\epsilon$	algorithm based on [9]	
max discounted-prize path (α_{DP})	$6.75+\epsilon$	$(1+\alpha_{EP})(1+1/\alpha_{EP})^{\alpha_{EP}}$	
max-prize path (α_{PP})	4	$1 + \lceil \alpha_{EP} \rceil$	$1481/1480$
max-prize tree (α_{PT})	8	$2\alpha_{PP}$	
max-prize cycle (α_{PC})	8	$2\alpha_{PP}$	$1481/1480$
max-prize multiple-path (α_{kPP})	5	$\alpha_{PP} + 1$	$1481/1480$

Figure 1: Approximation factors and reductions for our problems.

2.1 Results

We present a constant-factor approximation algorithm for the max-prize path (rooted Orienteering) problem, solving an open problem of [3, 1], as well as the discounted-reward TSP. Central to our results is a constant-factor approximation for the *min-excess path* problem defined above, which uses an algorithm for the min-cost s-t path problem as a subroutine. We also give constant-factor approximations to several related problems, including the max-prize tree problem—the "dual" to the k-MST (min-cost tree) problem—and max-prize cycle. Specific constants are given in Figure 1. For the Min-Excess problem, we derive an improved approximation of $2+\epsilon$ in section 3.2, based on a tighter analysis of the min-cost s-t path algorithm of [9]. This improvement gives a better approximation factor of $6.75+\epsilon$ for the Max Discounted-Prize Path problem.

Our approximation algorithms reflect a series of reductions from one approximation problem to another. Improvements in the approximations for various problems will propagate through. We state approximation

factors in the form α_{XY} where XY denotes the problem being approximated; the first letter denotes the objective (cost, prize, excess, or discounted prize denoted by C, P, E, and D respectively), and the second the structure (path, cycle, or tree denoted by P, C, or T respectively).

2.2 Preliminaries

To support dynamic programming in the max-prize variants, we begin by scaling all prizes to polynomially bounded integers (in the number of vertices n). We can do this by guessing the value Π of the optimum solution via binary search[4] and multiplying all prizes by n^2/Π, yielding a graph with optimal prize value n^2. If we now round every prize down to the nearest integer, we lose at most n units of prize, which is a negligible multiplicative factor. This negligible factor does mean that an approximation algorithm with guarantee c on polynomially bounded inputs has (weaker) guarantee "arbitrarily close to c" on arbitrary inputs. Likewise, for the min-cost or min-excess variants, we can assume that the given prize value Π is polynomially bounded.

3 Min-Excess Path

Let P^* be the shortest path from s to t with $\Pi(P^*) \geq k$. Let $\epsilon(P^*) = d(P^*) - d(s,t)$. Our algorithm returns a path P with $\Pi(P) \geq k$ and length $d(P) = d(s,t) + \alpha_{EP}\epsilon(P^*)$, where $\alpha_{EP} = \frac{3}{2}\alpha_{CP} - \frac{1}{2}$. Thus we obtain a $(2.5+\delta)$-approximation to min-excess path using an algorithm of Chaudhuri et al. [9] for min-cost s-t path (MCP) with $\alpha_{CP} = 2 + \delta$.

We begin with a brief description of the min-cost path algorithm and approximation. In their paper Chaudhuri et al. provide a subroutine for constructing a tree containing nodes s and t that spans at least k vertices[5] and has cost at most $(1+\delta)$ times the cost of the shortest s-t path with k vertices, for any fixed constant δ. To construct an s-t path from the tree obtained by the algorithm of Chaudhuri et al., we can double all the edges, except those along the tree path from s to t. This gives us a partial "Euler tour" of the tree that starts at s and ends at t. Clearly, the cost of such a path is at most $(2+\delta)$ times the cost of the shortest s-t path spanning prize k, for any fixed constant δ.

Now we return to the harder Min-Excess Path (MEP) problem. The idea for our algorithm for Min-Excess Path (MEP) is as follows. Suppose that the optimum solution path encounters all its vertices in increasing order of distance from s. We call such a path *monotonic*. We can find this optimum monotonic path via a simple dynamic program: for each possible prize value p and for each vertex i in increasing order of distance from s, we compute the minimum excess path that starts at vertex s, ends at i, and collects prize at least p.

We solve the general case by breaking the optimum path into *segments* that are either monotonic (so can be found optimally as just described) or "wiggly" (generating a large amount of excess). We show that the total length of the wiggly portions is comparable to the excess of the optimum path; our solution uses the optimum monotonic paths and approximates the length of the wiggly portions by a constant factor, yielding an overall increase proportional to the excess.

Consider the optimal path P^* from s to t. We divide it into segments in the following manner. For any real d, define $f(d)$ as the number of edges on P^* with one endpoint at distance less than d from s and the other endpoint at distance at least d from s. Note that $f(d) \geq 1$ for all $0 \leq t \leq d_t$ (it may also be nonzero for some $d \geq d_t$). Note also that f is piecewise constant, changing only at distances equal to vertex distances. We break the real line into intervals according to f: the *type one intervals* are the maximal intervals on

[4] Technically we will be finding the highest value Π such that our algorithm comes within its claimed approximation ratio.

[5] The algorithm can be transformed easily to obtain a tree spanning a given target *prize value*—to each node v with a prize π_v, we attach $\pi_v - 1$ leaves, and run the algorithm on this new graph.

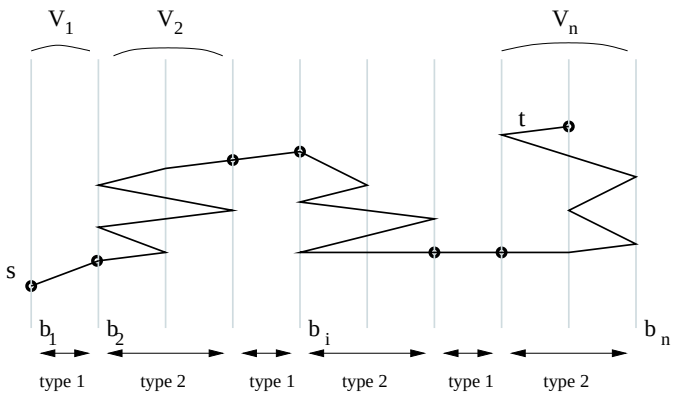

Figure 2: Segment partition of a path in graph G

which $f(d) = 1$; the type 2 intervals are the maximal intervals on which $f(d) \geq 2$. These intervals partition the real line (out to the maximum distance reached by the optimum solution) and alternate between types 1 and 2. Let the interval boundaries be labeled $0 = b_1 < b_2 \cdots b_m$, where b_m is the maximum distance of any vertex on the path, so that the i^{th} interval is (b_i, b_{i+1}). Note that each b_i is the distance label for some vertex. Let V_i be the set of vertices whose distance from s falls in the i^{th} interval. Note that the optimum path traverses each set V_i exactly once—once it leaves some V_i it does not return. One of any two adjacent intervals is of type 1; if the path left this interval and returned to it then $f(d)$ would exceed 1 within the interval. Thus, the vertices of P^* in set V_i form a contiguous *segment* of the optimum path that we label as $S_i = P^* \cap V_i$.

A segment partition is shown in Figure 2.

Note that for each i, there may be (at most) 1 edge crossing from V_i to V_{i+1}. To simplify the next two lemmas, let us split that edge into two with a vertex at distance b_i from s, so that every edge is completely contained in one of the segments (this can be done since one endpoint of the edge has distance exceeding b_i and the other endpoint has distance less than b_i).

Lemma 3.1. *A segment S_i of type 1 has length at least $b_{i+1} - b_i$. A segment S_i of type 2 has length at least $3(b_{i+1} - b_i)$, unless it is the segment containing t in which case it has length at least $3(d_t - b_i)$.*

Proof. The length of segment S_i is lower bounded by the integral of $f(d)$ over the i^{th} interval. In a type 1 interval the result is immediate. For a type 2 interval, note that $f(d) \geq 1$ actually implies that $f(d) \geq 3$ by a parity argument—if the path crosses distance d twice only, it must end up at distance less than d. □

Corollary 3.2. *The total length of type-2 segments is at most $3\epsilon(P^*)/2$.*

Proof. Let ℓ_i denote the length of segment i. We know that the length of P^* is $d_t + \epsilon(P^*) = \sum \ell_i$. At the same time, we can write

$$d_t \leq b_m = \sum_{i=1}^{m-1} (b_{i+1} - b_i) \leq \sum_{i \text{ type } 1} \ell_i + \sum_{i \text{ type } 2} \ell_i/3$$

It follows that

$$\epsilon(P^*) = \sum \ell_i - d_t \geq \sum_{i \text{ type } 2} 2\ell_i/3$$

Multiplying both sides by $3/2$ completes the proof. □

Having completed this analysis, we note that the corollary remains true even if we do not introduce extra vertices on edges crossing interval boundaries. The crossing edges are no longer counted as parts of segments, but this only decreases the total length of type 2 segments.

3.1 A Dynamic Program

Our algorithm computes, for each interval that might be an interval of the optimum solution, a segment corresponding to the optimum solution in that interval. It then uses a dynamic program to paste these fragments together using (and paying for) edges that cross between segments. The segments we compute are defined by 4 vertices: the closest-to-s and farthest-from-s vertices, c and f, in the interval (which define the start- and end-points of the interval: our computation is limited to vertices within that interval), and the first and last vertices, x and y, on the segment within that interval. They are also defined by the amount p of prize we are required to collect within the segment. There are therefore $O(\Pi n^4)$ distinct segments to compute, where Π is the total prize in the graph. For each segment we find an optimum solution for a type 1 and a type 2 interval. For a type-1 interval the optimum path is monotonic; we can therefore compute (in linear time) an optimum (shortest) monotonic path from x to y that collects prize p. If the interval is of type 2, the optimum path need not be monotonic. Instead, we use the MCP routine to approximate to within a constant factor the minimum length of a path that starts at x, finishes at y, stays within the boundaries of the interval defined by c and f, and collects prize at least p.

Given the optimum type 1 and near-optimum type-2 segment determined for each set of 4 vertices and prize value, we can find the optimal way to paste some subset of them together monotonically using a dynamic program. Note that the segments corresponding to the optimum path are considered in this dynamic program, so our solution will be at least as good as the one we get by using the segments corresponding to the ones on the optimum path (i.e., using the optimum type-1 segments and using the approximately optimum type-2 segments). We need only show that this solution is good.

We focus on the segments corresponding to the optimum path P^*. Consider the segments S_i of length ℓ_i on the optimum path. If S_i is of type 1, our algorithm will find a (monotonic) segment with the same endpoints collecting the same amount of prize of no greater length. If S_i is of type 2, our algorithm (through its use of subroutine MCP) will find a path with the same endpoints collecting the same prize over length at most $\alpha_{CP}\ell_i$. Let L_1 denote the total length of the optimum type 1 segments, together with the lengths of the edges used to connect between segments. Let L_2 denote the total length of the optimum type 2 segments. Recall that $L_1 + L_2 = d_t + \epsilon(P^*)$ and that (by Corollary 3.2) $L_2 \leq \frac{3}{2}\epsilon(P^*)$. By concatenating the optimum type-1 segments and the approximately optimum type-2 segments, the dynamic program can (and therefore will) find a path collecting the same total prize as P^* of total length at most

$$\begin{aligned} L_1 + \alpha_{CP} L_2 &= L_1 + L_2 + (\alpha_{CP} - 1)L_2 \\ &\leq d_t + \epsilon(P^*) + (\alpha_{CP} - 1)\left(\frac{3}{2}\epsilon(P^*)\right) \\ &= d_t + \left(\frac{3}{2}\alpha_{CP} - \frac{1}{2}\right)\epsilon(P^*). \end{aligned}$$

In other words, we approximate the minimum excess to within a factor of $\frac{3}{2}\alpha_{CP} - \frac{1}{2}$.

3.2 An Improved Approximation for Min-Excess

Our approximation guarantee for min-excess path α_{EP} is based on the approximation guarantee α_{CP} for the min-cost path problem. However, we use the algorithm for the latter problem as a "black-box" subroutine.

In this section we show how to slightly improve our approximation guarantee for min-excess path problem by exploiting the details of the min-cost path algorithm derived from the work of Chaudhuri et al. [9].

Let us consider a segment of type-2 of an optimum min-excess path with endpoints u and v and length $\ell = d(u,v) + \epsilon$. If we apply the algorithm of Chaudhuri et al. with roots u and v, we get a tree containing u and v of total edge cost arbitrarily close to ℓ. Our min-cost path algorithm then converts this tree into a path from u to v by doubling all edges, except for the ones on the tree path from u to v. Noting that the total cost of "non-doubled" edges is $d(u,v) = \ell - \epsilon$, we get a path from u to v of length at most $(1+\delta)\ell + \epsilon$, for any fixed small constant δ.

Now we can apply Corollary 3.2 to obtain an approximation in terms of the excess. However, note that the ϵ in the above expression is the excess of the path between the nodes u and v, and is not the same as the difference of the excess of the path P^* at v and its excess at u. In order to obtain a bound in terms of the latter, let ϵ_u denote the excess of P^* from s to u, and ϵ_v the excess of P^* from s to v. Then, $\ell = (d_v + \epsilon_v) - (d_u + \epsilon_u) \leq d(u,v) + \epsilon_v - \epsilon_u$. Therefore, $\epsilon \leq \epsilon_v - \epsilon_u$, and the Chaudhuri et al. algorithm returns a path of length at most $(1+\delta)\ell + \epsilon_v - \epsilon_u$.

The dynamic program finds a path collecting the same total prize as P^* and of total length at most

$$\begin{aligned} L_1 + (1+\delta)L_2 + \sum_{\text{type 2 segments}} (\epsilon_v - \epsilon_u) &\leq L_1 + (1+\delta)L_2 + \epsilon(P^*) \\ &= d_t + 2\epsilon(P^*) + \delta L_2 \\ &\leq d_t + 2\epsilon(P^*) + \frac{3}{2}\delta\,\epsilon(P^*) \end{aligned}$$

where the last statement follows from Corollary 3.2. Therefore, we get an approximation ratio of $2 + \delta'$ for the min-excess path problem, for any small constant δ'.

4 Maximum Discounted-Prize Path

In this section we present an approximation algorithm for the Discounted-Reward TSP problem which builds upon our min-excess path algorithm. Recall that we aim to optimize $\rho(P) = \sum \gamma^{d_v^P} \pi_v$. Assume without loss of generality that the discount factor is $\gamma = 1/2$—we simply rescale each length ℓ to ℓ' such that $\gamma^\ell = (\frac{1}{2})^{\ell'}$, i.e., $\ell' = \ell \log_2(1/\gamma)$.

We first establish a property of an optimal solution that we make use of in our algorithm. Define the *scaled prize* π' of a node v to be the (discounted) reward that a path gets at node v if it follows a shortest path from the root to v. That is, $\pi'_v = \pi_v \gamma^{d_v}$. Let $\Pi'(P) = \sum_{v \in P} \pi'_v$. Note that for any path P, the discounted reward obtained by P is at most $\Pi'(P)$.

Now consider an optimal solution P^*. Fix a parameter ϵ that we will set later. Let t be the last node on the path P^* for which $d_t^{P^*} - d_t \leq \epsilon$, i.e., the excess of path P^* at t is at most ϵ. Consider the portion of P^* from root s to t. Call this path P_t^*.

Lemma 4.1. *Let P_t^* be the part of P^* from s to t. Then, $\rho(P_t^*) \geq \rho(P^*)(1 - \frac{1}{2^\epsilon})$.*

Proof. Assume otherwise. Suppose we shortcut P^* by taking a shortest path from s to the next node visited by P^* after t. This new path collects (discounted) rewards from the vertices of $P^* - P_t^*$, which form more than a $\frac{1}{2^\epsilon}$ fraction of the total discounted reward by assumption. The shortcutting procedure decreases the distance on each of these vertices by at least ϵ, meaning these rewards are "undiscounted" by a factor of at least 2^ϵ over what they would be in path P^*. Thus, the total reward on this path exceeds the optimum, a contradiction. □

It follows that we can approximate $\rho(P^*)$ by approximating $\rho(P_t^*)$. Based on the above observation, we give the algorithm of Figure 3 for finding an approximately optimal solution. Note that "guess t" and "guess k" are implemented by exhausting all polynomially many possibilities.

Algorithm for Discounted PC-TSP

1. Re-scale all edge lengths so that $\gamma = 1/2$.
2. Replace the prize value of each node with the prize discounted by the shortest path to that node: $\pi'_v = \gamma^{d_v}\pi_v$. Call this modified graph G.
3. Guess t—the last node on optimal path P^* with excess less than ϵ.
4. Guess k—the value of $\Pi'(P_t^*)$.
5. Apply our min-excess path approximation algorithm to find a path P collecting scaled prize k with small excess.
6. Return this path as the solution.

Figure 3: Approximation for Maximum Discounted-Prize Path

Our analysis below proceeds in terms of $\alpha = \alpha_{EP}$, the approximation factor for our min-excess path algorithm.

Lemma 4.2. *Our approximation algorithm finds a path P that collects discounted reward $\rho(P) \geq \Pi'(P_t^*)/2^{\alpha\epsilon}$.*

Proof. The prefix P_t^* of the optimum path shows that it is possible to collect scaled prize $k = \Pi'(P_t^*)$ on a path with excess ϵ. Thus, our approximation algorithm finds a path collecting the same scaled prize with excess at most $\alpha\epsilon$. In particular, the excess of any vertex v in P is at most $\alpha\epsilon$. Thus, the discounted reward collected at v is at least

$$\rho(v) \geq \pi_v \left(\frac{1}{2}\right)^{d_v+\alpha\epsilon} = \pi_v \left(\frac{1}{2}\right)^{d_v} \left(\frac{1}{2}\right)^{\alpha\epsilon} = \pi'_v \left(\frac{1}{2}\right)^{\alpha\epsilon}$$

Summing over all $v \in P$ and observing $\Pi'(P) \geq \Pi'(P^*)$ completes the proof. □

Combining Lemma 4.2 and Lemma 4.1, we get the following:

Theorem 4.3. *The solution returned by the above algorithm has $\rho(P) \geq (1 - \frac{1}{2^\epsilon})\rho(P^*)/2^{\alpha\epsilon}$.*

Proof.

$$\begin{aligned}
\rho(P) &\geq \Pi'(P^*)/2^{\alpha\epsilon} && \text{by Lemma 4.2} \\
&\geq \rho(P_t^*)/2^{\alpha\epsilon} && \text{by definition of } \pi' \\
&\geq \left(1 - \frac{1}{2^\epsilon}\right)\rho(P^*)/2^{\alpha\epsilon} && \text{by Lemma 4.1}
\end{aligned}$$

□

We can now set ϵ as we like. Writing $x = 2^{-\epsilon}$ we optimize our approximation factor by maximizing $(1-x)x^\alpha$ to deduce $x = \alpha/(\alpha+1)$. Plugging in this x yields an approximation ratio of $(1+\alpha_{EP})(1+1/\alpha_{EP})^{\alpha_{EP}}$.

5 Orienteering

In this section we present an algorithm for computing an approximately maximum-prize path of length at most D that starts at a specified vertex s. We will use the algorithm for min-excess path given in section 3 as a subroutine. Our algorithm for the Max-Prize problem is given in Figure 4. As before "guess k" is implemented by performing a binary search.

Algorithm for Max-Prize Path (Orienteering)

1. Guess k, the amount of prize collected by an optimum orienteering solution. Let $\alpha = \lceil \alpha_{EP} \rceil + 1$.
2. For each vertex v, compute min-excess path from s to v collecting prize k/α.
3. There exists a v such that the min-excess path returned has length at most D; return the corresponding path.

Figure 4: Algorithm for Max-Prize Path (Orienteering)

We analyze this algorithm by showing that any optimum orienteering solution contains a low-excess path which, in turn, is an approximately optimum orienteering solution. More precisely, we prove that for some vertex v, there exists a path from s to v with excess at most $\frac{D-d_v}{\alpha_{EP}}$ that collects prize at least $\frac{\Pi^*}{\alpha_{PP}}$ where α_{EP} is the approximation ratio for min-excess path, α_{PP} is the desired approximation ratio for Max-Prize Path, and Π^* is the prize of the optimum Max-Prize Path. Assuming this path exists, our min-excess path computation on this vertex v will find a path with total length at most $d_v + \alpha_{EP}\frac{D-d_v}{\alpha_{EP}} = D$ and prize at least $\frac{\Pi^*}{\alpha_{PP}}$, providing an α_{PP}-approximation for orienteering.

Let t be the last vertex on the optimum orienteering path. We first consider the case where t is the vertex at maximum distance from s on the optimum path.

Lemma 5.1. *If there is a path from s to t of length at most D that collects prize Π, such that t is the furthest point from s along this path, then there is a path from s to some node v with excess at most $\frac{D-d_v}{r}$ and prize at least $\frac{\Pi}{r}$ (for any integer $r \geq 1$).*

Proof. For each point a along the original path P, let $\epsilon(a) = d_a^P - d_a$; in other words, $\epsilon(a)$ is the excess in the length of the path to a over the shortest-path distance. We have $\epsilon(t) \leq D - d_t$. Consider mapping the points on the path to a line from 0 to $\epsilon(t)$ according to their excess (we observe that excess only increases as we traverse path P). Divide this line into r intervals with length $\frac{\epsilon(t)}{r}$. Some such interval must contain at least $\frac{\Pi}{r}$ prize, since otherwise the entire interval from 0 to $\epsilon(t)$ would not be able to collect prize Π. Suppose such an interval starts with node a and ends with node v. We consider a path from s to v that takes the shortest s-a path, then follows path P from a to v. This path collects the prize of the interval from a to v in the original path, which is a prize of at least $\frac{\Pi}{r}$ as desired. The total length of this path is $d_a + d^P(a,v) = d_a + d_v^P - d_a^P = d_v + \epsilon(v) - \epsilon(a) \leq d_v + \frac{\epsilon(t)}{r}$. The excess of this path is $\frac{\epsilon(t)}{r} \leq \frac{D-d_t}{r} \leq \frac{D-d_v}{r}$. □

Of course, in general the optimum orienteering path might have some intermediate node that is farther from s than the terminal node t. We will generalize the above lemma to account for this case.

Lemma 5.2. *If there is a path from s to t of length at most D that collects prize Π, then there is a path from s to some node v with excess at most $\frac{D-d_v}{r}$ and prize at least $\frac{\Pi}{r+1}$ (for any integer $r \geq 1$).*

Proof. Let f be the furthest point from s along the given path P. We are interested in the case where $f \neq t$. We can break path P into two pieces; first a path from s to f and then a path from f to t. Using the symmetry of our metric, we can produce a second path from s to f by using the shortest path from s to t and then following the portion of our original path from f to t in reverse. We now have two paths from s to f, each of which has length at most D. The total length of these paths is bounded by $D + d_t$. We will call our paths A and B, and let their lengths be $d_f + \delta_A$ and $d_f + \delta_B$ respectively. Note that $\delta_A + \delta_B \leq D + d_t - 2d_f < D - d_f$.

We now map path A to the interval from 0 to δ_A according to the excess at each point, much as in Lemma 5.1. We consider dividing this interval into pieces of length $\frac{\delta_A + \delta_B}{r}$ (the last sub-interval may have shorter length if δ_A does not divide evenly). We perform the same process on path B. We have created a total of $r + 1$ intervals (this relies on the assumption that r is integral, allowing us to bound the sum of the ceilings of the number of intervals for each path). We conclude that some such interval has prize at least $\frac{\Pi}{r+1}$. We suppose without loss of generality that this interval spans a portion of path A from a to v. We now consider a path that travels from s to a via the shortest path and then from a to v following path A. The length of this path is bounded by $d_v + \frac{\delta_A + \delta_B}{r}$ for an excess of at most $\frac{D - d_f}{r} \leq \frac{D - d_v}{r}$ as desired. □

Making use of Lemma 5.2, we can prove that our algorithm for orienteering obtains a constant approximation. Making use of Chaudhuri et al.'s approximation for min-cost s-t path [9] along with our result on min-excess path from section 3, we have a 4-approximation for Orienteering.

Theorem 5.3. *Our algorithm is an $(\lceil \alpha_{EP} \rceil + 1)$-approximation for the max-prize path (orienteering) problem, where α_{EP} is the approximation factor for min-excess path.*

Proof. Lemma 5.2 implies that there exists a path from s to some v with excess $\frac{D - d_v}{\lceil \alpha_{EP} \rceil}$ obtaining prize $\frac{\Pi^*}{\lceil \alpha_{EP} \rceil + 1}$. Such a path has length $d_v + \frac{D - d_v}{\lceil \alpha_{EP} \rceil}$, implying that the approximation algorithm for min-excess will find a path from s to v with length at most $d_v + (D - d_v) = D$ and at least the same prize. The algorithm described will eventually try the proper values of k and v and find such a path in polynomial time. □

6 Extensions

6.1 Max-Prize Tree and Max-Prize Cycle

In this section, we consider the tree and cycle variants of the Orienteering problem. In Max-Prize Tree, given a graph G with root r, prize function Π and lengths d, we are required to output a tree $\mathcal{T}$ rooted at r with $d(\mathcal{T}) \leq \mathcal{D}$ and maximum possible reward $\Pi(\mathcal{T})$. This problem is also called the Budget Prize-Collecting Steiner Tree problem [15]. Although the unrooted version of the problem can be approximated to within a factor of $5 + \epsilon$ via a 3-approximation for k-MST [15], the version of the problem in which a tree is required to contain a specified vertex has remained open until recently.

Let the optimal solution for this problem be a tree $\mathcal{T}^*$. Double the edges of this tree to obtain an Euler tour of length at most $2D$. Now, divide this tour into two paths, each starting from the root r and having length at most D. Among them, let P' be the path that has greater reward. Now consider the Max-Prize Path problem on the same graph with distance limit D. Clearly the optimal solution P^* to this problem has $\Pi(P^*) \geq \Pi(P') \geq \frac{\Pi(\mathcal{T}^*)}{2}$. Thus, we can use the α_{PP}-approximation for Orienteering to get a $2\alpha_{PP}$-approximation to T^*.

Finally we note that we can use our algorithm for the Orienteering problem to approximate Max-Prize Cycle. Namely, we can find an approximately maximum-prize cycle of length at most D that contains a specified vertex s. To this end we apply our algorithm to an instance of the Orienteering problem with the starting node s and the length constraint $D/2$. To obtain a cycle from the resulting path we connect its endpoints by a shortest path. Clearly, the length of the resulting cycle is at most D. Now, notice that an

optimal max-prize cycle of length D can span at most twice the amount of prize that an optimal max-prize path of length $D/2$. Thus, using α_{PP}-approximation to Orienteering we get $2\alpha_{PP}$-approximation to the Max-Prize Cycle problem.

6.2 Multiple-Path Orienteering

In this section we consider a variant of the Orienteering in which we are allowed to construct up to k paths, each having length at most D.

We approximate this problem by applying the algorithm in Section 4 successively k times, to construct the k paths. At the i-th step, we set the prizes of all points visited in the first $i-1$ paths to 0, and constructed the i-th path on the new graph, using the Orienteering algorithm in Section 5. Using a set-cover like argument, we get the following approximation guarantees for the cases when all paths have the same starting point and when different paths have different starts.

Theorem 6.1. *If all the paths have a common start node, the above algorithm gives a $1/(1-e^{-\alpha_{PP}})$ approximation to Multiple-Path Orienteering. If the paths have different start nodes, the above algorithm gives a $\alpha_{PP}+1$ approximation to Multiple-Path Orienteering.*

Proof. Consider first the case when all the paths have the same starting point. Let the difference in the reward collected by the optimal solutionand the reward collected by our solution upto stage i be Π_i. At the beginning, this is the total reward of the optimal solution. At step i, at least one of the paths in the optimal solution collects reward, not collected by the algorithm by stage i, of value at least $\frac{1}{k}\Pi_i$. Then, using the approximation guarantee of the algorithm for orienteering, our solution collects at least a $\frac{1}{k\alpha_{PP}}$ fraction of this reward. That is, $\Pi_{i+1} \leq (1-\frac{1}{k\alpha_{PP}})\Pi_i$. By the end of k rounds, the total reward collected by optimal solution, but not collected by us, is at most $(1-\frac{1}{k\alpha_{PP}})^k\Pi(P^*) \leq e^{-\alpha_{PP}}\Pi(P^*)$, and the result follows.

Next consider the case when different paths have different starting locations. Let O_i be the set of points visited by the i-th path in the optimal solution, and A_i be the corresponding set of points visited by our algorithm. Let Δ_i be the set of points that are visited by the i-th path in the optimal solution and some other path in our solution. Let $O = \cup_i O_i$, $A = \cup_i A_i$ and $\Delta = \cup_i \Delta_i$. Now, in the i-th stage, there is a valid path starting at the i-th source, that visits all points in $O_i \setminus \Delta_i$. Thus we have $\Pi(A_i) \geq \frac{1}{\alpha_{PP}}(\Pi(O_i) - \Pi(\Delta_i))$. Summing over i, we get $\alpha_{PP}\Pi(A) \geq (\Pi(O) - \Pi(\Delta))$. But $\Pi(\Delta) \leq \Pi(A)$. Thus $\Pi(A) \geq \frac{1}{\alpha_{PP}+1}\Pi(O)$. □

7 Hardness of Approximation

All the problems discussed in this paper are NP-hard, as they are generalizations of the Traveling Salesman Problem. In this section we show that the min-excess path problem and Orienteering are APX-hard, that is, it is NP-hard to approximate these problems to within an arbitrary constant factor.

The hardness of approximating the min-excess path problem follows from the APX-hardness of TSP [19]. In particular, we can approximate TSP to within an α factor by approximating the min-excess problem to within an α factor with a reward quota of n. We therefore get the following theorem:

Theorem 7.1. *The min-excess path problem is NP-hard to approximate to within a factor of $\frac{220}{219}$.*

Theorem 7.2. *Orienteering is NP-hard to approximate to within a factor of $\frac{1481}{1480}$.*

Proof. We reduce the TSP on $\{1,2\}$-metrics to Orienteering. In particular, let $G=(V,E)$ be a complete graph on n nodes, with edges lengths in the set $\{1,2\}$. Engebretsen and Karpinski [10] show that the TSP is NP-hard to approximate within a factor of $1+\alpha = \frac{741}{740}$ on such graphs.

Our reduction is as follows. Let the length of the optimal TSP solution be $L = n + \delta n$. (We simply try all values of L between n and $2n$.) Suppose that there is an algorithm that approximates Orienteering within a factor of $1 + \beta$, where $\beta \leq \frac{1}{1480}$. We apply this algorithm to the graph G with distance limit L. Note that the optimal solution (which is the optimal TSP path) collects $n - 1$ nodes within distance L (all nodes except the start, assuming a reward of 0 on the start node). Therefore, the solution returned by our algorithm collects $\frac{1}{1+\beta}(n-1)$ nodes. We augment this solution to a tour containing all the nodes, by using $(1 - \frac{1}{1+\beta})(n-1) + 1$ edges of length at most 2. Therefore, the length of our solution is at most

$$\begin{aligned} L + 2(1 - \tfrac{1}{1+\beta})(n-1) + 2 &= L + \tfrac{2\beta}{1+\beta}(n-1) + 2 \\ &< L + 2\beta n \\ &= L + \alpha n \leq (1+\alpha)L \end{aligned}$$

where the second inequality follows from assuming that $n > \frac{1}{\beta^2}$.

Therefore, we get a $(1+\alpha)$-approximation to TSP on G, contradicting the fact that TSP is NP-hard to approximate to within a $(1+\alpha)$ factor on $\{1,2\}$-metrics. □

Using a similar argument as for Orienteering, we get a $\frac{1481}{1480}$ hardness of approximation for the max-prize cycle problem as well.

8 Conclusions

In this paper we give constant factor algorithms for the Orienteering problem, Discounted-Reward TSP, and some of their variants. We also prove that it is NP-hard to obtain a PTAS for the Orienteering and min-excess problems. An interesting open problem is to obtain better approximations, or even a PTAS, for these problems when the underlying metric is planar. Another interesting open problem is to consider the directed versions of the problems, although we believe that it may be hard to approximate these to within constant or even logarithmic factors.

Even more ambitiously, returning to the MDP motivation for this work, one would like to generalize these results to probabilistic transition functions. However, this has the additional complication that the optimum solution may not even have a short description (it is no longer just a path). Still, perhaps some sort of non-trivial approximation bound, or a result holding in important special cases, can be found.

References

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

[2] S. Arora and G. Karakostas. A $2 + \epsilon$ approximation algorithm for the k -MST problem. In *Symposium on Discrete Algorithms*, pages 754–759, 2000.

[3] 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.

[4] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[5] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation Algorithms for Deadline-TSP and Vehicle Routing with Time-Windows. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, 2004.

[6] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[7] D. P. Bertsekas and J. N. Tsitsiklis. *Neural Dynamic Programming*. Athena Scientific, 1996.

[8] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. *JCSS*, 58:101–108, 1999.

[9] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, 2003.

[10] Lars Engebretsen and Marek Karpinski. Approximation hardness of tsp with bounded metrics. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, pages 201–212. Springer-Verlag, 2001.

[11] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 302–309, October 1996.

[12] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–315, 1992.

[13] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24:296–317, 1995.

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

[15] D. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: Theory and practice. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[17] T. Lane and L. P. Kaelbling. Approaches to macro decompositions of large markov decision process planning problems. In *Proceedings of the 2001 SPIE Conference on Mobile Robotics*, Newton, MA, 2001. SPIE.

[18] T. Lane and L. P. Kaelbling. Nearly deterministic abstractions of markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, Edmonton, 2002.

[19] Christos Papadimitriou and Santosh Vempala. On the approximability of the traveling salesman problem. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000.

[20] M. L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[21] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.