



Mathematics of Operations Research

Publication details, including instructions for authors and subscription information:
<http://pubsonline.informs.org>

Budgeted Prize-Collecting Traveling Salesman and Minimum Spanning Tree Problems

Alice Paul, Daniel Freund, Aaron Ferber, David B. Shmoys, David P. Williamson

To cite this article:

Alice Paul, Daniel Freund, Aaron Ferber, David B. Shmoys, David P. Williamson (2020) Budgeted Prize-Collecting Traveling Salesman and Minimum Spanning Tree Problems. *Mathematics of Operations Research* 45(2):576-590. <https://doi.org/10.1287/moor.2019.1002>

Full terms and conditions of use: <https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions>

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright 2019, INFORMS

Please scroll down for article—it is on subsequent pages



With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.

For more information on INFORMS, its publications, membership, or meetings visit <http://www.informs.org>

Budgeted Prize-Collecting Traveling Salesman and Minimum Spanning Tree Problems

Alice Paul,^a Daniel Freund,^b Aaron Ferber,^c David B. Shmoys,^c David P. Williamson^c

^aData Science Initiative, Brown University, Providence, Rhode Island 02912; ^bSloan School of Management, Massachusetts Institute of Technology, Cambridge, Massachusetts 02142; ^cOperations Research and Information Engineering, Cornell University, Ithaca, New York 14850

Contact: alice_paul@brown.edu (AP); df365@cornell.edu,  <http://orcid.org/0000-0001-8039-9805> (DF); amf272@cornell.edu (AF); dbs10@cornell.edu (DBS); davidpwilliamson@cornell.edu (DPW)

Received: October 6, 2017

Revised: July 19, 2018; December 3, 2018

Accepted: February 16, 2019

Published Online in Articles in Advance:
December 6, 2019

MSC2000 Subject Classification: Primary:
68W25; secondary: 05C85

OR/MS Subject Classification: Primary:
networks/graphs/traveling salesman; secondary:
analysis of algorithms

<https://doi.org/10.1287/moor.2019.1002>

Copyright: © 2019 INFORMS

Abstract. We consider constrained versions of the prize-collecting traveling salesman and the prize-collecting minimum spanning tree problems. The goal is to maximize the number of vertices in the returned tour/tree subject to a bound on the tour/tree cost. Rooted variants of the problems have the additional constraint that a given vertex, the root, must be contained in the tour/tree. We present a 2-approximation algorithm for the rooted and unrooted versions of both the tree and tour variants. The algorithm is based on a parameterized primal–dual approach. It relies on first finding a threshold value for the dual variable corresponding to the budget constraint in the primal and then carefully constructing a tour/tree that is, in a precise sense, just within budget. We improve upon the best-known guarantee of $2 + \epsilon$ for the rooted and unrooted tour versions and $3 + \epsilon$ for the rooted and unrooted tree versions. Our analysis extends to the setting with weighted vertices, in which we want to maximize the total weight of vertices in the tour/tree. Interestingly enough, the algorithm and analysis for the rooted case and the unrooted case are almost identical.

Funding: This work was supported by the National Science Foundation Division of Computing and Communication Foundations [Grants CCF-1522054, CCF-1526067, and CCF-1740822] Division of Civil, Mechanical and Manufacturing Innovation [Grant CMMI-1537394], and Division of Mathematical Sciences [Grant DMS-1839346]; the Air Force Office of Scientific Research; and a National Defense Science and Engineering Graduate fellowship.

Keywords: approximation algorithms • traveling salesman problem • constrained optimization

1. Introduction

In the classical traveling salesman problem (TSP), we are given an undirected graph $G = (V, E)$ with edge costs $c_e \geq 0$ for all $e \in E$. The goal is to construct a tour visiting all vertices in the graph while minimizing the cost of edges in the tour. If, however, we are given a bound on the cost of the tour, then we may not be able to visit all vertices. In particular, suppose that we are given a budget $D \geq 0$. In the *budgeted prize-collecting traveling salesman problem*, a feasible tour is a multiset of edges F such that (a) F is a tour on a subset $S \subseteq V$ and (b) the cost of the edges in F is at most D . The goal is to find a feasible tour F that maximizes $|S|$, the number of vertices visited. Here, we do not require the graph to be complete and allow a tour to visit nodes more than once. Similarly, in the *budgeted prize-collecting minimum spanning tree (MST) problem*, a feasible tree is a set of edges T such that (a) T specifies a tree spanning a subset $S \subseteq V$ and (b) the cost of the edges in T is at most D . Again, the goal is to find a feasible tree T that maximizes $|S|$. For either problem, we may consider the rooted versions, in which we are also given a vertex r with the added constraint that it must be contained in the returned solution. We say the problem is unrooted if there is no root r given.

The budgeted version of the traveling salesman problem arises naturally in many routing problems that have a distance or time constraint. For example, a bike-share system may have bike stations located around a city that need repair. Throughout the day, the system operator wants to route a repairman over his work period while maximizing the number of stations that receive maintenance. [In fact, this precise question emerged from our ongoing work with NYC Bike Share (Freund et al. [11]).] We can represent this problem as a budgeted prize-collecting traveling salesman problem. Furthermore, we can also capture the setting in which stations have varying importance; we discuss in Section 8 how to extend our algorithm to a setting in which vertices have weights and the goal is to maximize the weight of vertices visited. In Section 9, we apply our algorithm to such instances using Citi Bike data in New York City. The budgeted version of the minimum

spanning tree also arises in a range of applications, including telecommunication network design problems in which an infrastructure budget is weighed against the number of customers served.

In this paper, we present a parameterized primal–dual algorithm that achieves an approximation guarantee of 2 for all the problems mentioned above, including both the rooted and unrooted versions. The algorithm is based on a primal–dual subroutine that uses a linear programming (LP) relaxation of the problem. First, we search for a “good” value for the dual variable corresponding to the budget constraint in the primal. Having set this variable, we can then increase the other dual variables and form a forest of edges whose corresponding dual constraint is tight. For the tour problem, we then choose a tree in this forest and carefully prune it so that doubling this tree forms a tour that, in a precise sense, will be just within budget. For the tree problem, we prune edges such that the tree itself is just within budget. Last, we show that either our constructed tour/tree is within a factor of 2 of optimal or we recurse on a smaller subgraph.

1.1. Literature Review

There have been many prize-collecting variants of both the TSP and the MST problem that seek to balance the number of vertices in the tree or tour with the cost of edges used. Johnson et al. [16] characterize four main variants of prize-collecting MST problems: the Goemans and Williamson [14] minimization problem that minimizes the cost of edges plus a penalty for vertices not in the tree (this is the traditional prize-collecting objective), the net worth maximization problem that maximizes the weight of vertices in the tree minus the cost of used edges, the quota problem that minimizes the cost of a tree containing at least Q vertices, and, finally, the budget problem that maximizes the number of vertices in the tree subject to the cost of the tree being at most D . All of the variants above can be extended to a corresponding TSP version that constructs a tour rather than a tree.

Our algorithm is most similar to that of Garg [13], who presents a 2-approximation algorithm for the quota problem for MSTs, improving upon the previous results of Garg [12], Arya and Ramesh [2], and Blum et al. [4]. Johnson et al. [16] observe that a 2-approximation algorithm for the quota problem yields a $(3 + \epsilon)$ -approximation algorithm for the corresponding unrooted budget problem. To our knowledge, this was the previously best-known guarantee for the unrooted MST variant. Prior to Garg’s [12] result, Levin [18] proved a $(4 + \epsilon)$ -approximation algorithm. Our 2-approximation algorithm for the budgeted prize-collecting MST problem thus improves upon the best known approximation ratio for the unrooted version. Though the rooted version is mentioned by Johnson et al. [16], to the best of our knowledge, no guarantee for it was known prior to this work. From a technical perspective, the main distinction between our work and that of Garg [13] lies in how we find the threshold value for the dual variable. Furthermore, our overall proof relies on more precise accounting.

For the Goemans and Williamson [14] minimization problem for MSTs, Archer et al. [1] obtain a $(2 - \epsilon)$ -approximation guarantee, improving upon the long-standing bound of 2 obtained by Goemans and Williamson [14] in 1995. Furthermore, Archer et al. [1] successfully applied this algorithm to telecommunication network problems. Dilkina and Gomes [8] also study a variant of the problem in the context of wildlife conservation using mixed-integer programming. Last, Feigenbaum et al. [9] show the net worth maximization problem for MSTs is NP-hard to approximate within any constant.

To the best of our knowledge, the previous best approximation guarantee for the rooted and unrooted budgeted prize-collecting TSP arises from a special case of a result by Chekuri et al. [6]. Their work provides a $(2 + \epsilon)$ -approximation algorithm with running-time exponential in $O(1/\epsilon^2)$ for the more general orienteering problem, in which the goal is to find an $s - t$ path, where s and t are given, with bounded cost that maximizes the number of vertices visited on the path. By setting $s = t = r$, this then yields a $(2 + \epsilon)$ -approximation algorithm for the rooted budgeted prize-collecting TSP. For the unrooted case, one can iterate over all possible nodes as roots. In contrast, as we show in Section 8, our approach to the unrooted version does not rely on iterating over all possible roots. The orienteering problem itself has attracted much attention within the combinatorial optimization community, with other variants studied by Vidyarthi and Shukla [21], Chekuri and Korula [6], Chen and Har-Peled [7], Chekuri and Pál [5], and Gupta et al. [15].

There exist other adaptations of prize-collecting problems not discussed above. Specifically, Ausiello et al. [3] present a 2-approximation algorithm for an online variant of the quota problem for the TSP. Frederickson and Wittman [10] study the so-called traveling repairmen problem, in which each vertex can be visited only within a specific time window and the goal is to either maximize the number of vertices visited within a certain time period or to minimize the time visiting all vertices. They give constant-factor approximation algorithms for both variations. Last, Nagarajan and Ravi [19] study the problem of minimizing the number of tours to cover all vertices subject to each tour having bounded distance. They give a 2-approximation algorithm for tree metric distances.

This paper is structured as follows. In Section 2, we present the LP relaxation for the rooted version of the budgeted prize-collecting traveling salesman problem. In Section 3, we use this LP relaxation to develop a primal–dual subroutine that will inform our decisions. In Section 4, we use this subroutine to present an outline of the entire parameterized primal–dual algorithm and the proof of its approximation ratio, providing some intuition behind what type of tour will be near optimal. In Section 5, we prove an upper bound on the size of an optimal solution. Then, in Section 6, we show how to set the dual variable corresponding to the budget constraint, and in Section 7, we show how to construct our proposed tour that is within a factor of 2 of optimal. For ease of presentation, we present these results for the rooted version of the budgeted prize-collecting traveling salesman problem with unit weights and show how the analysis extends to the weighted case and the MST version in Section 8. In Section 8, we also show how our algorithm tackles the unrooted versions of the two problems without having to run the rooted version on all possible roots. Last, we present computational experiments in Section 9.

2. LP Formulation

As mentioned, we present our result in terms of the *rooted* budgeted prize-collecting traveling salesman problem. In this version of the problem, there is the added constraint that the returned tour must contain a root node $r \in V$. However, in Section 8, we show that our algorithm (and analysis) directly extends to the unrooted case with a few minor changes.

For each $S \subseteq V$, let $z_S \in \{0, 1\}$ be a variable representing whether the vertices in S are the ones on which the tour is constructed; for each edge $e \in E$, we let $x_e \in \mathbb{Z}^+$ be a variable representing how many copies of e to include in the tour. Then, the following is a linear programming relaxation for the rooted budgeted prize-collecting traveling salesman problem:

$$\begin{aligned} & \text{maximize} && \sum_{S \subseteq V: r \in S} |S| z_S \\ & \text{subject to} && \sum_{e: e \in \delta(S)} x_e \geq 2 \sum_{T: S \subsetneq T} z_T \quad \forall S \subsetneq V, \\ & && \sum_{e \in E} c_e x_e \leq D, \\ & && \sum_{S \subseteq V} z_S \leq 1, \\ & && z_S, x_e \geq 0. \end{aligned}$$

The first constraint states that if the tour visits the vertices in a subset T and $S \subsetneq T$, then we must have at least two edges across the cut S . The dual of this linear program is given by the following:

$$\begin{aligned} & \text{minimize} && \lambda_1 D + \lambda_2 \\ & \text{subject to} && \left(2 \sum_{T: T \subsetneq S} y_T \right) + \lambda_2 \geq |S| \quad \forall S \subseteq V: r \in S, \\ & && \left(2 \sum_{T: T \subsetneq S} y_T \right) + \lambda_2 \geq 0 \quad \forall S \subseteq V: r \notin S, \\ & && \sum_{S: e \in \delta(S)} y_S \leq \lambda_1 c_e \quad \forall e \in E, \\ & && \lambda_1, \lambda_2, y_S \geq 0. \end{aligned}$$

In order to construct a tour, we rely on a primal–dual subroutine that returns a tree containing the root. We first note that if we find $\lambda_1 \geq 0$ and $y_S \geq 0$ that satisfy the dual constraint for every edge, then we can always set λ_2 to be the maximum of 0 and $\max_{S \subseteq V: r \in S} [|S| - (2 \sum_{T: T \subsetneq S} y_T)]$ so that we have a feasible dual solution. Suppose that we first set the value of λ_1 . The primal–dual subroutine uses this value to construct a full dual solution and corresponding tree containing r . Doubling this tree to form a tour may or may not be feasible with respect to the budget constraint. In particular, small values of λ_1 will produce tours that violate the budget, and large values of λ_1 will produce tours that are far from optimal. Therefore, we need to adjust λ_1 to find a feasible solution with bounded approximation ratio. We first describe the primal–dual subroutine before presenting our overall algorithm.

3. Primal–Dual Subroutine

The primal–dual subroutine for fixed λ_1 returns a tree containing the root and is similar to the 2-approximation algorithm for the prize-collecting traveling salesman problem without a budget constraint presented by Goemans and Williamson [14]. Similarly to Goemans and Williamson [14], we define a potential of a set S as a function of the dual variables of the strict subsets of S .

Definition 1. For any subset $S \subseteq V$, we define the *potential* of S to be

$$\pi(S) := |S| - \left(2 \sum_{T: T \subsetneq S} y_T \right).$$

Definition 2. A subset $S \subseteq V$ is *neutral* if $2 \sum_{T: T \subsetneq S} y_T = |S|$, in other words, if $y_S = \frac{1}{2} \pi(S)$.

At the beginning of the primal–dual subroutine, we set all y_S to be 0 and set the collection of active sets to be all singleton nodes. Furthermore, we set $T = \emptyset$. Then, in each iteration, we increase y_S corresponding to all $S \subseteq V$ in the collection of active sets until either a dual constraint for an edge between two sets becomes tight or a set becomes neutral. If an edge becomes tight between two subsets S_1 and S_2 , we add the edge to T and replace both S_1 and S_2 in the collection of active sets by $S_1 \cup S_2$. We remark that the potential of this new set S is then equal to

$$\pi(S) = \pi(S_1) + \pi(S_2) - 2y_{S_1} - 2y_{S_2}.$$

If, instead, a set becomes neutral, we mark it inactive and remove it from the collection of active sets. Once there are no more active sets, we prune inactive sets of degree 0 or 1 that do not contain the root and return the remaining set of edges T (see Algorithm 1 and Figure 1). Pruned subsets of degree 0 correspond to components that did not reach the root. While this algorithm is similar to the algorithm of Goemans and Williamson [14], keeping the root component active is useful for our analysis and for the extension to the unrooted case in Section 8.

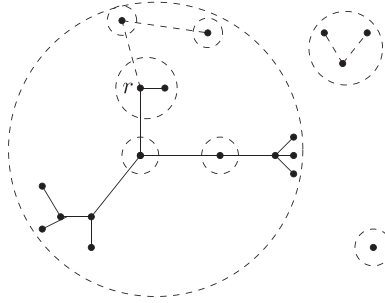
Algorithm 1 (Primal–Dual Algorithm [PD(λ_1)])

```

procedure PD( $\lambda_1 \geq 0$ )
     $y_S \leftarrow 0$ ,  $T \leftarrow \{\}$ .
    mark the subset  $\{i\}$  as active for all  $i \in V$ .
    while there exists an active subset do
        raise  $y_S$  uniformly for all active subsets  $S$  until either
        if an active set  $S$  becomes neutral then
            mark  $S$  as inactive.
        else if the dual constraint for edge  $e$  between  $S_1$  and  $S_2$  becomes tight then
             $T \leftarrow T \cup \{e\}$ .
            mark  $S = S_1 \cup S_2$  as active, remove  $S_1$  and  $S_2$  from the active subsets.
        end if
    end while
     $T' \leftarrow T$ .
    while there exists a set  $S$  marked inactive such that  $|\delta(S) \cap T'| < 2$  and  $r \notin S$  do
        remove all edges with at least one endpoint in  $S$  from  $T'$ .
    end while
    return  $T'$ 
end procedure
    
```

Because sets are removed from the set of active subsets when an incident edge becomes tight, the dual constraint for every edge will remain satisfied throughout the subroutine. Thus, λ_1 and y satisfy the dual constraints for every edge, and we can extend them to a feasible dual solution $(y, \lambda_1, \lambda_2)$. Furthermore, by construction, the dual constraint of every $e \in T$ is tight, and the edges in the set T form a forest throughout the subroutine. This implies that, after the pruning phase, T' is a tree containing r . Finally, beyond being part of a feasible solution, the construction of y also guarantees that $\mathcal{S}$, defined to be the collection of all sets active in some iteration throughout the subroutine, is laminar. This includes sets that are active even for a period of

Figure 1. Pruning the forest T to find the tree T' containing r . Dashed circles represent subsets that were marked inactive, and dashed lines represent pruned edges.



length zero (i.e., $y_S = 0$). Indeed, each set in $\mathcal{S}$, other than singletons, is the union of two disjoint other sets in $\mathcal{S}$. We define $\mathcal{S}^+ = \mathcal{S} \cup \{V\}$, which is also laminar.

4. Main Result

In this section, we give an outline of how the parameterized primal–dual algorithm uses the primal–dual subroutine described in the previous section to construct a feasible tour with bounded approximation guarantee. Assume that we have run the primal–dual subroutine with value λ_1 to find a feasible dual solution $(y, \lambda_1, \lambda_2)$, where we may not know the actual value of λ_2 . The subroutine also returns a tree of tight edges containing r . Our algorithm adjusts λ_1 so that this tree has cost very close to $\frac{1}{2}D$. In particular, the algorithm searches over possible values of λ_1 to find a threshold value such that $\text{PD}(\lambda_1^-)$ returns T' with cost $\geq \frac{1}{2}D$ and $\text{PD}(\lambda_1^+)$ returns T' with cost $< \frac{1}{2}D$. Here, $x^- = x - \varepsilon$ and $x^+ = x + \varepsilon$, where ε is arbitrarily small. Last, the algorithm prunes $T' = \text{PD}(\lambda_1^-)$ until doubling the resulting tree is a feasible solution. See Algorithm 2.

Algorithm 2 (Overview of the Algorithm)

1. Find λ_1 such that $\text{PD}(\lambda_1^-)$ returns T' with cost $\geq \frac{1}{2}D$ and $\text{PD}(\lambda_1^+)$ returns T' with cost $< \frac{1}{2}D$. See Section 6.
2. Prune $T' = \text{PD}(\lambda_1^-)$ to form a tree T_A with cost $\leq \frac{1}{2}D$ such that either T_A has cost $= \frac{1}{2}D$ or there exists an edge $e \in T'$ such that $T_A \cup \{e\}$ has cost $> \frac{1}{2}D$. See Sections 6 and 7.
3. Double T_A and return the shortcut tour.

The potentials outputted by the subroutine $\text{PD}(\lambda_1)$ help us give an upper bound on the size of an optimal solution and a lower bound on the size of our returned solution in order to provide an overall approximation guarantee. Let O^* be the subset of vertices visited by an optimal tour and F^* be the edges in that tour. Furthermore, let O be the minimal set in $\mathcal{S}^+$ that contains O^* . Because $V \in \mathcal{S}^+$, such a set always exists. In Section 5, we prove the following bound on the size of O^* .

Theorem 1. $|O^*| \leq \lambda_1 D + \pi(O)$.

In Sections 6 and 7, we explicitly show how to find the threshold value λ_1 and how to construct the tree T_A of tight edges on a subset S_A such that $r \in S_A$ and cost $\sum_{e \in T_A} c_e \leq \frac{1}{2}D$. Doubling the edges in T_A and shortcutting produces a tour on S_A that has length at most D . Furthermore, if the cost of T_A is $< \frac{1}{2}D$, we show that we can find an incident edge $\bar{e} \in T$ such that adding $\bar{e}$ to T_A creates a tree $\bar{T}$ on a subset $\bar{S}$ with cost $\sum_{e \in \bar{T}} c_e > \frac{1}{2}D$. In other words, adding one more edge to T_A makes doubling it infeasible. If instead the cost of T_A is equal to $\frac{1}{2}D$, then we simply let $\bar{T} = T_A$ and $\bar{S} = S_A$. Let Q be the maximum potential set in $\mathcal{S}^+$ containing $\bar{S}$. In Section 7, we prove the following bound on $|S_A|$.

Theorem 2. $|S_A| > \frac{1}{2}\lambda_1 D + \pi(Q) - 1$.

Thus, the parameterized primal–dual algorithm can find a feasible tour on a subset S_A , where $|S_A|$ is bounded using the potential $\pi(Q)$. If $\pi(Q)$ is large, this implies that we have found a large subset our tour can visit. For example, if $\pi(Q) \geq \lambda_2$, then $|S_A| \geq \frac{1}{2}\lambda_1 D + \lambda_2 \geq |O^*|$, by the feasibility of the dual solution. However, the value of λ_2 is unknown. Therefore, we use the potentials to bound the solution. In particular, suppose that $\pi(Q) \geq \pi(O)$; then

$$|S_A| + 1 > \frac{1}{2}[\lambda_1 D + \pi(O)] \geq \frac{1}{2}|O^*|.$$

We may assume without loss of generality (w.l.o.g.) that $|O^*|$ is even, because we can always make a copy of each vertex that has an edge of cost zero incident to the original. Hence, the above implies that if $\pi(Q) \geq \pi(O)$, then $|S_A| \geq \frac{1}{2}|O^*|$.

However, it may be the case that $\pi(Q) < \pi(O)$. Therefore, the parameterized primal–dual algorithm finds the maximal set in $\mathcal{G}^+$ with potential strictly greater than $\pi(Q)$ that contains the root r and recurses on the graph induced by this set, returning the largest feasible tour found. If $\pi(Q) < \pi(O)$, then O must be a subset of the set we recurse on. Because the subset on which we recurse does not contain $\bar{S}$, by definition of Q as the maximum potential set containing $\bar{S}$, we recurse on a strict subgraph of G , implying that we eventually reach the case that $\pi(O) \leq \pi(Q)$ and $|S_A| \geq \frac{1}{2}|O^*|$, yielding the 2-approximation guarantee. Finally, because we recurse on a strict subgraph of G , we call the subroutine at most $O(n)$ times.

Theorem 3. *The parameterized primal–dual algorithm is a 2-approximation for the rooted budgeted prize-collecting traveling salesman problem.*

5. Upper Bound

In this section, we provide the proof of Theorem 1 for any fixed λ_1 . Recall that O is the minimal set in $\mathcal{G}^+$ that contains the optimal subset O^* , and F^* is an optimal tour on O^* . Assume that we have run the primal–dual subroutine with value λ_1 to obtain a dual solution $(y, \lambda_1, \lambda_2)$. Theorem 1 states that $|O^*| \leq \lambda_1 D + \pi(O)$. The proof relies on the feasibility of the dual solution along with the following lemma.

Lemma 1. *For any $S \subseteq V$, $(2 \sum_{T: T \subseteq S} y_T) \leq |S|$.*

Proof. Any set S can be partitioned into maximal disjoint laminar subsets $S_1, S_2, \dots, S_c \in \mathcal{G}$. Therefore,

$$2 \sum_{T: T \subseteq S} y_T = 2 \sum_{i=1}^c \sum_{T: T \subseteq S_i} y_T \leq \sum_{i=1}^c |S_i| = |S|,$$

where the inequality comes from the fact that neutral subsets are marked inactive. $\square$

Proof of Theorem 1. We first note that we can partition the power set of O into subsets of $O - O^*$ and subsets that contain vertices in O^* :

$$2 \sum_{T: T \subseteq O} y_T = 2 \sum_{T: T \subseteq O - O^*} y_T + 2 \sum_{\substack{T: T \subseteq O \\ T \cap O^* \neq \emptyset}} y_T. \quad (1)$$

Thus,

$$\begin{aligned} |O| &= 2 \sum_{T: T \subseteq O} y_T + \pi(O) \\ &= 2 \sum_{T: T \subseteq O - O^*} y_T + 2 \sum_{\substack{T: T \subseteq O \\ T \cap O^* \neq \emptyset}} y_T + \pi(O) \leq |O - O^*| + 2 \sum_{\substack{T: T \subseteq O \\ T \cap O^* \neq \emptyset}} y_T + \pi(O), \end{aligned}$$

where the first line holds by the definition of potentials, the second by Equation (1), and the last by Lemma 1. Rearranging, we get

$$\begin{aligned} |O^*| &\leq 2 \sum_{\substack{T: T \subseteq O \\ T \cap O^* \neq \emptyset}} y_T + \pi(O) \leq \sum_{e \in F^*} \sum_{T: e \in \delta(T)} y_T + \pi(O) \\ &\leq \lambda_1 \sum_{e \in F^*} c_e + \pi(O) \leq \lambda_1 D + \pi(O). \end{aligned}$$

The second line holds because F^* is a tour on O^* , and so, by the minimality of O , each subset T with $y_T > 0$ that contains a subset of O^* has at least two edges in its cut $\delta(T)$. The third line holds by the dual feasibility of $(y, \lambda_1, \lambda_2)$, and the last line holds by the primal feasibility of F^* . $\square$

6. Setting λ_1

We now turn our attention to finding λ_1 and constructing a feasible tour. Our goal is to set λ_1 for the primal–dual subroutine so as to find a tree containing the root with cost very close to $\frac{1}{2}D$ and such that the set of spanned vertices has high potential. Note that λ_1 controls the cost of the edges, and as λ_1 increases, edges

become more expensive, yielding smaller connected components in the primal–dual subroutine. In particular, for $\lambda_1 = 0$, all edges go tight immediately, and for $\lambda_1 > n/(2 \min_{e \in E} c_e)$, all vertices go neutral before a single nonzero edge goes tight. If edges go tight and/or subsets go neutral at the same time, we break ties in some order. As discussed later, this ordering is important because it changes the outputted tree T' .

If a minimum spanning tree on G has cost $\leq \frac{1}{2}D$, then we double this tree to get a feasible and optimal tour. Otherwise, suppose that we have found values λ_l and λ_r ($\lambda_l < \lambda_r$) such that $\text{PD}(\lambda_l^+)$ returns T' with cost $\geq \frac{1}{2}D$ and $\text{PD}(\lambda_r^-)$ returns T' with cost $< \frac{1}{2}D$. Here, $x^- = x - \varepsilon$ and $x^+ = x + \varepsilon$, where ε is arbitrarily small.

Lemma 2. *In polynomial time, we can find a threshold value λ_1 such that $\text{PD}(\lambda_1^-)$ returns T' with cost $\geq \frac{1}{2}D$ and $\text{PD}(\lambda_1^+)$ returns T' with cost $< \frac{1}{2}D$.*

Proof. We refer to an edge going tight during the primal–dual subroutine as an edge event, and we refer to a subset going neutral as a subset event. We prove the result by induction. Assume we have values λ_l and λ_r such that, for some k , the first $k \geq 0$ events are the same when running the subroutine for any λ_1 between λ_l^+ and λ_r^- . Furthermore, assume that for each subset S , we can find values α_S and β_S such that at the end of the first k events, $y_S = \lambda_1 \alpha_S + \beta_S$ for any λ_1 between λ_l^+ and λ_r^- . Note that this is trivially true for the base case with λ_l and λ_r defined above and $k = 0$ because all y values will be zero.

To find the next event to occur, we need to find the time after the k th event that each subset will go neutral and each edge will go tight. Observe that an active set S will go neutral at time

$$\frac{1}{2}|S| - \sum_{T \subseteq S} y_T = \frac{1}{2}|S| - \sum_{T \subseteq S} [\lambda_1 \alpha_T + \beta_T],$$

an edge with exactly one endpoint in an active component will go tight at time

$$\lambda_1 c_e - \sum_{T: e \in \delta(T)} y_T = \lambda_1 c_e - \sum_{T: e \in \delta(T)} [\lambda_1 \alpha_T + \beta_T],$$

and an edge with both endpoints in different active components will go tight at time $\frac{1}{2}$ the above amount. The minimum of these values determines the next event to occur. Because all of these times are affine in λ_1 , we can divide the interval between λ_l^+ and λ_r^- into smaller subintervals such that the first $k + 1$ events are identical on these subintervals. Note that if more than two events tie, some events are *degenerate* in that those events do not occur next in any subinterval. See Figure 2.

Considering these subintervals, we either identify a threshold point λ_1 or we identify a subinterval between $\hat{\lambda}_l$ and $\hat{\lambda}_r$ such that $\text{PD}(\hat{\lambda}_l^+)$ returns T' with cost $\geq \frac{1}{2}D$ and $\text{PD}(\hat{\lambda}_r^-)$ returns T' with cost $< \frac{1}{2}D$. Furthermore, because the time of the $(k + 1)$ th event is an affine function in λ_1 , we can add this function to the affine function y_S for each active set S to get the new affine function for this y value, updating the α 's and β 's accordingly. Thus, the inductive hypothesis holds, and eventually we find such a threshold point. $\square$

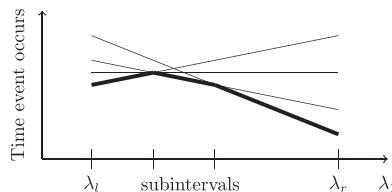
We use this threshold point λ_1 to understand the subroutine $\text{PD}(\lambda_1)$. In particular, we will show that if the subroutine breaks event ties in one way, then the returned tree is the same as $\text{PD}(\lambda_1^-)$, and if the subroutine breaks ties another way, then the resulting tree is the same as $\text{PD}(\lambda_1^+)$. Consider running the subroutine for λ_l^+ and λ_r^- and comparing event by event. We let y^+ correspond to the y variables when running $\text{PD}(\lambda_l^+)$ and let y^- correspond to the y variables when running $\text{PD}(\lambda_r^-)$. The following lemma shows that the two routines look nearly identical.

Lemma 3. *Throughout the two subroutines, the following two properties hold:*

- All active components in (V, T) are the same except for infinitesimal time.
- For all $S \subseteq V$, the difference between y_S^+ and y_S^- is infinitesimal.

Here, infinitesimal means that the time or difference goes to zero as $\varepsilon \rightarrow 0$.

Figure 2. Finding the subintervals between λ_l and λ_r where the time of the next event is in bold.



Proof. At the start of the subroutines, the properties hold because all y^+ and y^- variables are zero. Now assume that they hold at some time t into the subroutines. As argued above, the next event to occur depends on the minimum of functions linear in λ_1 . Furthermore, because the current active components are the same, the possible subset and edge events are the same.

In particular, the time for each subset to go neutral in $\text{PD}(\lambda_1^+)$ is $\frac{1}{2}|S| - \sum_{T \subseteq S} y_T^+$, which is infinitesimally different from the time for that subset to go neutral in $\text{PD}(\lambda_1^-)$. Similarly, the time for each edge to go tight is at most infinitesimally different between the two subroutines. Therefore, the next event to occur is different between the two subroutines only if two events occur at the same time for $\text{PD}(\lambda_1)$.

If the next event is the same for the two subroutines, then the active components remain the same, and we raise all active components by amounts differing by an infinitesimal amount. Therefore, the inductive properties continues to hold. Otherwise, suppose the next event is different. We consider four cases:

1. Subset X goes neutral for $\text{PD}(\lambda_1^-)$, and subset Y goes neutral for $\text{PD}(\lambda_1^+)$.
2. Edge e goes tight for $\text{PD}(\lambda_1^-)$, and edge f goes tight for $\text{PD}(\lambda_1^+)$.
3. Edge e goes tight for $\text{PD}(\lambda_1^-)$, and subset X goes neutral for $\text{PD}(\lambda_1^+)$.
4. Subset X goes neutral for $\text{PD}(\lambda_1^-)$, and edge e goes tight for $\text{PD}(\lambda_1^+)$.

In the first case, the times for both X and Y to go neutral must be infinitesimally different and the other subset goes neutral in infinitesimal time after the first. Therefore, once both X and Y are neutral, the difference of the amounts by which we have raised the y variables in $\text{PD}(\lambda_1^-)$ and in $\text{PD}(\lambda_1^+)$ is infinitesimally small, and the current active components are the same. Thus, the two inductive properties continue to hold.

Similarly for the second case, if e and f are not between the same two components, the other edge goes tight in infinitesimal time after, and the inductive properties continue to hold. Otherwise, e and f are between the same components. Thus, when e goes tight, f is no longer eligible to go tight, but the newly merged active component is the same for both subroutines. Again, the inductive properties continue to hold.

In the third case, if edge e has an endpoint in an active component that is not X , then e goes tight in infinitesimal time after X goes neutral for $\text{PD}(\lambda_1^+)$ and the components remain the same, maintaining the inductive properties. Otherwise, one endpoint of e must be in X and the other endpoint of e is in an inactive component Y , and right after e goes tight for $\text{PD}(\lambda_1^-)$, the newly merged subset has infinitesimally little potential left and goes inactive in infinitesimal time thereafter. Furthermore, if in the future any active component merges with X or Y in $\text{PD}(\lambda_1^+)$, then the edge e will immediately go tight after, maintaining that the active components stay the same. Again, this maintains the inductive properties.

Last, note that the time for a subset to go neutral has a negative slope in λ_1 , and the time for an edge to go tight has a positive slope in λ_1 . Because $\lambda_1^+ > \lambda_1^-$ and the y variables vary by an infinitesimally small amount, the fourth case cannot occur. In all cases, the inductive properties continue to hold, and the lemma holds. $\square$

The proof of Lemma 3 exactly exhibits the differences between the two subroutines: subsets that are neutral and marked inactive in $\text{PD}(\lambda_1^+)$ but have infinitesimally small potential in $\text{PD}(\lambda_1^-)$, pairs of edges that went tight between the same components, and edges in $\text{PD}(\lambda_1^-)$ that do not exist in $\text{PD}(\lambda_1^+)$. However, for the last case, these edges are between inactive components and components with infinitesimally small potential such that the merged component immediately goes inactive and that merged component remains untouched for the rest of the subroutine. Therefore, these edges are not pruned in $\text{PD}(\lambda_1^-)$ if and only if the inactive component contains the root. We summarize these structural differences in the following corollary.

Corollary 1. *The only differences in the outputted trees T' of $\text{PD}(\lambda_1^+)$ and $\text{PD}(\lambda_1^-)$ occur from the following:*

1. *a subset X and edge e that go tight at the same time in $\text{PD}(\lambda_1)$ such that in $\text{PD}(\lambda_1^+)$, X is marked inactive (and e may or may not go tight), but in $\text{PD}(\lambda_1^-)$, e goes tight before X can go inactive so neither are pruned, or*
2. *a pair of edges e and f between the same components that go tight at the same time in $\text{PD}(\lambda_1)$ such that e goes tight first for $\text{PD}(\lambda_1^+)$ but f goes tight first for $\text{PD}(\lambda_1^-)$.*

Assume that we run $\text{PD}(\lambda_1)$, breaking event ties to behave the same as $\text{PD}(\lambda_1^+)$. Then, $\text{PD}(\lambda_1)$ returns T' with cost $< \frac{1}{2}D$. However, we can think about reversing these ties one by one. In particular, consider breaking the first i ties according to $\text{PD}(\lambda_1^-)$ and then the rest by $\text{PD}(\lambda_1^+)$. By Lemma 3, reversing these ties only changes the y variables by an infinitesimally small amount. The only difference occurs when entering the pruning phase.

Thus, eventually we find the smallest i such that breaking the first i ties according to $\text{PD}(\lambda_1^-)$ returns T' with cost $\geq \frac{1}{2}D$. In other words, either we have identified a neutral subset X such that $r \notin X$ and marking X active rather than inactive changes T' to have cost $\geq \frac{1}{2}D$, or we have identified two edges e and f that tie such that adding e instead of f changes T' to have cost $\geq \frac{1}{2}D$. From here on, we assume that we always run $\text{PD}(\lambda_1)$ according to these tie-breaking rules.

7. Constructing a Tour

Continuing from the previous section, let y be the dual variables at the end of running $\text{PD}(\lambda_1)$ with the above tie-breaking rules, let T' be the set of edges after the pruning phase, and let $\mathcal{S}$ be defined as before. Last, let $\pi(S)$ be the potential of $S \subseteq V$ given y . By construction, the tree returned by $\text{PD}(\lambda_1)$ has cost $\geq \frac{1}{2}D$. Recall from Section 6 that either

1. there exists a neutral subset $X \in \mathcal{S}$ such that $r \notin X$ and if X is marked inactive, then the pruned tree will have cost $< \frac{1}{2}D$, or
2. there exist tight edges $e \in T$ and $f \notin T$ such that if we swap e with f in T , then the pruned tree will have cost $< \frac{1}{2}D$.

In the first case, when X is marked inactive, then a path of neutral subsets $N_1, N_2, \dots, N_p = X$ is pruned, yielding a component S_1 containing the root r with cost $< \frac{1}{2}D$. Similarly, in the second case, edge e prevented some neutral subsets $N_1, N_2, \dots, N_p$ from being pruned that had degree > 1 . However, by removing e and replacing it with f , these subsets are pruned, and we are left with component S_1 containing r with cost $< \frac{1}{2}D$. See Figure 3, (a) and (b).

For both cases, we use this threshold event to produce a tree T_A on a subset of vertices S_A with $r \in S_A$ and cost $\leq \frac{1}{2}D$. In doing so, we also find another tree $\bar{T}$ on a subset of vertices $\bar{S}$ containing S_A of cost $\geq \frac{1}{2}D$ such that $|S_A| \geq |\bar{S}| - 1$. Then, doubling T_A yields a feasible tour F_A that visits at most one node less than there are in $\bar{S}$. The tree $\bar{T}$ will be helpful later in obtaining a lower bound for $|S_A|$.

We start by setting T_A to be the edges in T' that span S_1 . By construction, these edges have cost $< \frac{1}{2}D$. We then try to grow T_A as much as possible along the path from S_1 to $N_1, N_2, \dots, N_p$. First, suppose that we can add this full path and the edges in T' that span each N_i to T_A without exceeding cost $\frac{1}{2}D$. Then, we set T_A to be this expanded tree and $S_A = S_1 \cup N_1 \cup \dots \cup N_p$. In other words, T_A is equal to T' at the end of $\text{PD}(\lambda_1)$. Furthermore, we set $\bar{T} = T_A$. By construction, the cost of $\bar{T}$ is equal to $\frac{1}{2}D$, and $|S_A| = |\bar{S}|$.

Otherwise, we continue to add $N_1, N_2, \dots$ to our tree until we reach a set $\bar{X} \in \{N_1, N_2, \dots, N_p\}$ such that adding the edges in T' that span $\bar{X}$ in T' to T_A implies that $\sum_{e \in T_A} c_e > \frac{1}{2}D$. In other words, we cannot add this whole subset to our tree without going over budget. Let $e = (u, v)$ be the edge that connects $\bar{X}$ to T_A in T' . If adding e to T_A already brings the cost of T_A strictly over $\frac{1}{2}D$, then we stop growing T_A and set $\bar{T} = T_A \cup \{e\}$. Otherwise, we add e to T_A and run a procedure **pick**($\bar{X}, v, \bar{T}_A$) that picks a subset of the edges in T' spanning $\bar{X}$ including v .

Specifically, the procedure **pick**(X, w, T_A) adds to T_A a set of edges in T' that span a subset of component X including w . We denote by $X_1, X_2 \in \mathcal{S}$ the two components that merged to form X , and by $e' = (u, v)$ the edge that connects X_1 and X_2 in T' . Without loss of generality, $u, w \in X_1, v \in X_2$. Furthermore, let T'_1 and T'_2 be the edges in T' with both endpoints in X_1 and X_2 , respectively. See Figure 4.

If the total cost of edges in $T_A \cup T'_1$ is greater than $\frac{1}{2}D$, then we know we should only add edges in this subtree to T_A , and we recursively invoke **pick**(X_1, w, T_A). If, instead, the total cost of edges in $T_A \cup T'_1 \cup \{e'\}$ is less than $\frac{1}{2}D$, then we can feasibly add e' and all edges in T'_1 without violating the budget. Thus, the procedure adds all these edges to T_A and recursively invokes **pick**(X_2, v, T_A) to pick the remaining edges in T'_2 . Finally, if the cost of edges in $T_A \cup T'_1$ is less than or equal to $\frac{1}{2}D$ but greater than $\frac{1}{2}D - c_{e'}$, then we cannot quite make it to T'_2 without going over budget. In this case, the procedure adds all edges in T'_1 to T_A and sets $\bar{T} = T_A \cup \{e'\}$. See Algorithm 3.

At the end of the procedure, we produce a tree T_A of cost $\leq \frac{1}{2}D$ that spans a subset S_A with $r \in S_A$ along with a tree $\bar{T}$ of cost $\geq \frac{1}{2}D$ that spans a subset $\bar{S}$, where $|\bar{S}| \leq |S_A| + 1$. Furthermore, if $|\bar{S}| = |S_A| + 1$, then $\bar{T}$ has cost $> \frac{1}{2}D$. We will use $\bar{T}$ to prove a bound on $|\bar{S}|$, which in turn will give a bound on $|S_A|$.

Figure 3. Neutral subsets pruned in each case to yield component S_1 with cost $< \frac{1}{2}D$. The subset S_1 may itself be neutral but not pruned.

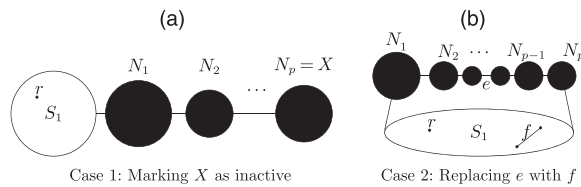
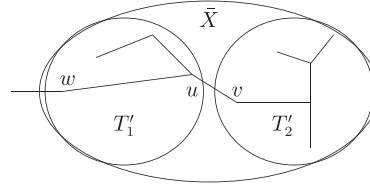


Figure 4. Illustration of the pick procedure.



Algorithm 3 (Pick Routine [pick(X, w, T_A)])

```

procedure pick( $X, w, T_A$ )
    let  $X_1, X_2$  be the two components that merged on edge  $e' = (u, v)$  to form  $X$ 
    (w.l.o.g.  $w, u \in X_1$  and  $v \in X_2$ )
    let  $T'_i$  be the edges in  $T'$  such that both endpoints are in  $X_i$  for  $i = 1, 2$ 
    if the cost of  $T_A \cup T'_1$  is  $\geq \frac{1}{2}D$  then
        call pick( $X_1, w, T_A$ )
    else if the cost of  $T_A \cup T'_1 \cup \{e'\}$  is  $< \frac{1}{2}D$  then
         $T_A = T_A \cup T'_1 \cup \{e'\}$ , and call pick( $X_2, w, T_A$ )
    else if the cost of  $T_A \cup T'_1 \cup \{e'\}$  is  $> \frac{1}{2}D$  then
         $T_A = T_A \cup T'_1$ ,  $\bar{T} = T_A \cup \{e\}$ 
    else
         $T_A = T_A \cup T'_1 \cup \{e\}$ ,  $\bar{T} = T_A$ 
    end if
end procedure
    
```

Let $Q \in \mathcal{S}^+$ be the maximum potential subset containing $\bar{S}$. Our goal is to prove Theorem 2, which states that

$$|S_A| \geq \frac{1}{2} \lambda_1 D + \pi(Q) - 1.$$

Our proof relies, informally speaking, on the vertex $\bar{v}$ at which the pick routine stopped. We define $\bar{v}$ as the (unique) vertex in $\bar{S}$ that has fewer edges incident in $\bar{T}$ than it does in T' . Though many vertices in the graph have fewer incident edges in $\bar{T}$ than in T' , the definition of the pick routine, combined with the characterization in Figure 3, (a) and (b), guarantees that only one of them is in $\bar{S}$. Furthermore, if $|\bar{S}| > |S_A|$, then $\bar{v}$ is exactly the one node in $\bar{S}$ that is not in S_A . Before proving Theorem 2, we first state the following useful lemma, the proof of which we delay to the end of the section as it closely resembles that of Goemans and Williamson [14] for the prize-collecting Steiner tree problem.

Lemma 4.

$$\sum_{e \in \bar{T}} \sum_{S: e \in \delta(S)} y_S \leq 2 \sum_{\substack{T: T \cap \bar{S} \neq \emptyset \\ \bar{v} \notin T}} y_T. \quad (2)$$

Proof of Theorem 2. By construction of $\bar{T}$, vertices in $Q - \bar{S}$ are either (i) in a single set $N \subset Q - \bar{S}$ that denotes the union of pruned subsets and sets N_i in Q that were not reached by the pick procedure (see Figure 1) or (ii) in the set $\bar{X} \in \{N_1, N_2, \dots, N_p\}$ set on which we started our pick routine. Thus, we may partition $Q = N \cup (\bar{X} - \bar{S}) \cup \bar{S}$. Notice that N is a union of neutral subsets and thus is itself neutral; that is, with S_N denoting the set of all subsets of N ,

$$|N| = 2 \sum_{T: T \in \mathcal{G}_N} y_T.$$

Similarly, let $\mathcal{G}_X$ be all subsets in $\mathcal{S}$ that are subsets of $\bar{X}$ and contain vertices in $\bar{X} - \bar{S}$. We may partition the subsets of $\bar{X}$ into sets that contain nodes in $\bar{X} - \bar{S}$ and sets that do not. We then find (because $\bar{X}$ is neutral)

$$|\bar{X}| = 2 \sum_{T: T \in \mathcal{G}_X} y_T + 2 \sum_{T: T \subseteq \bar{X} \cap \bar{S}} y_T \leq 2 \sum_{T: T \in \mathcal{G}_X} y_T + |\bar{X} \cap \bar{S}|,$$

where the inequality follows by Lemma 1. Thus, $|\bar{X} - \bar{S}| \leq 2 \sum_{T: T \in \mathcal{G}_X} y_T$.

The definition of $\bar{v}$ guarantees that any subset in $\mathcal{S}$ that contains vertices in $\bar{S}$ and $\bar{X} - \bar{S}$ also contains $\bar{v}$. Therefore, subsets T that intersect with $\bar{S}$ but do not contain $\bar{v}$ are neither in $\mathcal{S}_N$ nor in $\mathcal{S}_X$. It follows that

$$\begin{aligned} |Q| &= 2 \sum_{T: T \subseteq Q} y_T + \pi(Q) \geq 2 \sum_{\substack{T: T \cap \bar{S} \neq \emptyset \\ \bar{v} \notin T}} y_T + 2 \sum_{T: T \in \mathcal{S}_N} y_T + 2 \sum_{T: T \in \mathcal{S}_X} y_T + \pi(Q) \\ &\geq 2 \sum_{\substack{T: T \cap \bar{S} \neq \emptyset \\ \bar{v} \notin T}} y_T + |N| + |\bar{X} - \bar{S}| + \pi(Q) = 2 \sum_{\substack{T: T \cap \bar{S} \neq \emptyset \\ \bar{v} \notin T}} y_T + |Q - \bar{S}| + \pi(Q). \end{aligned}$$

Rearranging,

$$|\bar{S}| \geq 2 \sum_{\substack{T: T \cap \bar{S} \neq \emptyset \\ \bar{v} \notin T}} y_T + \pi(Q) \geq \sum_{e \in \bar{T}} \sum_{S: e \in \delta(S)} y_S + \pi(Q) = \lambda_1 \cdot \sum_{e \in \bar{T}} c_e + \pi(Q).$$

The second inequality follows from Lemma 4, whereas the final equality is due to the fact that the dual constraints are tight for all edges obtained by the primal–dual subroutine. If $|\bar{S}| = |S_A|$, this completes the proof of the theorem. Otherwise, suppose that $|\bar{S}| = |S_A| + 1$. Then, $\sum_{e \in \bar{T}} c_e > \frac{1}{2}D$, and the asserted inequality still holds. $\square$

Proof of Lemma 4. Consider a single iteration of the algorithm, and let $\mathcal{C}$ be the current set of components C such that $|\delta(C) \cap \bar{T}| \geq 1$. In other words, these are the components incident to edges in $\bar{T}$. We can partition $\mathcal{C}$ into active components $\mathcal{C}_A$ and inactive components $\mathcal{C}_I$. Furthermore, let $C_{\bar{v}}$ be the component that contains $\bar{v}$. We first show that active components are not incident to too many edges in $\bar{T}$ on average.

Starting with the graph $(V, \bar{T})$, we consider the graph obtained by shrinking each component in $\mathcal{C}$ to a single vertex and removing all vertices not in a component in $\mathcal{C}$. The remaining edges are a subset of $\bar{T}$ and form a tree on $\mathcal{C}$. Furthermore, the degree of each vertex v in this tree is $d_v = |\delta(C) \cap \bar{T}|$, where C is the corresponding component. We set R to be the set of vertices corresponding to components in $\mathcal{C}_A$ and set B be the set of vertices corresponding to components in $\mathcal{C}_I$. Because the edges in $\bar{T}$ form a tree on $\mathcal{C}$,

$$\sum_{v \in R} d_v + \sum_{v \in B} d_v \leq 2|R| + 2|B| - 2.$$

Let C be an inactive component such that $|\delta(C) \cap \bar{T}| = 1$ and $r \notin C$. Note that $|\delta(C) \cap T'| > 1$, where T' is the set of all tight edges returned by the subroutine, because otherwise C would have been pruned. Therefore, the other component incident to C was not chosen to be part of $\bar{T}$. This implies that either C is $\bar{X}$ or C is some X_1 encountered during the picking procedure such that we chose only vertices in X_1 and not X_2 . In either case, $\bar{v} \in C$.

If $C_{\bar{v}} \in \mathcal{C}_I$, then this and the component containing the root are the only possible components in B with degree equal to 1 and $\sum_{v \in B} d_v \geq 2|B| - 2$. In other words,

$$\sum_{C \in \mathcal{C}_A} |\delta(C) \cap \bar{T}| = \sum_{v \in R} d_v \leq 2|R| = 2|\mathcal{C}_A|. \quad (3)$$

Otherwise, $C_{\bar{v}} \in \mathcal{C}_A$, and the component containing the root is the only possible vertex in B with degree less than 2. Thus, $\sum_{v \in B} d_v \geq 2|B| - 1$. This implies that

$$\sum_{C \in \mathcal{C}_A} |\delta(C) \cap \bar{T}| \leq 2|R| - 1 = 2|\mathcal{C}_A| - 1. \quad (4)$$

We now prove the main claim in Equation (2) using a proof by induction. At the start of the algorithm, the left-hand side (LHS) and right-hand side (RHS) are both equal to 0. On each iteration, let $\mathcal{C}_A$ be the set of active components C such that $|\delta(C) \cap \bar{T}| \geq 1$, and let $C_{\bar{v}}$ be the component containing $\bar{v}$. Suppose that we raise y_{C_i} by ε for every active component C_i . The LHS of the claim is raised by $\sum_{C \in \mathcal{C}_A} |\delta(C) \cap \bar{T}| \varepsilon$. If $C_{\bar{v}} \in \mathcal{C}_I$, then the RHS is raised by exactly $2|\mathcal{C}_A| \varepsilon$. By Equation (3), the LHS is less than or equal to RHS. If, on the other hand, $C_{\bar{v}} \in \mathcal{C}_A$, then the RHS is raised by exactly $2(|\mathcal{C}_A| - 1) \varepsilon$. By Equation (4), the LHS is less than or equal to RHS. In both cases, the inductive statement continues to hold. $\square$

8. Extensions

In this section, we extend the algorithm and its analysis in three different directions. First, we show that the 2-approximation extends to the version of the problem in which each vertex has a weight associated to it and the

goal is to maximize the combined weight of the nodes that are part of the tour. Second, we show that our results also hold when the goal is to return a tree rather than a tour. Last, we show that the algorithm, in a natural way, also gives a 2-approximation for the unrooted problem. The usual approach to show that an approximation algorithm for a rooted problem also gives an approximation for the unrooted would be to iterate over all possible roots and take the best solution. Here, we show that the analysis for the rooted case also applies directly to the unrooted case.

8.1. Weights

Suppose that each vertex v is associated with an integer weight $p_v \geq 0$ and the goal is to find a feasible tour F that visits a subset $S \subseteq V$ to maximize $\sum_{v \in S} p_v$. We can imagine transforming this problem to an equivalent unweighted version by creating copies of each vertex v with zero-cost edges to v . Then in the primal–dual subroutine, all these added edges go tight instantaneously, yielding a weighted “cluster” with potential equal to p_v . Thus, we do not even need to perform the transformation and can actually just begin the algorithm with these weighted clusters as our initial active sets with potential equal to the weight of v . All proofs continue to hold through the equivalence to the unweighted version.

Theorem 4. *The parameterized primal–dual algorithm is a 2-approximation for the rooted budgeted prize-collecting traveling salesman problem with weights.*

8.2. Trees

Second, we discuss the extension of the algorithm to the rooted budgeted prize-collecting minimum spanning tree problem. The corresponding linear programming relaxation for the budgeted prize-collecting minimum spanning tree problem is as follows:

$$\begin{aligned} & \text{maximize} && \sum_{S \subseteq V: r \in V} |S| z_S \\ & \text{subject to} && \sum_{e: e \in \delta(S)} x_e \geq \sum_{T: S \subsetneq T} z_T \quad \forall S \subsetneq V, \\ & && \sum_{e \in E} c_e x_e \leq D, \\ & && \sum_{S \subseteq V} z_S \leq 1, \\ & && z_S, x_e \geq 0. \end{aligned}$$

The only difference is the removal of the factor of 2 from the first constraint, which changes the dual constraint associated with each subset S to become

$$\begin{aligned} \sum_{T: T \subsetneq S} y_T + \lambda_2 &\geq |S| \quad \forall S \subseteq V : r \in S, \\ \sum_{T: T \subsetneq S} y_T + \lambda_2 &\geq 0 \quad \forall S \subseteq V : r \notin S. \end{aligned}$$

Based on this change, we redefine the potential of a set to be

$$\pi(S) = |S| - \sum_{T: T \subsetneq S} y_T,$$

and a set to be neutral if $\sum_{T: T \subsetneq S} y_T = |S|$. Given these new definitions, the primal–dual subroutine runs exactly as it did before. Last, we set the threshold value λ_1 such that the tree returned by $\text{PD}(\lambda_1^-)$ has cost $\geq D$ and the tree returned by $\text{PD}(\lambda_1^+)$ has cost $< D$. We then run the pick procedure to find a tree just within budget D . It is easily verified that all proofs continue to hold for this variant.

Theorem 5. *The parameterized primal–dual algorithm is a 2-approximation for the rooted budgeted prize-collecting minimum spanning tree problem.*

8.3. Unrooted

The most intuitive way to think about the unrooted version is to think of considering every possible component in T as a possible tour, not just the component containing the root. With that interpretation, one can

summarize our results for the unrooted case as follows: the linear program then captures the objective by summing over all subsets $S \subseteq V$, and the dual has the constraint $(2 \sum_{T: T \subseteq S} y_T) + \lambda_2 \geq |S|$ for every $S \subseteq V$, rather than just ones containing the root. The definitions of potentials and neutral remain exactly the same, and the only difference in the algorithm is that we prune all sets S that are marked inactive and have $|\delta(S) \cap T'| = 1$. Each remaining component corresponds to a potential tour. Finally, instead of recursing only on the subset containing the root, we now recurse on all maximal subsets with potential $> \pi(Q)$. Because $\mathcal{S}$ is a laminar family, we recurse on disjoint sets of vertices, so the set of graphs on which we recurse is also a laminar family, and we call the subroutine $O(n)$ times.

The statements and proofs of the upper bound in Theorem 1 and the lower bound in Theorem 2 remain unchanged. The statements of Lemma 3 and Lemma 4 also continue to hold true, but the proof of Lemma 4 changes a little bit. After establishing $\sum_{v \in R} d_v + \sum_{v \in B} d_v \leq 2|R| + 2|B| - 2$, we let C be any inactive component such that $|\delta(C) \cap \bar{T}| = 1$. As before, we know that $\bar{v} \in C$. However, if $C_{\bar{v}} \in \mathcal{C}_I$, then $C_{\bar{v}}$ is the only possible component in B with degree 1 and $\sum_{v \in B} d_v \geq 2|B| - 1$; thus, we get the stronger inequality

$$\sum_{C \in \mathcal{C}_A} |\delta(C) \cap \bar{T}| = \sum_{v \in R} d_v \leq 2|R| - 1 = 2|\mathcal{C}_A| - 1.$$

Similarly, if instead $C_{\bar{v}} \in \mathcal{C}_A$, then all vertices in B have degree at least 2 and $\sum_{v \in B} d_v \geq 2|B|$, implying that

$$\sum_{C \in \mathcal{C}_A} |\delta(C) \cap \bar{T}| \leq 2|R| - 2 = 2|\mathcal{C}_A| - 2,$$

which is again a stronger inequality. The rest of the proof remains the same.

9. Computational Experiments

In this section, we complete computational experiments in order to better understand the performance of our algorithm in practice. The primal–dual algorithm as detailed in this paper was implemented in C++11 using binary search to find λ_1 . The experiments were conducted on a Dell R620 with two Intel 2.70 GHz eight-core processors and 96 GB of RAM.

The first set of graphs we used for the experiments was the 37 symmetric TSP instances with at most 400 nodes in the TSPLIB data set (Reinelt [20]). The second set of graphs was 37 weighted instances constructed using the Citi Bike network of bike-sharing stations in New York City. Each instance corresponds to a week of usage data at these stations, and the weight of a vertex corresponds to the number of broken docks at that station during that week. The number of broken docks was estimated from usage data with a probabilistic method similar to that of Kaspi et al. [17]. Details about both types of constructed instances are given in Table 1. Throughout this section, we consider only the unrooted case.

For each test graph G , we first found an upper bound on the cost of a tour by computing 2 times the cost of a minimum spanning tree in G . We then set the budget for our tour to be $f = 25\%$, 50% , or 75% of this upper bound. We denote by W the total weight of the vertices; for TSPLIB instances, this is the number of vertices. After finding our solution of weight A , we computed an upper bound on the weight of visited vertices, $U = \min(\lambda_1 D + \max_{S \in \mathcal{S}} \pi(S), W)$, and recorded the percentage optimality gap as $100 \times (U - A)/U$. Results are given in Table 2. The column headed “% weight” gives the percentage of the total weight W captured by the constructed tour, and the one headed “% budget” gives the percentage of the distance budget used after shortcutting the tree.

We report several interesting structural results. First, the average time seems to be heavily influenced by the number of edges; the bike instances were quicker to complete even though the average number of nodes was higher. However, the average time does not seem to grow with the budget (and hence with the size of the computed solution) because most of the time is spent finding λ_1 . The average optimality gap, on the other hand, does improve with the budget. This is likely because for larger budgets, the upper bound is given by W , rather than $\lambda_1 D + \max_{S \in \mathcal{S}} \pi(S)$. Also of interest is that $\max_{S \in \mathcal{S}} \pi(S)$ contributed little to our upper bound U . As

Table 1. Graph statistics for both groups of graphs averaged over all instances.

Instance type	$ V $	$ E $	Total vertex weight
TSPLIB	158.14	15,658.43	158.14
Bike	319.54	4634.77	1,302.51

Table 2. Computational results of the primal–dual algorithm for each group of graphs and budget with results averaged over all instances.

Instance type	f	Time (s)	# recursions	% opt. gap	% weight	% budget
TSPLIB	0.25	74.16	0.59	46.67	33.06	77.38
TSPLIB	0.5	72.61	0.14	41.89	58.08	69.89
TSPLIB	0.75	71.24	0.22	18.62	81.38	68.80
Bike	0.25	25.15	0.28	45.74	43.37	66.90
Bike	0.5	33.21	0.28	25.89	74.01	67.13
Bike	0.75	30.46	0.05	8.29	91.68	67.37

Note. opt., optimality.

Table 3. Improvement of optimality (opt.) gap using virtual budgets.

f	0.25	0.5	0.75
% opt. gap without virtual budgets	45.74	25.89	8.29
% opt. gap with virtual budgets	27.96	11.87	0.17

a result, our optimality gaps depend mostly on the value of λ_1 , rather than the potentials, and may be far from tight. However, the fact that, on average, we use only around two-thirds of the distance budget implies that the solutions could be improved as well. To ensure that we use a larger part of the budget, we ran further experiments on the Citi Bike instances. In these, we ran binary search over possible *virtual budgets* in the input until finding one for which the resulting tour used at least 90% of the actual budget. This significantly reduced our optimality gaps, as shown in Table 3.

10. Conclusion and Future Work

In this paper, we provide 2-approximation algorithms for the rooted and unrooted budgeted prize-collecting traveling salesman problem and prize-collecting minimum spanning tree problem that have at their base a classic primal–dual approach. The key insights are to use constructed potentials to evaluate feasible subsets of vertices to visit and to identify the structure of a good tour. In particular, we construct a tree that closely follows the structure of the laminar collection of subsets with positive dual value. Furthermore, we ensure this tree is just within budget in that adding one extra edge, and doubling the tree in the case of the tour, violates the budget.

An obvious open question seeks to improve the approximation guarantee or prove that the current guarantee is the best possible. Specifically, it would be interesting to know whether a $(3/2)$ -approximation algorithm is possible given that that is the current best guarantee for the unconstrained traveling salesman problem. Another interesting direction is to investigate whether the number of recursions, currently bounded by $O(n)$, can be reduced or eliminated altogether by inferring more from the potentials.

Acknowledgments

The authors gratefully acknowledge many helpful comments from the reviewers, including ones that pushed them to consider the rooted cases.

References

- [1] Archer A, Bateni MH, Hajiaghayi MT, Karloff H (2011) Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM J. Comput.* 40(2):309–332.
- [2] Arya S, Ramesh H (1998) A 2.5-factor approximation algorithm for the k -MST problem. *Inform. Processing Lett.* 65(3):117–118.
- [3] Ausiello G, Demange M, Laura L, Paschos V (2004) Algorithms for the on-line quota traveling salesman problem. *Inform. Processing Lett.* 92(2):89–94.
- [4] Blum A, Ravi R, Vempala S (1996) A constant-factor approximation algorithm for the k -MST problem. *Proc. 28th Annual ACM Sympos. Theory Comput.* (Association for Computing Machinery, New York), 442–448.
- [5] Chekuri C, Pál M (2005) A recursive greedy algorithm for walks in directed graphs. *Proc. 46th Annual IEEE Sympos. Foundations Comput. Sci.* (Institute of Electrical and Electronics Engineers, Piscataway, NJ), 245–253.
- [6] Chekuri C, Korula N, Pál M (2012) Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms* 8(3):Article 23.
- [7] Chen K, Har-Peled S (2006) The orienteering problem in the plane revisited. *Proc. 22nd Annual Sympos. Comput. Geometry* (Association for Computing Machinery, New York), 247–254.
- [8] Dilkina B, Gomes C (2010) Solving connected subgraph problems in wildlife conservation. Lodi A, Milano M, Toth P, eds. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, vol. 6140 (Springer, Berlin), 102–116.

- [9] Feigenbaum J, Papadimitriou CH, Shenker S (2001) Sharing the cost of multicast transmissions. *J. Comput. System Sci.* 63(1):21–41.
- [10] Frederickson GN, Wittman B (2012) Approximation algorithms for the traveling repairman and speeding deliveryman problems. *Algorithmica* 62(3–4):1198–1221.
- [11] Freund D, Norouzi-Fard A, Paul A, Henderson SG, Shmoys DB (2017) Data-driven rebalancing methods for bike-share systems. Working paper, Massachusetts Institute of Technology, Cambridge.
- [12] Garg N (1996) A 3-approximation for the minimum tree spanning k vertices. *Proc. 37th Annual Sympos. Foundations Comput. Sci.* (IEEE Computer Society, Washington, DC), 302–309.
- [13] Garg N (2005) Saving an epsilon: A 2-approximation for the k -MST problem in graphs. *Proc. 37th Annual ACM Sympos. Theory Comput.* (Association for Computing Machinery, New York), 396–402.
- [14] Goemans MX, Williamson DP (1995) A general approximation technique for constrained forest problems. *SIAM J. Comput.* 24(2):296–317.
- [15] Gupta A, Krishnaswamy R, Nagarajan V, Ravi R (2012) Approximation algorithms for stochastic orienteering. *Proc. 23rd Annual ACM-SIAM Sympos. Discrete Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia), 1522–1538.
- [16] Johnson DS, Minkoff M, Phillips S (2000) The prize collecting Steiner tree problem: Theory and practice. *Proc. 11th Annual ACM-SIAM Sympos. Discrete Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia), 760–769.
- [17] Kaspi M, Raviv T, Tzur M (2016) Detection of unusable bicycles in bike-sharing systems. *Omega* 65(December):10–16.
- [18] Levin A (2004) A better approximation algorithm for the budget prize collecting tree problem. *Oper. Res. Lett.* 32(4):316–319.
- [19] Nagarajan V, Ravi R (2012) Approximation algorithms for distance constrained vehicle routing problems. *Networks* 59(2):209–214.
- [20] Reinelt G (1991) TSPLIB—A traveling salesman problem library. *ORSA J. Comput.* 3(4):376–384.
- [21] Vidyarthi S, Shukla KK (2015) Approximation algorithms for P2P orienteering and stochastic vehicle routing problem. Working paper, Amazon, Seattle.