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 is similar to the objective considered in Markov Decision Processes (MDPs) except we only receive a reward the* first *time a node is visited. 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].*

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

† MIT Laboratory for Computer Science. email: {karger,mariam}@theory.lcs.mit.edu

‡ Department of Computer Science, University of New Mexico. email: terran@cs.unm.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 is similar to the classic setting of *Markov Decision Processes* [18, 17], except that we are giving the robot a reward only for the *first* time it visits a site (and, we are assuming a deterministic environment).

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.

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 [12, 4, 10, 3]. In the Artificial Intelligence community, problems of this sort are often modeled as Markov Decision Processes [5, 6, 14, 17, 18]. 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. At any given time step, an agent acting in an MDP will be located at some state $s \in S$, where he can choose an action $a \in A$. The agent is subsequently relocated to a new state s' determined by the transition probability distribution $T(s'|s, a)$. At each state s, an agent collects reward $R(s)$ (or, sometimes, rewards are put on state-action pairs). For example, a package-delivery robot might get a reward every time it correctly delivers a package. Note that each action defines a probability distribution of the next state; if actions were pre-determined, then we would get just a Markov chain.

In order to encourage the robot to perform the tasks that we want, and to do so in a timely manner, a standard objective considered in MDPs is to maximize *discounted reward* [14, 17, 18]. 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 $R_{tot} = \sum_{t=0} R(s_t)\gamma^t$. This guides the robot to get 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 can be described just by a *policy*, a mapping from states to actions. The goal of planning in an MDP is to determine the optimal policy: the mapping of states to actions that maximizes expected discounted reward $E[R_{tot}]$.

There are well-known algorithms for solving MDPs in time polynomial in the state space [5, 17, 18]. However, one drawback of the MDP model is that the robot receives the reward for a state every time that state is visited (or every time the robot performs that action from that state if rewards are on state-action pairs). 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 locations it has already visited. This causes an exponential increase in the size of the state space. Thus, it would be preferable to directly model the case of rewards that are given only the *first* time a state is visited [15, 16].

As a first step towards tackling this general problem, we abandon the stochastic element and restrict to deterministic, reversible actions. This leads us to study the *Discounted-Reward Traveling Salesman Problem*, in which we assume we have an undirected weighted graph (edge 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 as a Prize-Collecting Traveling Salesman (PC-TSP) or Orienteering Problem [4, 12, 11, 3, 1]. In the PC-TSP, a salesman is required to collect at least some given amount of reward (possibly zero), while minimizing the distance travelled plus the sum of rewards foregone (in a roughly equivalent problem, k-TSP, every node has a prize of one unit and the salesman is required to visit at least k nodes). In the Orienteering problem, one instead fixes a deadline D (a maximum distance that can be traveled) and aims to *maximize* total reward collected by that time.

There are several approximations known for the PC-TSP and k-TSP problems [11, 2, 9, 7, 3], the best being a $(2 + \epsilon)$-approximation due to Arora and Karakostas [2]. Most of these approximations are based on a classic Primal-Dual algorithm for the Prize Collecting Steiner Tree problem, due to Goemans and Williamson [11]. These algorithms for PC-TSP extend easily to the *unrooted* version of the Orienteering problem in which we do not fix the starting location [13, 3]. In particular, given a path of value Π but whose length is cD for some $c > 1$, we can just break the path into c pieces of length at most D, and then take the best one, whose total value will be at least Π/c. However, this doesn't work for the rooted problem because the "best piece" in the above reduction might be far from the start. Arkin et. al [1] give a constant-factor approximation to the rooted Orienteering problem for the special case of points in the plane. However, there is no previously known $O(1)$ approximation algorithm for the rooted Orienteering Problem or Discounted-Reward TSP in general graphs.

1.1.3. Result Summary 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 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 the min-cost s-t path problem (k-path problem in [8]) 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, a $1 + \lceil \alpha_{EP} \rceil$ approximation for Orienteering, and a roughly $e(\alpha_{EP} + 1)$ approximation for Discounted-Reward TSP. Using a $(2+\epsilon)$-approximation for min-cost s-t path, due to Chaudhuri et. al [8], we get constants of $2+\epsilon$, 4, and $6.75+\epsilon$ for these problems respectively.

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 7.

2. Notation and Definitions

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 *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[13, 2, 9, 7, 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 $d(s,t) + \alpha\epsilon \le \alpha(d(s,t) + \epsilon)$ 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	Current approx.	Source/Reduction
min-cost s-t path (α_{CP})	$2+\epsilon$	[8]
min-excess path (α_{EP})	$2.5+\epsilon$	$\frac{3}{2}(\alpha_{CP})-\frac{1}{2}$
max discounted-prize path (α_{DP})	$8.12+\epsilon$	$(1+\alpha_{EP})(1+1/\alpha_{EP})^{\alpha_{EP}}$
max-prize path (α_{PP})	4	$1+\lceil\alpha_{EP}\rceil$
max-prize tree (α_{PT})	8	$2\alpha_{PP}$
max-prize cycle (α_{PC})	8	$2\alpha_{PP}$
max-prize multiple-path (α_{kPP})	5	$\alpha_{PP}+1$
max discounted multiple-path (α_{kDP})	$9.12+\epsilon$	$\alpha_{DP}+1$

Figure 1. Approximation factors and reductions for our problems.

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.

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 6.3, based on a tighter analysis of the min-cost s-t path algorithm of [8]. 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[1] 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.

Let nodes in V be ordered from $v_1 = s$ through v_n in order of their distance from s. (Note that t is not necessarily the last vertex in this order). Let $d_i = d(s, v_i)$, so $d_1 \le d_2 \le \cdots \le d_n$. For convenience in the analysis, we assume all d_i are distinct (in the algorithm we can handle equal distances by breaking ties lexicographically).

3. Min-Excess Path

Let P^* be the shortest path from s to t with $\Pi(P^*) \ge k$. Let $\epsilon(P^*) = d(P^*) - d(s,t)$. Our algorithm returns a path P with $\Pi(P) \ge k$ 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 $(2.5+\epsilon)$-approximation to min-excess path using an algorithm of Chaudhuri et. al [8] for min-cost s-t path with $\alpha_{CP} = 2+\epsilon$. A brief description of the min-cost path algorithm and approximation is given in the appendix.

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

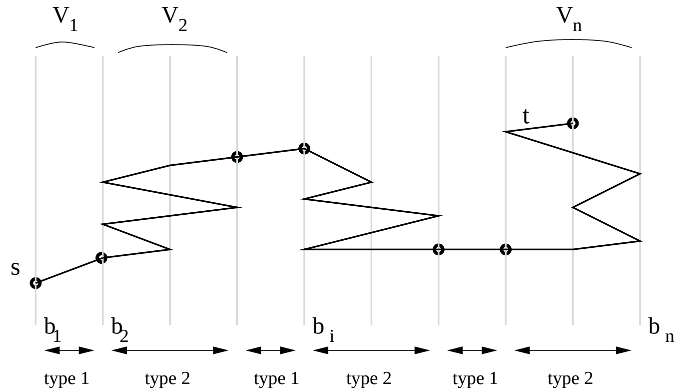

Figure 2. Segment partition of a path in graph G

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 will solve the general case by breaking the optimum path into continuous *segments* that are either monotonic (so can be found optimally as just described) or "wiggly" (generating a large amount of excess). We will 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 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). Placing a vertex at each interval boundary ensures that the length of a segment is *equal to* the integral of $f(d)$ over its interval.

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 $3\epsilon/2$.*

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

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

It follows that

$$\epsilon = \sum \ell_i - d_t \ge \sum_{i \text{ type } 2} 2\ell_i/3$$

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

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 endpoints 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 segment 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 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 $L_1 + L_2 = d_t + \epsilon$ and that (by Corollary 3.2) $L_2 \le 3\epsilon/2$. 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

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

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

4. Maximum Discounted-Prize Path

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 $d_t^{P^*} - d_t \le \epsilon$—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, $\rho(P_t^*) \ge \rho(P^*)(1 - \frac{1}{2^\epsilon})$.*

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 at least $\frac{1}{2^\epsilon}$ of the total 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 $\rho(P) \geq \Pi'(P)/2^{\alpha\epsilon}$.*

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 $v \in P$ 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.

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

□

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

We would like to compute the maximum-prize path of length at most D, starting at s. We will use the algorithm for min-excess path given in section 3 as a subroutine. Our algorithm is in Figure 4.

Algorithm for Max-Prize Path (Orienteering)

1. Guess k, the amount of prize collected by an optimum orienteering solution.
2. For each vertex v, compute min-excess path from s to v collecting prize k.
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)

As can be seen from our algorithm, we solve Max-Prize Path by directly invoking our Min-Excess Path algorithm. Our analysis consists of 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 $\frac{\pi^*}{\alpha_{PP}}$ (here 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 vertex on the optimum orienteering path at maximum distance from s. We first consider the case where the optimum orienteering path ends at t (as opposed to doubling back towards s).

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 $\frac{\epsilon(t)}{r} = \frac{D-d_t}{r} \leq \frac{D-d_v}{r}$. □

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. 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 [8] along with our result on min-excess path from section 3, we have a 4-approximation for Orienteering.

Theorem 5.3. *There 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 $\frac{D-d_v}{\alpha_{EP}}$ obtaining prize $\frac{\pi^*}{\lceil \alpha_{EP} \rceil + 1}$. Such a path has length $d_v + \frac{D-d_v}{\alpha_{EP}}$, 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 (actually the approximation factor will be $\lceil \alpha_{EP} \rceil + 1 + \epsilon$ and the running time will depend logarithmically on $\frac{1}{\epsilon}$; note that this running time is still polynomial in the size of the input). □

6. Extensions

6.1. Budget Prize Collecting Steiner Tree

In this section, we consider the tree variant of the Orienteering problem, called Max-Prize Tree in our notation. Namely, 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 [13].

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^*.

6.2. Multiple-Path Orienteering and Discounted-Reward TSP

In this section we consider a variant of the Orienteering and Discounted-Reward TSP in which we are allowed to construct up to k paths. For the Orienteering problem, each path must have length at most D. For the Discounted-Reward problem, the k robots move simultaneously, so the discounting is done based on the individual lengths of the paths.

For both problems, we apply the algorithms described in sections 4 and 5 respectively to successively 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 previously described algorithms. Using a set-cover like argument, we get the following approximation guarantees.

Theorem 6.1. *If all the paths have a common start node, the above algorithm gives a $1/(1-e^{-\alpha_{PP}})$ ($1/(1-e^{-\alpha_{DP}})$) approximation for the Multiple-Path Orienteering (resp. Discounted-Reward TSP).*

If the paths have different start nodes, the above algorithm gives a $\alpha_{PP}+1$ ($\alpha_{DP}+1$) approximation for the Multiple-Path Orienteering (resp. Discounted-Reward TSP).

Proof. We consider the Multiple-Path Orienteering problem. The corresponding result for Discounted-Reward TSP can be derived analogously.

First consider the case when all the paths have a common source. Let the reward collected by the optimal solution, but not collected by our solution by stage i, be Π_i. At least one of the paths in the optimal solution collects at least a k fraction of this reward. 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. By the end of k rounds, the total reward collected by optimal solution, but not collected by us, is at most $(1-\frac{1}{k\alpha_{PP}})^k \leq e^{-\alpha_{PP}}$, and the result follows.

Next consider the case when the paths have different sources. 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 $\Pi(A) \geq \frac{1}{\alpha_{PP}+1}\Pi(O)$. □

6.3. An Improved Approximation for Min-Excess and Maximum Discounted-Prize Path

In this section we show how to slightly improve our approximation factors for Min-Excess and Maximum Discounted-Prize Path. The min-cost s-t path algorithm of Chaudhuri et. al has the property that if the optimal path has length $L = d(s,t) + \epsilon$, then it returns a path with length at most $(1+\delta)L+\epsilon$, for any fixed small constant δ. A brief description of their algorithm and guarantee is given in the appendix.

Using this guarantee, note that if a segment of type 2 has endpoints u and v and length $\ell = d(u,v) + \epsilon'$, then the length of this segment in the path found by the dynamic program is at most $(1+\delta)\ell + \epsilon'$.

Now for any node u, let ϵ_u be the excess of the path P^* from s to u. Then, using $d(u,v) \geq d_v - d_u$ and $d_v + \epsilon_v = d_u + \epsilon_u + \ell$ we get $\epsilon' \leq \epsilon_v - \epsilon_u$. Furthermore, using $\ell \leq \frac{3}{2}(\epsilon_v - \epsilon_u)$, from Corollary 3.2, we get that the length of the segment is bounded by

$$\ell + (1 + \frac{3}{2}\delta)(\epsilon_v - \epsilon_u)$$

Summing over all segments, we get an approximation ratio of $2 + \frac{3}{2}\delta$ for Min-Excess, for any small constant δ.

Using this improved approximation for Min-excess, we get an approximation factor of roughly $6.75 + \delta$ for the Maximum Discounted-Prize Path problem.

7. Conclusions

In this paper we give constant factor algorithms for the Orienteering problem, Discounted-Reward TSP, and some of their variants. An interesting open problem is to consider other discount functions, or different deadlines for each vertex. For example, the reward collected at vertex v at time t could be given by $\rho_v = \Pi_v$ if $t < T_v$ and 0 otherwise. 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] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

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

[15] 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.

[16] 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.

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

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

Appendix: The min-cost s-t path algorithm of Chaudhuri et. al [8]

Chaudhuri et. al give a primal-dual algorithm for constructing an s-t path that collects prize at least k, and has minimum possible length. For certain good values of k, this algorithm returns a tree containing s and t that contains prize at least k and has cost at most the cost of the shortest s-t *path* collecting reward k. Let the size of this tree be $d(s,t) + \epsilon$. Then, doubling all the edges of this tree except those along the tree path from s to t, we obtain an s-t path of length at most $d(s,t) + 2\epsilon$, because the tree path from s to t has length at least $d(s,t)$.

Using techniques from Arora et. al [2], one can obtain an s-t path for any value of k, while increasing the cost by a factor of $(1+\delta)$, for any fixed constant δ. This gives an s-t path of cost at most $(1+\delta)d(s,t)+(2+\delta)\epsilon$. In particular, it implies a $(2+\delta)$-approximation for min-cost s-t path.