Prize-Collecting TSP with a Budget Constraint

Alice Paul[1], Daniel Freund[2], Aaron Ferber[3], David B. Shmoys[4], and David P. Williamson[5]

1 **Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA**
ajp336@cornell.edu
2 **Center for Applied Mathematics, Cornell University, Ithaca, NY, USA**
df365@cornell.edu
3 **Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA**
amf272@cornell.edu
4 **Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA**
dbs10@cornell.edu
5 **Operations Research and Information Engineering, Cornell University, Ithaca, NY, USA**
dw36@cornell.edu

Abstract

We consider constrained versions of the prize-collecting traveling salesman and the 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. We present a 2-approximation algorithm for these problems based on a primal-dual approach. The algorithm relies on 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 just within budget. Thereby, we improve the best-known guarantees from $3+\epsilon$ and $2+\epsilon$ for the tree and the tour version, respectively. Our analysis extends to the setting with weighted vertices, in which we want to maximize the total weight of vertices in the tour/tree subject to the same budget constraint.

1998 ACM Subject Classification G.2.2 [Mathematics of Computing] Graph Theory

Keywords and phrases Approximation Algorithms, Traveling Salesman Problem

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.62

1 Introduction

In the classical traveling salesman problem, 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 valid tour is a multiset of edges F such that (a) F specifies 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 valid 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 problem**, a valid tree is a set of edges T such that (a) T specifies a spanning tree on 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 valid tree T that maximizes $|S|$.

The budgeted version of the traveling salesman problem arises naturally in many routing problems that have a distance or time constraint. For example, a bikeshare system has bike stations located around a city that may 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 New York City Bikeshare [11]). We can represent this problem as a budgeted prize-collecting traveling salesman problem. Further, we can also capture the setting where stations have varying importance; we discuss in Section 6 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 7, we apply our algorithm to 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 where an infrastructure budget is weighed against the number of customers served.

In this paper, we present a 2-approximation algorithm for both problems. Our algorithm is based on a primal-dual subroutine which uses a linear programming relaxation of this 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 will be just within budget. For the tree problem, we prune edges such that the tree itself is just within budget. Lastly, we show that either our constructed tour/tree is within a factor of 2 of optimal or we can identify a subgraph to recurse on.

Literature Review

There have been many prize-collecting variants of both the traveling salesman problem (TSP) and the minimum spanning tree problem (MST) that seek to balance the number of vertices in the tree or tour with the cost of edges used. Johnson, Minkoff, and Phillips [16] characterize four main variants of prize-collecting MST problems: the Goemans-Williamson Minimization problem that minimizes the cost of edges plus a penalty for vertices not in the tree, 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 MST, improving upon the previous results of Garg [12], Arya and Ramesh [2], and Blum, Ravi, and Vempala [4]. Johnson et al. [16] observe that a 2-approximation algorithm to the Quota problem yields a $(3+\epsilon)$-approximation algorithm to the corresponding Budget problem. To our knowledge this was the previously best-known guarantee for the MST variant. Prior to this result, Levin [18] proved a $(4+\epsilon)$-approximation algorithm. Our 2-approximation algorithm for the budgeted prize-collecting MST thus improves upon the best known approximation ratio. While our algorithm is similar to that of Garg [13], our analysis differs in how we find the threshold value for the dual variable; further, our overall proof relies on more precise accounting.

For the Goemans-Williamson Minimization problem for MST, 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. Further, Archer et al. [1] successfully applied this algorithm to telecommunication network problems. Lastly, Feigenbaum et al. [9] show the Net Worth Maximization problem for MST is NP-hard to approximate within any constant.

To the best of our knowledge, the previous best approximation guarantee for the budgeted prize-collecting TSP arises from a special case of a result by Chekuri, Korula, and Pál [6]. Their work provides a $(2+\epsilon)$-approximation algorithm for the more general orienteering problem, where 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 $s=t$ and iterating over all vertices, this yields a $(2+\epsilon)$-approximation algorithm for the budgeted prize-collecting TSP. The orienteering problem itself has attracted much attention within the combinatorial optimization community, with other variants studied by [21], [5], [8], [7], and [15].

There exist other adaptations of prize-collecting problems not discussed above. Specifically, Ausiello, Demange, Laura, and Paschos [3] present a 2-approximation algorithm for an on-line variant of the Quota problem for the TSP. Frederickson and Wittman [10] study the so-called traveling repairmen problem, in which each vertex can only be visited 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 of this problem. Lastly, 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.

The paper is structured as follows. In Section 2, we present the linear programming (LP) relaxation for the budgeted prize-collecting traveling salesman problem. In Section 3, we use this LP to present the primal-dual subroutine that will inform our decisions and develop some intuition behind what types of tours will be near optimal. In Section 4, we show how to set the dual variable corresponding to the budget constraint, and in Section 5, we show how to construct our proposed tour. In Section 6, we prove that our overall algorithm is a 2-approximation algorithm and present computational experiments in Section 7. For ease of presentation, we present only our result for the budgeted prize-collecting traveling salesman problem but the analysis extends easily to the corresponding MST case.

2 Notation

For each $S \subseteq V$, let $z_S \in \{0,1\}$ be a variable representing whether or not we choose to tour the vertices in S, and for each edge $e \in E$, 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 budgeted prize-collecting traveling salesman problem.

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

The first constraint states that if we choose to tour a subset T such that $S \subset T$ then we must have at least two edges across the cut S. The dual of this linear program is given by

the following.

$$\begin{aligned} &\text{minimize } \Lambda_1 D + \Lambda_2 \\ &\text{subject to } (2 \sum_{T:T\subset S} y_T) + \Lambda_2 \geq |S| \quad \forall S \subseteq V \\ &\qquad \sum_{S:e\in\delta(S)} y_S \leq \Lambda_1 c_e \quad \forall e \in E \\ &\qquad \Lambda_1, \Lambda_2, y_S \geq 0 \end{aligned}$$

In order to construct a tour, we will rely on a primal-dual subroutine. We first note in Theorem 2 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 such that we have a full feasible dual solution. Suppose that we first set the value of Λ_1. The primal-dual subroutine will use this set value to construct a full dual solution and corresponding potential tours. These tours may or may not be feasible with respect to the budget constraint. Therefore, we will adjust Λ_1 to find a feasible solution with bounded approximation ratio.

3 Primal-Dual Subroutine

The primal-dual subroutine for a fixed Λ_1 is similar to the 2-approximation algorithm for the prize-collecting traveling salesman problem without a budget constraint presented by Goemans and Williamson [14]. Initially, we set all y_S to be 0 and set our collection of active sets to be all singleton nodes. Then, in each iteration, we increase y_S corresponding to all $S \subset V$ in the collection of active sets until either a dual constraint for an edge between two sets becomes tight, or a set becomes neutral.

▶ **Definition 1.** We say a subset $S \subseteq V$ is **neutral** if $2\sum_{T:T\subseteq S} y_T = |S|$.

If an edge becomes tight between two subsets S_1 and S_2, we add the edge to our solution and remove both S_1 and S_2 from the collection of active sets and add $S_1 \cup S_2$ to it. If a set becomes neutral, we mark the set as inactive and remove it from the collection of active sets. Once the collection of active sets is empty, we prune inactive sets of degree 1 and return the remaining edges in our solution (cf. Algorithm 1).

Algorithm 1 Primal-Dual Algorithm (PD(λ_1))

```
1:  procedure PD(λ1 ≥ 0)
2:      y_S ← 0, Λ1 ← λ1, T ← {}.
3:      mark all i ∈ V as active.
4:      while there exists an active subset do
5:          raise y_S uniformly for all active subsets S until either
6:          if an active set S becomes neutral then
7:              mark S as inactive.
8:          else if the dual constraint for edge e between S1 and S2 becomes tight then
9:              T ← T ∪ {e}.
10:             mark S = S1 ∪ S2 as active, remove S1 and S2 from the active subsets.
11:     T' ← T.
12:     while there exists a set S marked inactive such that |δ(S) ∩ T'| = 1 do
13:         remove all edges with at least one endpoint in S from T'.
        return two of each edge in T'.
```

Properties of the algorithm PD(λ_1) (by construction)

1. The algorithm terminates in polynomial time.
2. Throughout the algorithm, T is a forest, and by extension T' is a forest.
3. For all edges, the corresponding dual constraint is satisfied.
4. For all $e \in T$, the dual constraint for e is tight.

▶ **Theorem 2.** *Given $\lambda_1 \geq 0$, let y be as created by the algorithm. Then, there exists a value $\lambda_2 \geq 0$ such that $(y, \lambda_1, \lambda_2)$ is a feasible dual solution.*

Proof. Since all edge constraints are satisfied, we may set λ_2 to the maximum of zero and

$$\min_{S \subseteq V} \left[|S| - (2 \sum_{T:T \subset S} y_T) \right].$$

By construction, all dual constraints will be satisfied. ◀

3.1 Analysis

In this section, we assume that we have set $\Lambda_1 = \lambda_1$ in the primal-dual subroutine such that we produced a feasible dual solution $(y, \lambda_1, \lambda_2)$ (where we may not know the actual value of λ_2). We let $\mathcal{S}$ be the collection of sets that were active in some iteration of the algorithm and let $\mathcal{S}^+ = \mathcal{S} \cup \{V\}$. Since any set in $\mathcal{S}$ is either a single node or the union of other sets in $\mathcal{S}$, this is a laminar collection.

▶ **Lemma 3.** *For any $S \subseteq V$, $(2 \sum_{T:T \subseteq S} y_T) \leq |S|$.*

Proof. Any set S can be divided into maximal disjoint laminar sets $S_1, S_2, \ldots, S_c \in \mathcal{S}$. 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 we make inactive any neutral subset. ◀

We first define a potential $\pi(S)$ for each subset $S \subseteq V$. These values will help us find an upper bound on the size of a feasible tour.

▶ **Definition 4.** For any subset $S \subseteq V$, we define the **potential** of S to be

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

For a set $S \in \mathcal{S}$, $\pi(S)$ is exactly equal to twice the amount that we could have increased y_S until S went neutral. In particular, if S was formed by the union of S_1 and S_2, then

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

If S_2 went inactive before merging with S_1, then this simplifies to $\pi(S) = \pi(S_1) - 2y_{S_1}$.

Given these potentials and our constructed dual solution, we give a bound on the size of an optimal solution.

▶ **Theorem 5.** *Let $O^\star$ be an optimal subset of vertices to tour and $F^\star$ be the edges in an optimal tour on $O^\star$. Further, let O be the minimal set in $\mathcal{S}^+$ that contains $O^\star$. Since $V \in \mathcal{S}^+$, such a set always exists. Then,*

$$|O^\star| \leq \lambda_1 D + \pi(O).$$

Proof. We provide the proof in the full version of the paper. ◀

Given the bound in Theorem 5, we argue that to construct a good tour we should try to find a tree $\bar{T}$ with cost close to $\frac{1}{2}D$ such that the set $\bar{S}$ of spanned vertices has high potential. Then, doubling this tree will give a feasible tour close to optimal. In order to find such a tree, we first rely on finding a good value of Λ_1.

4 Setting Λ_1

Our goal is to set Λ_1 so as to find a tree with cost very close to $\frac{1}{2}D$. 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 $\Lambda_1 > n/(2\min_{e:c_e>0} c_e)$ all vertices go neutral before a single non-zero edge goes tight. When edges go tight and subsets go neutral at the same time, we may assume that subset events are considered first. Further, we assume that we break edge/subset ties using cost/size and then some known ordering (e.g. lexicographical).

If a minimum spanning tree on the graph 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 l and r $(l < r)$ such that when we run PD(l^+) the largest component in T' has cost $\geq \frac{1}{2}D$ and when we run PD(r^-) the largest component in T' has cost $< \frac{1}{2}D$. Here, $x^- = x - \varepsilon$ and $x^+ = x + \varepsilon$ where ε is infinitesimally small.

▶ **Lemma 6.** *In polynomial time, we can find a threshold value λ_1 such that when we run PD(λ_1^-) the largest component in T' has cost $\geq \frac{1}{2}D$ and when we run PD(λ_1^+) the largest component in T' has 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. Assume we have values l and r such that the first k events are the same when running the subroutine for any Λ_1 between l^+ and r^-. Further, 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 l^+ and r^-. Note that this is trivially true for the base case with l and r defined above and $k = 0$ since all y values will be zero.

To find the next event to occur, we need to find the time after the kth 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 will determine the next event to occur. Since all these times are affine in Λ_1, we can divide the interval between l^+ and r^- into smaller subintervals such that the first $k+1$ events will be identical on these subintervals. See Figure 1.

By looking at these subintervals, either we identify a threshold point λ_1 or there exists a subinterval between l^+_{new} and r^-_{new} such that when we run PD(l^+_{new}) the largest component in T' has cost $\geq \frac{1}{2}D$ and when we run PD(r^-_{new}) the largest component in T' has cost $< \frac{1}{2}D$.

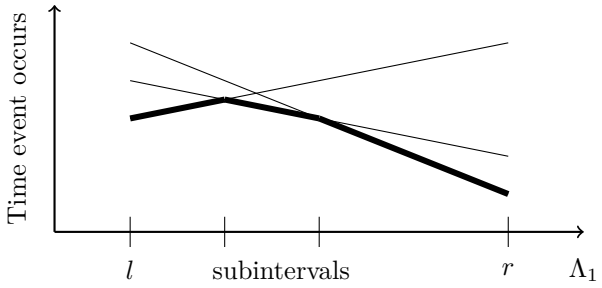


Figure 1 Finding the subintervals between l and r where the time of the next event is in bold.

Further, since 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 can find a threshold point λ_1. ◀

We use this threshold point λ_1 to understand the subroutine for $\mathrm{PD}(\lambda_1)$. Consider running the subroutine for λ_1^+ and λ_1^- and comparing event by event. We let y^+ correspond to the y variables when running $\mathrm{PD}(\lambda_1^+)$ and y^- to the y variables when running $\mathrm{PD}(\lambda_1^-)$.

▶ **Lemma 7.** *Throughout the two subroutines, the following two properties hold:*
- *All active components in (V, T) are the same.*
- *For all $S \subseteq V$, the difference between y_S^+ and y_S^- is infinitesimally small.*

Proof. At the start of the subroutines this is true since all y^+ and y^- variables are zero. Now assume that this is true 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. Further, since 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 $\mathrm{PD}(\lambda_1^+)$ is $\frac{1}{2}|S| - \sum_{T \subseteq S} y_T^+$, and is infinitesimally different from the time for that subset to go neutral in $\mathrm{PD}(\lambda_1^-)$. Similarly, the time for each edge to go tight is infinitesimally different between the two subroutines. Therefore, the next event to occur is only different between the two subroutines if two events occur at the same time for $\mathrm{PD}(\lambda_1)$.

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

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

In the first case, the times for both X and Y to go neutral must be infinitesimally different and the other subset will go neutral immediately after the first. Therefore, after both X and Y go neutral, the amount that we have raised all y variables will be infinitesimally different and the current active components will be the same. Thus, the two inductive properties will continue to hold.

Similarly for the second case, if e and f are not between the same two components, the other edge will go tight immediately after, and the inductive properties will 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 will be the same for both subroutines. Again, the inductive properties will continue to hold.

In the third case, if edge e has an endpoint in an active component that is not X, then e will go tight immediately after X goes neutral for $\text{PD}(\lambda_1^+)$ and the components will 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, and right after e goes tight for $\text{PD}(\lambda_1^-)$, the newly merged subset will have infinitesimally small remaining potential and will go inactive immediately. Again, this maintains the inductive properties.

Lastly, 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. Since $\lambda_1^+ > \lambda_1^-$ and the y variables are infinitesimally different, the fourth case cannot occur. In all cases, the inductive properties continue to hold and the lemma holds. ◀

The proof of Lemma 7 exactly exhibits the differences between the two subroutines. First, there may be subsets that are neutral and marked inactive in $\text{PD}(\lambda_1^+)$ but have infinitesimally small potential in $\text{PD}(\lambda_1^-)$. Second, there may be pairs of edges that went tight between the same components. Lastly, there may be edges in $\text{PD}(\lambda_1^-)$ that do not exist in $\text{PD}(\lambda_1^+)$. However, these edges are between inactive components and components with infinitesimally small potential. Therefore, these edges will be pruned in $\text{PD}(\lambda_1^-)$ and will not contribute to the component of size $\geq \frac{1}{2}D$.

Since we assume we break ties by considering subsets before edges and lower weight edges first, $\text{PD}(\lambda_1)$ will behave the same as $\text{PD}(\lambda_1^+)$. Therefore, the largest component in T' when running $\text{PD}(\lambda_1)$ has 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 the analysis in Lemma 7, reversing these ties will not change the y variables or active components. The only difference will be going into the pruning phrase.

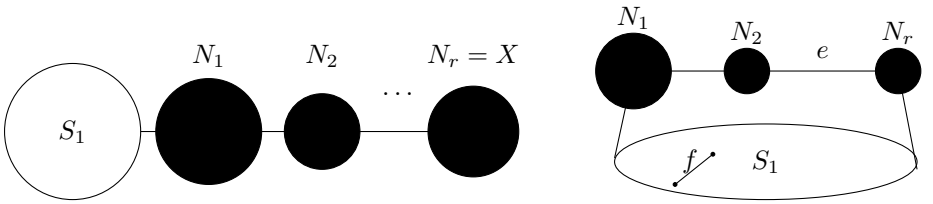
Thus, eventually we find the smallest k such that breaking the first k ties according to $\text{PD}(\lambda_1^-)$ yields a component of size $\geq \frac{1}{2}D$. In other words, we have either identified a neutral subset S such that marking S active rather than inactive changes the largest component to have size $\geq \frac{1}{2}D$ or we have identified two edges e and f that tie such that adding e instead of f changes the largest component to have size $\geq \frac{1}{2}D$. From here on, we assume that we always run $\text{PD}(\lambda_1)$ according to these tie-breaking rules.

5 Constructing a Tour

Let y be all of the dual variables for $\text{PD}(\lambda_1)$, let T' be the set of edges after the pruning phase, and let $\mathcal{S}$ be defined as before. Lastly, let $\pi(S)$ be the potential of $S \subseteq V$ given y. By construction, the largest component returned by $\text{PD}(\lambda_1)$ has size $\geq \frac{1}{2}D$. Recall from Section 4 that either

1. there exists a neutral subset $X \in \mathcal{S}$ such that if X is marked inactive then the largest component in T' has 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 largest component has size $< \frac{1}{2}D$.

In the first case, when X is marked inactive, then a path of neutral subsets $N_1, N_2, \ldots, N_r = X$ is pruned yielding a component S_1 with cost $< \frac{1}{2}D$. Similarly, in the second case, having the edge e prevented some neutral subsets $N_1, N_2, \ldots, N_r$ 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 with cost $< \frac{1}{2}D$. See Figures 2a and 2b.



(a) Case 1: Marking X as inactive.

(b) Case 2: Replacing e with f.

Figure 2 Neutral subsets pruned in each case to yield component S_1 with cost $< \frac{1}{2}D$.

For both cases, we will use this threshold event to produce a tree T_A on a subset of vertices S_A of cost $\leq \frac{1}{2}D$. In doing so, we will also find another tree $\bar{T}$ on a subset of vertices $\bar{S}$ of cost $\geq \frac{1}{2}D$ such that $|S_A| \geq |\bar{S}| - 1$. Then, doubling T_A will yield a feasible tour F_A that visits almost as many vertices as in $\bar{S}$. The tree $\bar{T}$ will be helpful 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 will then try to grow T_A as much as possible along the path from S_1 to $N_1, N_2, \ldots, N_r$. First, suppose that we can add this full path and the edges that span each N_i to T_A without going over cost $\frac{1}{2}D$. Then, we set T_A to be this expanded tree and $S_A = S_1 \cup N_1 \cup \ldots \cup N_r$. Further, we set $\bar{T}$ to be the edges in T' in the corresponding component at the end of $\mathrm{PD}(\lambda_1)$. By construction, the cost of $\bar{T}$ is $\geq \frac{1}{2}D$ and $|S_A| \geq |\bar{S}|$.

Otherwise, we continue to add N_1, N_2, ... to our tree until we reach a component $\bar{X} \in \{N_1, N_2, \ldots, N_r\}$ such that adding the edges that span $\bar{X}$ 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 $\mathbf{pick}(\bar{X}, v, \bar{T})$ that will pick a subset of the edges spanning $\bar{X}$ including v.

Specifically, the procedure $\mathbf{pick}(X, w, T_A)$ 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, $u \in X_1, v \in X_2$. Further, let T'_1 and T'_2 be the edges in T' with both endpoints in X_1 and X_2, respectively. See Figure 3.

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 $\mathbf{pick}(X_1, w, T_A)$. 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 all edges in T'_1 and e' without going over budget. Thus, the procedure adds all these edges to T_A and recursively invokes $\mathbf{pick}(X_2, v, T_A)$ 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 $\bar{T} = T_A \cup \{e'\}$.

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 along with a tree $\bar{T}$ of cost $\geq \frac{1}{2}D$ that spans a subset $\bar{S}$ where $|\bar{S}| \leq |S_A| + 1$. Further, if $|\bar{S}| = |S_A| + 1$, then $\bar{T}$ has cost $> \frac{1}{2}D$.

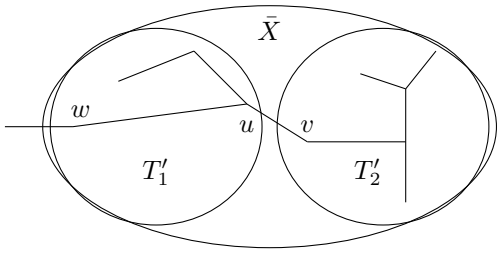


Figure 3 Illustration of the pick procedure.

5.1 Properties of $\bar{T}$

We have now constructed a tree T_A of cost $\leq \frac{1}{2}D$ that spans a subset S_A along with a tree $\bar{T}$ of cost $\geq \frac{1}{2}D$ that spans a subset $\bar{S}$ containing at most one more vertex than S_A. Further, 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 $|S_A|$.

Let $Q \in \mathcal{S}$ be a subset containing $\bar{S}$. Since $\bar{S}$ is a subset of an active set, such a set will always exist. Our goal will be to show that

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

Let $\bar{v} = \bar{S} - S_A$ (possibly equal to $\emptyset$). We first state the following useful lemma. Since the proof closely resembles that of Goemans and Williamson [14] for the Prize-Collecting Steiner Tree Problem, we defer the proof to the full version of the paper.

▶ **Lemma 8.**

$$\sum_{e \in \bar{T}} \sum_{S : e \in \delta(S)} y_S \leq 2 \sum_{\substack{T : T \cap \bar{S} \neq \emptyset \\ \bar{v} \notin T}} y_T. \tag{1}$$

▶ **Theorem 9.** *Let Q be any set in $\mathcal{S}$ containing $\bar{S}$. Then,*

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

Proof. Vertices in $Q - \bar{S}$ are either in a neutral subset N (the combination of pruned subsets and N_i not reached) or are in the set $\bar{X} \in \{N_1, N_2, \ldots, N_r\}$ that we started our pick routine on. Let $\mathcal{S}_N$ be all subsets in $\mathcal{S}$ that are subsets of N. By the definition of neutral subsets,

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

Similarly, let $\mathcal{S}_X$ be all subsets in $\mathcal{S}$ that are subsets of $\bar{X}$ and contain vertices in $\bar{X} - \bar{S}$. These are all the previously active subsets T such that $y_T > 0$ and T contains vertices in $\bar{X} - \bar{S}$ before the set $\bar{X}$ went neutral. Thus,

$$|\bar{X} - \bar{S}| \leq 2 \sum_{T : T \in \mathcal{S}_X} y_T.$$

Any subset in $\mathcal{S}$ that contains vertices in $\bar{S}$ and $\bar{X} - \bar{S}$ must contain v. Therefore, the only subsets of Q that are not in $\mathcal{S}_N$ or $\mathcal{S}_X$ are those that contain a subset of $\bar{S}$ but do not

contain $\bar{v}$. In other words,

$$|Q| = 2 \sum_{T:T\subseteq Q} y_T + \pi(Q)$$
$$\geq 2 \sum_{\substack{T:T\cap\bar{S}\neq\emptyset \\ \bar{v}\notin T}} y_T + 2 \sum_{T\in\mathcal{S}_N} y_T + 2 \sum_{T:T\in\mathcal{S}_X} y_T + \pi(Q) \quad \geq 2 \sum_{\substack{T:T\cap\bar{S}\neq\emptyset \\ \bar{v}\notin T}} y_T + |Q - \bar{S}| + \pi(Q)$$

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 8 and the third from property 4 of the algorithm. If $|\bar{S}| = |S_A|$, then we are done. Otherwise, suppose that $|\bar{S}| = |S_A|+1$. Then, $\sum_{e\in\bar{T}} c_e > \frac{1}{2}D$. In either case, the theorem holds. ◀

6 Approximation Ratio

The previous sections show that we can produce a feasible tour F_A on a subset S_A such that $|S_A| > \frac{1}{2}\lambda_1 D + \pi(Q) - 1$, where Q is the set in $\mathcal{S}$ of maximum potential that contains $\bar{S}$. Recall from Theorem 5, that for an optimal subset of vertices $O^\star$, $|O^\star| \leq \lambda_1 D + \pi(O)$, where O is the minimal subset in $\mathcal{S}^+$ that contains $O^\star$. Suppose that $\pi(Q) \geq \pi(O)$. In this case,

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

Without loss of generality, assume $|O^\star|$ is even (we can always make a copy of each vertex that has an edge of cost zero incident to the original). This implies that $|S_A| \geq \frac{1}{2}|O^\star|$.

On the other hand, suppose that $\pi(Q) < \pi(O)$. By the definition of Q, $Q \not\subseteq O$ since Q was the set of maximum potential that contained $\bar{S}$. Thus, either O is contained in a laminar set that is a strict subset of Q (and does not contain all vertices in $\bar{S}$) or O is disjoint from Q. By looking at the maximal sets with potential higher than $\pi(Q)$, we can recurse on each disjoint subgraph and return the best solution found. Overall, this shows that we can find a feasible tour F_A on a subset S_A such that $|S_A| \geq \frac{1}{2}|O^\star|$.

▶ **Theorem 10.** *The described algorithm is a 2-approximation for the budgeted prize-collecting traveling salesman problem.*

To see that this algorithm extends to the weighted version, imagine creating copies of each vertex v with zero cost edges to v. Since all these edges will go tight instantaneously in the primal-dual subroutine, we can actually just begin the algorithm with these weighted "clusters" as our initial active sets with potential equal to the weight of v.

7 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 2.70GHz 8-core processors and 96GB of RAM.

The first set of graphs we used for the experiments are the 37 symmetric TSP instances with at most 400 nodes in the TSPLIB data set [20]. The second set of graphs are 37 weighted

Table 1 Graph statistics for each group of graphs averaged over all instances.

Instance Type	$\|V\|$	$\|E\|$	Total Vertex Weight
TSPLIB	158.14	15658.43	158.14
Bike	319.54	4634.77	1302.51

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

instances constructed using the Citi Bike network of bikesharing 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 the usage data using a similar probabilistic method to that of Kaspi, Raviv, and Tzur [17]. Details about both types of constructed instances are given in Table 1.

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. W denotes the total weight of the vertices, for TSPLIB instances, the number of vertices. After finding our solution of weight A, we compute an upper bound on the weight of visited vertices $U = \min(\lambda_1 D + \max_{S \in \mathcal{S}} \pi(S), W)$ and record the percent optimality gap as $100 \times (U - A)/U$. Results are given in Table 2. Column 6 gives the percentage of the total weight W captured by the constructed tour, and Column 7 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 outputted solution) since most of the time is spent finding the value of Λ_1. The average optimality gap, on the other hand, does improve with the budget. This is likely due to the fact that for larger budgets the upper bound is given by W. Also of interest is that $\max_{S \in \mathcal{S}} \pi(S)$ contributed little to our upper bound U. As 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 only use around $2/3$ 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 with which the resulting tour uses at least 90% of the actual budget. This reduced our optimality gaps from 45.74%, 25.89%, and 8.29% to 27.96%, 11.87%, and 0.17%, respectively. Lastly, it is interesting that the algorithm rarely ever needs to recurse on a subgraph.

8 Conclusion and Future Work

In this paper, we provide a 2-approximation algorithm for the budgeted prize-collecting traveling salesman problem that has at its base a classic primal-dual approach. The key insights are to use constructed potentials to evaluate potential subsets to tour 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. Further, we ensure this tree is just within budget in that adding one extra edge will make doubling the tree an infeasible tour. An obvious open question seeks to improve the approximation guarantee or prove the current guarantee is the best possible. Specifically, it would be interesting to know whether or not a (3/2)-approximation algorithm is possible given that that is the current best guarantee for the unconstrained traveling salesman problem. Another interesting direction would be to see if one can avoid recursing by inferring more from the potentials.

References

1 Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal on Computing*, 40(2):309–332, 2011.

2 Sunil Arya and Hariharan Ramesh. A 2.5-factor approximation algorithm for the k-MST problem. *Information Processing Letters*, 65(3):117–118, 1998.

3 G. Ausiello, M. Demange, L. Laura, and V. Paschos. Algorithms for the on-line quota traveling salesman problem. *Information Processing Letters*, 92(2):89–94, 2004.

4 Avrim Blum, Ramamurthy Ravi, and Santosh Vempala. A constant-factor approximation algorithm for the k-MST problem. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*, pages 442–448. ACM, 1996.

5 Chandra Chekuri and Nitish Korula. Approximation algorithms for orienteering with time windows. *arXiv preprint arXiv:0711.4825*, 2007.

6 Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. *ACM Transactions on Algorithms (TALG)*, 8(3):23, 2012.

7 Chandra Chekuri and Martin Pal. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 245–253. IEEE, 2005.

8 Ke Chen and Sariel Har-Peled. The orienteering problem in the plane revisited. In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, pages 247–254. ACM, 2006.

9 Joan Feigenbaum, Christos H. Papadimitriou, and Scott Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001.

10 Greg N. Frederickson and Barry Wittman. Approximation algorithms for the traveling repairman and speeding deliveryman problems. *Algorithmica*, 62(3-4):1198–1221, 2012.

11 Daniel Freund, Ashkan Norouzi-Fard, Alice Paul, Shane G. Henderson, and David B. Shmoys. Data-driven rebalancing methods for bike-share systems. Working Paper, 2017.

12 N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS'96, pages 302–, Washington, DC, USA, 1996. IEEE Computer Society.

13 Naveen Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing (STOC)*, pages 396–402. ACM, 2005.

14 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

15 Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for stochastic orienteering. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1522–1538. SIAM, 2012.

16 David S. Johnson, Maria Minkoff, and Steven Phillips. The prize collecting Steiner tree problem: theory and practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 760–769. Society for Industrial and Applied Mathematics, 2000.

17 Mor Kaspi, Tal Raviv, and Michal Tzur. Detection of unusable bicycles in bike-sharing systems. *Omega*, 65:10–16, 2016.

18 Asaf Levin. A better approximation algorithm for the budget prize collecting tree problem. *Operations Research Letters*, 32(4):316–319, 2004.

19 Viswanath Nagarajan and R. Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 59(2):209–214, 2012.

20 Gerhard Reinelt. TSPLIB – a traveling salesman problem library. *ORSA Journal on Computing*, pages 376–384, 1991.

21 Shalabh Vidyarthi and Kaushal K. Shukla. Approximation algorithms for P2P orienteering and stochastic vehicle routing problem. *arXiv preprint arXiv:1501.06515*, 2015.