2-Approximation for Prize-Collecting Steiner Forest

ALI AHMADI, Computer Science, University of Maryland, College Park, United States
IMAN GHOLAMI, Computer Science, University of Maryland, College Park, United States
MOHAMMADTAGHI HAJIAGHAYI, Computer Science, University of Maryland, College Park, United States
PEYMAN JABBARZADE, Computer Science, University of Maryland, College Park, United States
MOHAMMAD MAHDAVI, Computer Science, University of Maryland, College Park, United States

Approximation algorithms for the prize-collecting Steiner forest (PCSF) problem have been a subject of research for more than three decades, starting with the seminal works of Agrawal et al. and Goemans and Williamson on Steiner forest and prize-collecting problems. In this article, we propose and analyze a natural deterministic algorithm for PCSF that achieves a 2-approximate solution in polynomial time. This represents a significant improvement compared to the previously best known algorithm with a 2.54-approximation factor developed by Hajiaghayi and Jain in 2006. Furthermore, Könemann et al. have established an integrality gap of at least 9/4 for the natural LP relaxation for PCSF. However, we surpass this gap through the utilization of an iterative algorithm and a novel analysis technique. Since 2 is the best known approximation guarantee for the Steiner forest problem, which is a special case of PCSF, our result matches this factor and closes the gap between the Steiner forest problem and its generalized version, PCSF.

CCS Concepts: • **Mathematics of computing → Graph algorithms**; **Approximation algorithms;**

Additional Key Words and Phrases: Steiner forest, prize-collecting, approximation algorithm

ACM Reference Format:
Ali Ahmadi, Iman Gholami, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Mohammad Mahdavi. 2025. 2-Approximation for Prize-Collecting Steiner Forest. *J. ACM* 72, 2, Article 17 (April 2025), 27 pages. https://doi.org/10.1145/3722551

1 Introduction

The Steiner forest problem, also known as the generalized Steiner tree problem, is a fundamental NP-hard problem in computer science and a more general version of the Steiner tree problem. In this problem, given an undirected graph $G = (V, E, c)$ with edge costs $c : E \to \mathbb{R}_{\geq 0}$ and a set of pairs of vertices $\mathcal{D} = \{(v_1, u_1), (v_2, u_2), \ldots (v_k, u_k)\}$ called *demands*, the objective is to find a subset

This work was partially supported by DARPA QuICC, ONR MURI 2024 award on Algorithms, Learning, and Game Theory, Army-Research Laboratory (ARL) grant W911NF2410052, and NSF AF:Small grants 2218678, 2114269, 2347322.
Authors' Contact Information: Ali Ahmadi, Computer Science, University of Maryland, College Park, Maryland, United States; e-mail: ahmadia@umd.edu; Iman Gholami, Computer Science, University of Maryland, College Park, Maryland, United States; e-mail: igholami@umd.edu; MohammadTaghi Hajiaghayi, Computer Science, University of Maryland, College Park, Maryland, United States; e-mail: hajiagha@cs.umd.edu; Peyman Jabbarzade, Computer Science, University of Maryland, College Park, Maryland, United States; e-mail: peymanj@umd.edu; Mohammad Mahdavi, Computer Science, University of Maryland, College Park, Maryland, United States; e-mail: mahdavi@umd.edu.

This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2025 Copyright held by the owner/author(s).
ACM 0004-5411/2025/04-ART17
https://doi.org/10.1145/3722551

of edges with the minimum total cost that connects v_i to u_i for every $i \leq k$. In this article, our focus is on the **prize-collecting Steiner forest (PCSF)** problem, which is a generalized version of the Steiner forest problem.

Balas [6] first introduced general prize-collecting problems in 1989 and Bienstock et al. [9] developed the first approximation algorithms for these problems. In the prize-collecting version of the Steiner forest problem, we are given an undirected graph $G = (V, E, c)$ with edge costs $c : E \to \mathbb{R}_{\geq 0}$ and a set of pairs of vertices $\mathcal{D} = \{(v_1, u_1), (v_2, u_2), \ldots (v_k, u_k)\}$ called *demands*, along with non-negative penalties π_{ij} for each demand (i, j). The objective is to find a subset of edges and pay their costs, while also paying penalties for the demands that are not connected in the resulting forest. Specifically, we aim to find a subset of demands Q and a forest F such that if a demand (i, j) is not in Q, its endpoints i and j are connected in F, while minimizing the total penalty of the demands in Q and the sum of the costs of the edges in F. Without loss of generality, we assign a penalty of 0 to pairs that do not represent a demand, ensuring that there is a penalty associated with each pair of vertices. This allows us to define the penalty function $\pi : V \times V \to \mathbb{R}_{\geq 0}$, where $V \times V$ represents the set of all unordered pairs of vertices with $i \neq j$. In this article, we significantly improve the approximation factor of the best-known algorithm for PCSF.

For the Steiner forest problem, the first approximation algorithm was introduced by Agrawal et al. [2]. Their algorithm addressed a more generalized version of the Steiner forest problem and achieved a 2-approximation for Steiner forest. Later, Goemans and Williamson [16] provided a simplified simulation of their algorithm, which yields a $(2-\frac{2}{n})$-approximate solution for the Steiner forest problem, where n is the number of vertices.[1] However, no further advancements have been made in improving the approximation factor of this problem since then. There has been a study focused on analyzing a natural algorithm for the problem, resulting in a constant approximation factor worse than 2 [18]. In this article, we close the gap between the Steiner forest problem and its generalized version, PCSF, by presenting a 2-approximation algorithm for PCSF.

The Steiner tree problem is a well-studied special case of the Steiner forest problem. In the Steiner tree problem, one endpoint of every demand is a specific vertex known as *root*. In contrast to the Steiner forest problem, the approximation factor of the Steiner tree problem has seen significant progress since the introduction of the $(2 - \frac{2}{n})$-approximation algorithm by Goemans and Williamson [16]. Several improvements have been made [24, 26, 28], leading to a 1.39 approximation factor achieved by Byrka et al. [12]. Lower bounds have also been established, with Karp [23] proving the NP-hardness of the Steiner tree problem and consequently the Steiner forest problem, and Bern and Plassman [8] and Chlebík and Chlebíková [13] demonstrating that achieving an approximation factor within 96/95 is NP-hard. These advancements, along with the established lower bounds, underscore the extensive research conducted in the field of Steiner tree and Steiner forest problems.

Regarding the previous works in the prize-collecting version of these problems, Goemans and Williamson [16] provided a $(2 - \frac{1}{n-1})$-approximation algorithm for **prize-collecting Steiner tree (PCST)** and the **prize-collecting Travelling Salesman problem (PCTSP)** in addition to their work on the Steiner forest problem. However, they did not provide an algorithm specifically for the PCSF problem, leaving it as an open problem. Later, Hajiaghayi and Jain [20] in 2006 proposed a deterministic primal-dual $(3-\frac{2}{n})$-approximation algorithm for the PCSF problem, which inspired our work. They also presented a randomized LP-rounding 2.54-approximation algorithm for the

[1]Indeed, Goemans and Williamson [22] (Section 4.6.1) explicitly mention that "the primal-dual algorithm we have presented simulates an algorithm of Agrawal, Klein, and Ravi [AKR95]. Their algorithm was the first approximation algorithm for this [Steiner forest a.k.a. generalized Steiner tree] problem and has motivated much of the authors' research in this area." The seminal work of Agrawal et al. [1, 2] recently received the 30-year STOC Test of Time Award.

problem. In their paper, they mentioned that finding a better approximation factor, ideally 2, remained an open problem. However, no improvements have been made to their result thus far. Furthermore, other 3-approximation algorithms have been proposed using cost sharing [17] or iterative rounding [19] (see, e.g., other works [7, 21, 27] for further studies on PCSF and its generalizations). To the best of our knowledge, our article is the first work that improves the approximation factor of Hajiaghayi and Jain [20].

Moreover, advancements have been made in the PCST problem since the initial $(2 - \frac{1}{n-1})$-approximation algorithm by Goemans and Williamson [16]. The barrier of a 2-approximation factor was broken by Archer et al. [5], and subsequently, Ahmadi et al. [4] further improved upon this problem, presenting a 1.799-approximation algorithm, which currently holds the best approximation factor for this problem. Additionally, there have been significant advancements in prize-collecting TSP, whose natural LP relaxation shares similarities with the natural LP relaxation of PCST. Various works have been done in this area [5, 11, 14], and the currently best-known approximation factor is 1.599 [10]. These works demonstrate the importance and interest surrounding prize-collecting problems, emphasizing their significance in the research community.

For a while, the best-known lower bound for the integrality gap of the natural LP relaxation for PCSF was 2. However, Könemann et al. [25] proved that the integrality gap of this LP is at least 9/4. This result suggests that it is not possible to achieve a 2-approximation algorithm for PCSF solely through primal-dual approaches based on the natural LP, similar to the approaches presented in other works [19, 20]. This raises doubts about the possibility of achieving an algorithm with an approximation factor better than 9/4.

However, in this article, we provide a positive answer to this question. Our main result, Theorem 1.1, demonstrates the existence of a natural deterministic algorithm for the PCSF problem that achieves a 2-approximate solution in polynomial time.

Theorem 1.1. *There exists a deterministic algorithm for PCSF that achieves a 2-approximate solution in polynomial time.*

We address the 9/4 integrality gap by analyzing a natural iterative algorithm. In contrast to previous approaches in the Steiner forest and PCSF fields that compare solutions with feasible dual LP solutions, we compare our solution directly with the optimal solution and assess how much the optimal solution surpasses the dual. While our algorithm incorporates a primal-dual algorithm as a subroutine, relying solely on LP techniques would be insufficient to overcome the integrality gap.

In addition, we analyze a general approach that can be applied to various prize-collecting problems. In any prize-collecting problem, an algorithm needs to decide which demands to pay penalties for and which demands to satisfy. Let us assume that for a prize-collecting problem, we have a base algorithm A. We propose a natural iterative algorithm that begins by running A on an initial instance and storing its solution as one of the options for the final solution. The solution generated by algorithm A pays penalties for some demands and satisfies others. Subsequently, we assume that all subsequent solutions generated by our algorithm will pay penalties for the demands that A paid, set the penalties of these demands to zero, and run A again on the modified instance. We repeat this procedure recursively until we reach a state where algorithm A satisfies every demand with a non-zero penalty, meaning that further iterations will yield the same solution. This state is guaranteed to be reached since the number of non-zero demands decreases at each step. Finally, we select the solution with the minimum cost among the multiple solutions obtained for the initial instance. This natural iterative algorithm could be effective in solving prize-collecting problems. For example, since the appearance of the conference version of this article, the approximation factor for PCST has improved from 1.96 to 1.79 by utilizing this iterative method [4]. In this article,

we analyze its application to the PCSF problem using a variation of the algorithm proposed in the work of Hajiaghayi and Jain [20] as our base algorithm.

One interesting aspect of our findings is that the current best algorithm for the Steiner forest problem achieves an approximation ratio of 2, and this approximation factor has remained unchanged for a significant period of time. It is worth noting that the Steiner forest problem is a specific case of PCSF, where each instance of the Steiner forest can be transformed into a PCSF instance by assigning a sufficiently large penalty to each demand. Since our result achieves the same approximation factor for PCSF, improving the approximation factor for the PCSF problem proves to be more challenging compared to the Steiner forest problem. In future research, it may be more practical to focus on finding a better approximation factor for the Steiner forest problem, which has been an open question for a significant duration. Additionally, investigating the tightness of the 2-approximation factor for both problems could be a valuable direction for further exploration.

1.1 Primal-Dual LP for Prize-Collecting Steiner Forest

Before explaining our algorithm, we discuss the natural LP of PCSF and its dual. We also provide a brief explanation of the algorithm by Hajiaghayi and Jain [20] , with a detailed representation deferred to Section 2. Note that we introduce a modification to their algorithm, which is essential for the analysis of our 2-approximation algorithm.

The natural LP for PCSF is presented next.

$$\begin{aligned}
\text{Minimize} \quad & \sum_{e\in E} c_e x_e + \sum_{(i,j)\in V\times V} \pi_{ij} z_{ij} && \\
\text{Subject to} \quad & \sum_{e\in\delta(S)} x_e + z_{ij} \geq 1 && \forall S \subseteq V, (i,j) \in V\times V, S \odot (i,j) \\
& x_e \geq 0 && \forall e \in E \\
& z_{ij} \geq 0 && \forall (i,j) \in V\times V
\end{aligned}$$

Here, x_e and z_{ij} are relaxations of integral variables, where x_e indicates the inclusion of edge e in the solution, and z_{ij} represents whether vertices i and j are not connected in the solution. Additionally, $\delta(S)$ denotes the set of edges with exactly one endpoint in S, and $S \odot (i,j)$ denotes that $|S \cap \{i,j\}| = 1$.

The dual of the preceding LP is given next, with y_{Sij} as the dual variable, which we refer to as the assignment variable in the subsequent parts of the article.

$$\begin{aligned}
\text{Maximize} \quad & \sum_{\substack{S\subseteq V \\ S\odot(i,j)}} y_{Sij} && \\
\text{Subject to} \quad & \sum_{\substack{S:e\in\delta(S) \\ S\odot(i,j)}} y_{Sij} \leq c_e && \forall e \in E \\
& \sum_{\substack{S\subseteq V \\ S\odot(i,j)}} y_{Sij} \leq \pi_{ij} && \forall (i,j) \in V\times V \\
& y_{Sij} \geq 0 && \forall S \subseteq V, (i,j) \in V\times V
\end{aligned}$$

Hajiaghayi and Jain [20] introduced a modified version of this dual to enable applying the 2-approximation algorithm for Steiner forest by Goemans and Williamson [16]. This modified version is as follows.

$$
\begin{aligned}
\text{Maximize} \quad & \sum_{S \subseteq V} y_S \\
\text{Subject to} \quad & y_S = \sum_{\substack{(i,j) \in V \times V \\ S \odot (i,j)}} y_{Sij} && \forall S \subseteq V \\
& \sum_{S : e \in \delta(S)} y_S \leq c_e && \forall e \in E \\
& \sum_{\substack{S \subseteq V \\ S \odot (i,j)}} y_{Sij} \leq \pi_{ij} && \forall (i,j) \in V \times V \\
& y_{Sij} \geq 0 && \forall S \subseteq V, (i,j) \in V \times V \\
& y_S \geq 0 && \forall S \subseteq V
\end{aligned}
$$

From now on, we refer to this version as the dual LP for PCSF. Next, we outline the 3-approximation algorithm of Hajiaghayi and Jain [20] based on this dual. The complete details of this algorithm are provided in Section 2.

A 3-Approximation Algorithm for Prize-Collecting Steiner Forest. The algorithm of Hajiaghayi and Jain [20] constructs a forest while maintaining a valid instance of the dual LP. It ensures that the forest's cost is at most twice the dual LP objective, with penalties not exceeding this value. The algorithm tracks only the variables y_S for sets, where a configuration is valid if there exists an assignment to y_{Sij} satisfying all dual LP constraints. Initially, all y_S are set to zero.

Starting with an empty forest, the algorithm iteratively builds the final forest. A connected component is an active set if it must be extended to connect with other components and satisfy demands it cuts (i.e., demands with one endpoint in the component).

In each iteration, the dual variables y_S of active sets are increased uniformly until a dual LP constraint becomes tight. If an edge constraint $\sum y_S \leq c_e$ becomes tight, the corresponding edge e is added to the forest, merging the components of its endpoints. If a pair constraint $\sum y_{Sij} \leq \pi_{ij}$ becomes tight, the algorithm deactivates all active sets where further increasing to their y_S would violate constraints. Any pairs cut by these deactivated sets are labeled as tight pairs because their dual LP constraints are tight, and their penalties are paid.

This process continues until no active sets remain. At the end of the algorithm, all tight pairs are identified, and their penalties are paid. Note that the tightness of a pair depends not only on y_S but also on how the values are assigned to y_{Sij}. While Hajiaghayi and Jain [20] do not require assigning values to y_{Sij}, we add a step to carefully assign these values to minimize the number of tight pairs.

Finally, the algorithm removes edges that are not part of paths connecting non-tight pairs and returns the resulting forest along with the set of tight pairs for which penalties are paid. Although this approach may result in more penalties than those in the work of Hajiaghayi and Jain [20], it maintains the same approximation factor and enables the development of a 2-approximation algorithm.

To analyze the 3-approximation algorithm, note that the algorithm always maintains a valid dual LP solution. Thus, the sum of y_S provides a lower bound for the primal LP and the optimal solution.

Bounding the cost of the final forest follows the approach of Goemans and Williamson [16] for the Steiner forest problem, showing each increase in y_S raises the forest cost by at most twice the corresponding amount. Additionally, penalties are bounded by the sum of y_S, as pairs with tight constraints satisfy $\sum y_{Sij} = \pi_{ij}$. Since each y_{Sij} contributes to one pair constraint, the total penalty cannot exceed the sum of y_S. This ensures a 3-approximation solution.

1.2 Algorithm and Techniques

In Section 3, we present an iterative algorithm and prove its 2-approximation guarantee for PCSF. Here, we provide a brief explanation of how our algorithm works.

Let us refer to our 3-approximation algorithm as PCSF3. Our goal is to construct a 2-approximation algorithm called *IPCSF*, by iteratively invoking PCSF3. In IPCSF, we first invoke PCSF3 and obtain a feasible solution (Q_1, F_1'), where Q_1 represents the pairs for which we pay their penalty, and F_1' is a forest that connects the remaining pairs. Next, we set the penalty for each pair in Q_1 to 0. We recursively call IPCSF with the updated penalties. Intuitively, this can be beneficial because setting the penalties for certain pairs to 0 results in a smaller increase in the dual variables. Consequently, this encourages other pairs to avoid connecting with each other and not rely on the penalties associated with pairs in Q_1, which can help avoid connecting pairs the optimal solution would not connect.

Let us assume that (Q_2, F_2') is the result of this recursive call to IPCSF for the updated penalties. It is important to note that (Q_2, F_2') is a feasible solution for the initial instance, as it either connects the endpoints of each pair or places them in Q_2. Furthermore, it is true that $Q_1 \subseteq Q_2$, as the penalty of pairs in Q_1 is updated to 0, and they will be considered as tight pairs in further iterations of PCSF3. By induction, we assume that (Q_2, F_2') is a 2-approximation of the optimal solution for the instance with the updated penalties. Now, we want to show that either (Q_1, F_1') or (Q_2, F_2') is a 2-approximation of the optimal solution for the initial instance. We will select the one with the lower cost and return it as the output of the algorithm.

To analyze the algorithm, we focus on pairs in Q_1 that are connected in a fixed optimal solution and the value of dual variables achieved by PCSF3. Let $\mathcal{CP}$ denote the set of pairs $(i, j) \in Q_1$ that are connected in the optimal solution. We concentrate on this set because the optimal solution connects these pairs, and we will pay their penalties in both (Q_1, F_1') and (Q_2, F_2'). Let us assume cp represents the sum of y_{Sij} for pairs in $\mathcal{CP}$ in the dual solution determined by PCSF3. For any pair $(i, j) \in \mathcal{CP}$ and set S such that $y_{Sij} > 0$, since the pair (i, j) is connected in the optimal solution, we know that S cuts at least one edge of the optimal solution. Let cp_1 be the total value of y_{Sij} for pairs $(i, j) \in \mathcal{CP}$ where S cuts exactly one edge of the optimal solution, and cp_2 be the total value of y_{Sij} for pairs $(i, j) \in \mathcal{CP}$ where S cuts at least two edges. It follows that $cp_1 + cp_2 = cp$.

We now consider the values of cp_1 and cp_2 to analyze the algorithm. If cp_2 is sufficiently large, we can establish a stronger lower bound for the optimal solution compared to our previous bound, which was $\sum y_S$. The previous bound can be explained as follows: for each pair (i, j), the optimal solution either pays its penalty, which is at least the sum of y_{Sij}, or connects it. For each set S where $y_{Sij} > 0$ and the pair (i, j) is connected by the optimal solution, S cuts at least one edge in the optimal solution. This implies that the total cost of the forest in the optimal solution is at least the sum of y_{Sij} for all pairs (i, j) connected by S. Given that cp_2 is sufficiently large compared to cp_1, this indicates that a significant portion of the sets S cut at least two edges of the optimal solution. Consequently, this leads to a better bound for the optimal solution. This improved lower bound allows us to conclude that the output of PCSF3, (Q_1, F_1'), becomes a 2-approximate solution.

Alternatively, if cp_1 is significantly large, we can show that the optimal solution for the updated penalties is substantially smaller than the optimal solution for the initial instance. This is achieved by removing the edges from the initial optimal solution that are cut by sets like S which cut exactly one edge of the optimal solution and have $y_{Sij} > 0$ for some pairs $(i, j) \in \mathcal{CP}$. By minimizing the number of tight pairs at the end of PCSF3, we ensure that no pair without a tight constraint in the dual instance of PCSF3 is cut by any of these sets, and removing these edges will not disconnect those pairs. Consequently, we can construct a feasible solution for the updated penalties without utilizing any edges from the cutting edges of these sets in the initial optimal solution. In summary,

since (Q_2, F_2') is a 2-approximation of the optimal solution for the updated penalties, and the optimal solution for the updated penalties has a significantly lower cost than the optimal solution for the initial instance, (Q_2, F_2') becomes a 2-approximation of the optimal solution for the initial input.

Last but not least, we conduct further analysis of our algorithm to achieve a more refined approximation factor of $2 - \frac{1}{n}$, which asymptotically approaches 2.

1.3 Preliminaries

For a given set $S \subset V$, we define the set of edges that have exactly one endpoint in S as the *cutting edges* of S, denoted by $\delta(S)$. In other words, $\delta(S) = \{(u, v) \in E : |\{u, v\} \cap S| = 1\}$. We say that S cuts an edge e if e is a cutting edge of S—that is, $e \in \delta(S)$. We say that S cuts a forest F if there exists an edge $e \in F$ such that S cuts that edge.

For a given set $S \subset V$ and pair $\{i, j\} \in V \times V$, we say that S cuts (i, j) if and only if $|\{i, j\} \cap S| = 1$. We denote this relationship as $S \odot (i, j)$.

For a forest F, we define $c(F)$ as the total cost of edges in F—that is, $c(F) = \sum_{e \in F} c_e$.

For a set of pairs of vertices $Q \subseteq V \times V$, we define $\pi(Q)$ as the sum of penalties of pairs in Q—that is, $\pi(Q) = \sum_{(i,j) \in Q} \pi_{ij}$.

For a given solution SOL to a PCSF instance I, the notation $cost(\text{SOL})$ is used to represent the total cost of the solution. In particular, if SOL uses a forest F and pays the penalties for a set of pairs Q, then the total cost is given by $cost(\text{SOL}) = c(F) + \pi(Q)$.

For a graph $G = (V, E)$ and a vertex $v \in V$, we define $d_G(v)$ as the degree of v in G. Similarly, for a set $S \subset V$, we define $d_G(S)$ as the number of edges that S cuts—that is, $|E \cap \delta(S)|$.

Lemma 1.2. *Let F be an arbitrary forest and S be a subset of vertices in F. If S cuts only one edge e in F, then removing this edge will only disconnect pairs of vertices cut by S.*

Proof. Consider a pair (i, j) that is disconnected by removing e. This pair must be connected in forest F, so there is a unique simple path between i and j in F. This path must include edge e, as otherwise the pair would remain connected after removing e. Let the endpoints of e be u and v, where $u \in S$ and $v \notin S$. Without loss of generality, assume that i is the endpoint of the path that is closer to u than v. Then i is connected to u through the edges in the path other than e. As these edges are not cut by S and $u \in S$, it follows that i must also be in S. Similarly, it can be shown that j is not in S. Therefore, S cuts the pair (i, j). □

2 Representing a 3-Approximation Algorithm

In this section, we present the primal-dual algorithm of Hajiaghayi and Jain [20], which achieves a 3-approximation solution, with slight modifications in selecting the pairs for paying the penalty. Specifically, we pay for a superset of pairs compared to their approach. We also provide useful properties of this algorithm in Corollaries 2.1, 2.2, and 2.9, along with Lemmas 2.15 and 2.16. These properties are proved here and will be used in the analysis of our 2-approximation algorithm in the next section. The pseudocode for this method is detailed in Algorithm 1. Additionally, the algorithm can be understood through a coloring schema for better intuition, explained in the conference version of this article [3].

The algorithm is based on the dual LP formulation described in Section 1.1. The dual LP introduces two types of dual variables: the set variables y_S and the assignment variables y_{Sij}. We define the pair variables as $y_{ij} = \sum_{S \odot (i,j)} y_{Sij}$. Before describing the algorithm, we need to define edge constraints and tightness, as well as pair constraints and tightness based on the dual LP.

Definition 2.1 (Edge Constraint and Tightness). In the dual LP, the constraint for each edge $e \in E$ is given by $\sum_{S:e\in\delta(S)} y_S \leq c_e$, referred to as the *edge constraint*. If this constraint is tight for an edge e (i.e., $\sum_{S:e\in\delta(S)} y_S = c_e$), we say that edge e is a tight edge.

Definition 2.2 (Pair Constraint and Tightness). In the dual LP, the constraint for each pair $(i, j) \in V \times V$ is given by $y_{ij} = \sum_{S\odot(i,j)} y_{Sij} \leq \pi_{ij}$, referred to as the *pair constraint*. If this constraint is tight for a pair (i, j) (i.e., $y_{ij} = \pi_{ij}$), we say that pair (i, j) is a tight pair.

The algorithm increases the set variables while ensuring that the *edge constraint* is always satisfied. The values of the set variables y_S are divided and assigned to the assignment variables y_{Sij} such that $y_S = \sum_{S\odot(i,j)} y_{Sij}$, and the pair variables are then computed based on these assignment variables. Additionally, we ensure that the *pair constraint* is never violated by carefully observing the pair variables.

We introduce some variables used in Algorithm 1. Let F be an initially empty forest that we will build upon to eventually form our final forest. We maintain the set of connected components of F in FC and a subset of these connected components in $ActS$, which includes the active sets whose dual variables y_S are increasing. Initially, $ActS$ is set to FC.

As the algorithm progresses, the dual variable of active sets increases uniformly. This increase affects the *edge constraint* of edges between connected components, as y_S influences the *edge constraint* for edges cut by S. The process continues until an edge connecting different connected components becomes tight. When this happens, the edge is added to F, and FC is updated to reflect the new connected components. Since the edge connects two distinct components, F remains a forest. We then update $ActS$ by merging the sets containing the edge's endpoints. This ensures that the *edge constraint* for the edge remains satisfied as it will not be adjacent to an active set. This leads to the following corollary.

Corollary 2.1. *For any edge $e \in E$, the edge constraint will never be violated, which means*

$$\sum_{S:e\in\delta(S)} y_S \leq c_e.$$

We will later explain the procedure FindDeltaE, which identifies the first moment when a new edge becomes tight. Although our algorithm ensures that *edge constraint* is maintained, we must also address *pair constraint*. First, we define the validity of set variables based on *pair constraint*, assuming *edge constraint* is already handled.

Definition 2.3 (Valid Set Variables). The values of set variables are considered valid if there exists a set of values for assignment variables and consequently for pair variables such that the following conditions hold:

- For every set $S \subseteq V$, the value of the set variable for S can be distributed among pairs (i, j) satisfying $S \odot (i, j)$ such that $\sum_{(i,j):S\odot(i,j)} y_{Sij} = y_S$.
- For every pair $(i, j) \in V \times V$, the pair constraint is not violated—that is, $y_{ij} = \sum_{S:S\odot(i,j)} y_{Sij} \leq \pi_{ij}$.

If no such set of values satisfies these conditions, the set variables are considered invalid.

Using the validity of set variables, we define set tightness.

Definition 2.4 (Set Tightness). A set $S \subseteq V$ is defined as tight if increasing the value of y_S by any $\epsilon > 0$ without changing the y_S of other sets would make the set variables invalid.

The procedure FindDeltaP determines the first moment, starting from the current moment, when an active set becomes tight. This means we can increase the set variable of all active sets by

Δ_p without violating any pair constraints, as the increase would not make the set variables invalid, and otherwise we would have already had a tight set. We also use the procedure CheckSetIsTight to detect when active sets become tight and deactivate them to ensure that the *pair constraint* is not violated. This leads to the following corollary.

Corollary 2.2. *For any pair $(i, j) \in V \times V$, the pair constraint will never be violated, which means*

$$y_{ij} = \sum_{\substack{S \subseteq V \\ S \odot (i,j)}} y_{Sij} \leq \pi_{ij}.$$

To explain how Algorithm 1 operates, we begin by describing its key steps. Initially, the function FindDeltaE, called at Line 5, computes the maximum duration Δ_e for which the dual variables y_S of active sets can be increased without violating any *edge constraint*. Similarly, at Line 6, the algorithm uses FindDeltaP to determine the maximum increment Δ_p that ensures no *pair constraint* will be violated by increasing y_S for active sets by Δ_p. These procedures are essential for executing the algorithm in discrete steps.

In the next phase, the algorithm advances by $\min(\Delta_e, \Delta_p)$ at Line 9, resulting in at least one new edge becoming tight or one new tight set. During the loop at Line 11, newly tight edges are identified, and their endpoints' sets are merged. Similarly, within the loop at Line 19, the algorithm identifies and deactivates tight sets.

The process continues until no active sets remain. Finally, at Line 25, the function FindMinimalAssignment is called to find values for pair variables such that the number of tight pairs is minimal. This function is critical for ensuring that the final assignment of values to assignment and pair variables meets the requirements of the analysis in Section 3. The set of tight pairs returned by this function, denoted by Q, is the set of pairs for which penalties need to be paid. We derive our final forest F' from F by removing redundant edges that are not necessary for connecting demands in $(V \times V) \setminus Q$.

Before we explain the procedures FindDeltaE, FindDeltaP, CheckSetIsTight, and FindMinimalAssignment in more detail, we introduce the following lemma, which is essential in the analysis of these functions.

Lemma 2.3. *In the PCSF3 algorithm, the number of sets that have been active at some point during its execution is linear.*

Proof. During the algorithm, new active sets are only created in Line 16 by merging existing sets. Initially, we start with n active sets in $ActS$. Symmetrically, for each creation of a new active set, we have one merge operation over sets in FC, which reduces the number of sets in FC by exactly 1. Since we start with n sets in FC, the maximum number of merge operations is $n - 1$. Therefore, the total number of active sets throughout the algorithm is at most $2n - 1$. □

Finding the Maximum Value for Δ_e. In FindDeltaE, we determine the maximum value of Δ_e such that increasing the set variables of active sets by Δ_e does not violate *edge constraint* for any edge. We consider each edge $e = (u, v)$ where u and v are not in the same connected component, and at least one of them belongs to an active set. An edge e is tight when the sum of the dual variables of the sets that cut e equals c_e. Consequently, the remaining capacity for increasing the dual variables before the edge becomes tight is $c_e - \sum_{S:e \in \delta(S)} y_S$. Then, edge e becomes tight if we increase the dual variables of all active sets by $\frac{c_e - \sum_{S:e \in \delta(S)} y_S}{t}$, where t is the number of endpoints of e that are in active sets. To ensure that the *edge constraint* is not violated, we select Δ_e as the minimum increment required for an edge to become tight among all edges. This selection leads to the following corollary.

ALGORITHM 1: PCSF3($I = (G, \pi)$): A 3-Approximation Algorithm

Input: An undirected graph $G = (V, E, c)$ with edge costs $c : E \to \mathbb{R}_{\geq 0}$ and penalties $\pi : V \times V \to \mathbb{R}_{\geq 0}$.
Output: A set of pairs Q with a forest F' that connects the endpoints of every pair $(i, j) \notin Q$.

Initialize $F \leftarrow \emptyset$
Initialize $ActS, FC \leftarrow \{\{v\} : v \in V\}$
Implicitly set $y_S \leftarrow 0$ for all $S \subset V$
while $ActS \neq \emptyset$ **do**
  $\Delta_e \leftarrow$ FindDeltaE$(G, y, ActS, FC)$
  $\Delta_p \leftarrow$ FindDeltaP$(G, \pi, y, ActS)$
  $\Delta \leftarrow \min(\Delta_e, \Delta_p)$
  for $S \in ActS$ **do**
    $y_S \leftarrow y_S + \Delta$
  end for
  for $e \in E$ **do**
    Let $S_v, S_u \in FC$ be sets that contains each endpoint of e
    if $\sum_{S: e \in \delta(S)} y_S = c_e$ **and** $S_v \neq S_u$ **then**
      $F \leftarrow F \cup \{e\}$
      $FC \leftarrow (FC \setminus \{S_p, S_q\}) \cup \{S_p \cup S_q\}$
      $ActS \leftarrow (ActS \setminus \{S_p, S_q\}) \cup \{S_p \cup S_q\}$
    end if
  end for
  for $S \in ActS$ **do**
    if CheckSetIsTight(G, π, y, S) **then**
      $ActS \leftarrow ActS \setminus \{S\}$
    end if
  end for
end while
$Q \leftarrow$ FindMinimalAssignment(G, π, y)
Let F' be the subset of F obtained by removing unnecessary edges for connecting demands $(V \times V) \setminus Q$.
return (Q, F')

Corollary 2.4. *After increasing the values of y_S by Δ_e for active sets, at least one new edge becomes tight.*

Note that for the calculation of Δ_e for an edge, we assume the current active sets do not change during this time.

It is clear that the operations for FindDeltaE can be performed in polynomial time, leading to the following corollary.

Corollary 2.5. *The runtime of FindDeltaE is polynomial.*

Finding the Maximum Value for Δ_p. We employ an LP to determine the maximum value for Δ_p. Recall that here we want to increase the values of y_S as much as possible without violating any *pair constraint* and consequently ensuring the validity of set variables (Definition 2.3). The following LP finds such a value for Δ_p.

$$\begin{aligned} &\text{Maximize} && \Delta_p && \\ &\text{Subject to} && y_S + \Delta_p = \sum_{\substack{(i,j) \in V \times V \\ S \odot (i,j)}} y_{Sij} && \forall S \in ActS \end{aligned}$$

$$
\begin{aligned}
& y_S = \sum_{\substack{(i,j)\in V\times V \\ S \odot (i,j)}} y_{Sij} && \forall S \subseteq V \\
& \sum_{\substack{S\subseteq V \\ S \odot (i,j)}} y_{Sij} \leq \pi_{ij} && \forall (i,j) \in V \times V \\
& y_{Sij} \geq 0 && \forall S \subseteq V, (i,j) \in V \times V \\
& \Delta_p \geq 0
\end{aligned}
$$

In the preceding LP, Δ_p and y_{Sij} are variables, and y_S are constants. It is important to note that the second and fourth constraints can be ignored for any set S with $y_S = 0$, which is not active since $y_{Sij} = 0$ for such sets and any pair (i, j). Then, by Lemma 2.3, only a polynomial number of sets are considered in the constraints, and the LP has polynomial size. This leads to the following corollary.

Corollary 2.6. *Throughout the algorithm, each call to the FindDeltaP function executes in polynomial time.*

Based on the constraints of the preceding LP, the set variables remain valid after increasing the set variables of active sets by the value of Δ_p. Now, we show that after this increase, at least one active set becomes tight.

Lemma 2.7. *After increasing the value of y_S by Δ_p for active sets, at least one active set becomes tight.*

Proof. We use proof by contradiction. Assume that no active set becomes tight. From Definition 2.4, there exists an $\epsilon > 0$ such that for any $A \in ActS$, increasing y_A by ϵ independently does not violate the validity of set variables. Now, increasing y_A by $\frac{\epsilon}{|ActS|}$ for all $A \in ActS$ would not violate the validity of the set variables, as we can determine feasible values for the assignment variables by averaging the values of the assignment variables that we obtain by increasing independently. This contradicts the maximality of Δ_p, which is the objective of the LP. □

Check If a Set Is Tight. To determine if a set is tight, we solve a similar LP for an active set $A \in ActS$. The LP is formulated as follows.

$$
\begin{aligned}
\text{Maximize} \quad & \epsilon \\
\text{Subject to} \quad & y_S + \epsilon = \sum_{\substack{(i,j)\in V\times V \\ S \odot (i,j)}} y_{Sij} && S = A \\
& y_S = \sum_{\substack{(i,j)\in V\times V \\ S \odot (i,j)}} y_{Sij} && \forall S \subseteq V \\
& \sum_{\substack{S\subseteq V \\ S \odot (i,j)}} y_{Sij} \leq \pi_{ij} && \forall (i,j) \in V \times V \\
& y_{Sij} \geq 0 && \forall S \subseteq V, (i,j) \in V \times V \\
& \epsilon \geq 0
\end{aligned}
$$

In the LP described, ϵ and y_{Sij} are variables, whereas y_S are given constants. If $\epsilon > 0$, then based on Definition 2.4, the set A is not tight. Conversely, if $\epsilon = 0$, the set A is tight. Using similar arguments as for the FindDeltaP function, this LP is solvable in polynomial time.

Corollary 2.8. *Each call to the* CheckSetIsTight *function during the algorithm runs in polynomial time.*

Finding the Final Assignment with Minimal Tight Pairs. In the final step of the algorithm, we seek a valid assignment for the dual variables y_{Sij} that is consistent with the values y_S, aiming to minimize the number of tight pairs. We first find a feasible solution for the following system of linear constraints.

$$y_S = \sum_{\substack{(i,j)\in V\times V \\ S\odot(i,j)}} y_{Sij} \qquad \forall S \subseteq V$$

$$\sum_{\substack{S\subseteq V \\ S\odot(i,j)}} y_{Sij} \le \pi_{ij} \qquad \forall (i,j) \in V \times V$$

$$y_{Sij} \ge 0 \qquad \forall S \subseteq V, (i,j) \in V \times V$$

Note that the y_S values in the constraints are not variables and correspond to the values obtained using the primal-dual process. Additionally, for any set S with $y_S = 0$, we can safely assume that all y_{Sij} values are also equal to 0. Therefore, we only care about sets with $y_S > 0$, of which there is only polynomially many by Lemma 2.3. To find a feasible solution, we can view the system as an LP with an arbitrary bounded objective function and solve the LP in polynomial time.

We say that an assignment has minimal tight pairs if for any tight pair (i, j) and any set S with $y_{Sij} > 0$, any other pair (i', j') cut by S is also tight. Algorithm 2 provides a pseudocode for procedure FindMinimalAssignment which obtains an assignment with minimal tight pairs. We explain how this algorithm adjusts the y_{Sij} values to satisfy this condition.

Assume there exists a pair (i, j) along with set S and pair (i', j') that violates the condition. Since (i', j') is not tight and $y_{Sij} > 0$, there must exist some $\epsilon > 0$ such that $y_{i'j'} + \epsilon < \pi_{i'j'}$ and $y_{Sij} > \epsilon$. Then, consider a new assignment with ϵ deducted from y_{Sij} and added to $y_{Si'j'}$. This reduces y_{ij} by $\epsilon > 0$, so the pair (i, j) will not be tight anymore. In addition, the *pair constraint* for (i', j') will not be violated and (i', j') will not become tight as $y_{i'j'}$ will remain strictly less than $\pi_{i'j'}$. Last, y_S remains equal to the sum of y_{Sij} values as the net change in these variables is zero. So, we obtain a new valid assignment with one fewer tight pair. We can repeat this process while there exists a tight pair violating the minimality condition. Finally, we will be left with a minimal assignment for the variables where the following lemma stands.

Corollary 2.9. *In the final assignment obtained in* FindMinimalAssignment*, if a set $S \subseteq V$ cuts a tight pair (i, j) with $y_{Sij} > 0$, then all pairs (i', j') satisfying $S \odot (i', j')$ are also tight.*

It can also be seen that Algorithm 2 can be implemented in polynomial time. First, as discussed earlier, finding the initial assignment reduces to solving an LP with a polynomial number of variables and constraints. Additionally, the while loop can only be executed a polynomial number of times, as each iteration reduces the number of tight pairs by 1. Last, as the number of sets S with $y_S > 0$ is polynomial by Lemma 2.3, finding the while condition, as well as the violating pairs and sets, can be done in polynomial time.

Corollary 2.10. *The* FindMinimalAssignment *function runs in polynomial time.*

2.1 Analysis

In this section, we analyze some properties of the PCSF3 algorithm. We will show, in the next two lemmas, that any inactive set remains tight and that every component of the final forest F will be a tight set given the final y_S values.

ALGORITHM 2: FindMinimalAssignment(G, π, y): Obtaining Minimal Set of Tight Pairs along with Final Assignment

Input: An undirected graph $G = (V, E, c)$ with edge costs $c : E \to \mathbb{R}_{\geq 0}$ and penalties $\pi : V \times V \to \mathbb{R}_{\geq 0}$ and set variables y_S
Output: The set Q of tight pairs for which we will pay penalties.

```
Find a valid assignment for dual variables y_{Sij} consistent with y_S values
Calculate y_{ij} values as y_{ij} = ∑_{S:S⊙(i,j)} y_{Sij}
while ∃(i,j),(i',j') ∈ V×V and S ⊂ V that S⊙(i,j), S⊙(i',j'), y_{ij} = π_{ij}, y_{i'j'} < π_{i'j'}, and y_{S'ij} > 0 do
    ε ← min(y_{Sij}, π_{i'j'} − y_{i'j'})/2
    y_{Sij} ← y_{Sij} − ε
    y_{ij} ← y_{ij} − ε
    y_{Si'j'} ← y_{Si'j'} + ε
    y_{i'j'} ← y_{i'j'} + ε
end while
Let Q ← {(i,j) ∈ V×V : ∑_{S:S⊙(i,j)} y_{Sij} = π_{ij}}
return Q
```

Lemma 2.11. *Once a set S becomes tight, it remains tight throughout the algorithm.*

Proof. By Definition 2.4, a set S is considered tight if, for any $\epsilon > 0$, the values of set variables will not be valid if y_S is increased by ϵ. Additionally, as seen in Definition 2.3, the constraints for checking the validity of set variables only impose upper bounds on y_S values. Therefore, if a valuation of set variables is not valid, increasing the y_S values will not lead to a valid valuation either. As y_S values only increase throughout the algorithm, any set that is tight will continue to be tight. □

Lemma 2.12. *At the end of PCSF3, all remaining sets in FC are tight.*

Proof. In Line 2 of the algorithm, both *ActS* and *FC* are initialized with the same set of sets. Additionally, in Lines 15 and 16, the same sets are removed from *ActS* and *FC* or added to both data structures. The only difference occurs in Line 21, where tight sets are removed from *ActS* but not from *FC*. These removed sets will remain tight by Lemma 2.11. Therefore, at the end of the algorithm, since there are no sets remaining in *ActS*, all sets in *FC* are tight. □

Now we show that if a set is tight, all pairs with one endpoint in that set are tight.

Lemma 2.13. *If a set S is tight, then any pair (i, j) such that $S \odot (i, j)$ is also tight.*

Proof. We prove this by contradiction. Assume there is a pair (i, j) such that $S \odot (i, j)$, and this pair is not tight. Let $\epsilon = \pi_{ij} - y_{ij}$.

Consider increasing y_S, y_{Sij}, and y_{ij} by ϵ. After this increase, for each set S', we have $\sum_{(i',j'):S' \odot (i',j')} y_{S'i'j'} = y_{S'}$, and for each pair $y_{i'j'} = \sum_{S':S' \odot (i',j')} y_{S'i'j'} \leq \pi_{i'j'}$. According to Definition 2.3, this adjustment maintains valid set variables.

Since S is tight, it means no valid assignment for the set variables exists if y_S is increased. Therefore, the assumption that (i, j) is not tight contradicts the tightness of S, because the increase by ϵ should not result in a valid assignment.

Hence, if a set S is tight, all pairs (i, j) where $S \odot (i, j)$ must also be tight. □

The next lemma shows that the returned value (Q, F') forms a valid solution to the given instance—that is, any pair not connected by the forest F' is in Q.

Lemma 2.14. *The endpoints of any pair not in Q will be connected in the forest F'.*

Proof. As all tight pairs are in Q, any such pair will not be tight. The forest F' is obtained by removing redundant edges from F, which are edges that are not part of a path between the endpoints of a pair that is not tight. Hence, it suffices to show that every pair that is not tight is connected in F.

For the sake of contradiction, let us assume that there exists a pair (i, j) that is not tight and the endpoints i and j are not connected in F. Consider the set $S \in FC$ at the end of the algorithm that contains i. Since i and j are not connected in F, and S is a connected component of F, it follows that S cuts the pair (i, j). Since the algorithm is finished, S is tight by Lemma 2.12. This contradicts Lemma 2.13 because we have a tight set S such that $S \odot (i, j)$ is not tight. Therefore, our assumption is false, and every pair that is not tight is connected in F. As a result, after executing FindMinimalAssignment, the endpoints of any pair that is not tight will be connected in the forest F'. □

Next, we analyze the cost of the forest returned in the PCSF3 procedure.

Lemma 2.15. *Forest F' has cost*

$$c(F') \leq 2 \sum_{S \subset V} y_S.$$

Proof. We can prove this similarly to the proof presented by Goemans and Williamson [16]. First, we note that the cost of F' can be expressed as follows:

$$\begin{aligned} c(F') &= \sum_{e \in F'} c_e \\ &= \sum_{e \in F'} \sum_{\substack{S \subset V \\ e \in \delta(S)}} y_S && \text{(Edges of } F' \text{ are tight)} \\ &= \sum_{S \subset V} \sum_{\substack{e \in F' \\ e \in \delta(S)}} y_S && \text{(Change the order of summations)} \\ &= \sum_{S \subset V} d_{F'}(S) y_S. \end{aligned}$$

Now, it suffices to prove the equivalent statement:

$$\sum_{S \subset V} d_{F'}(S) y_S \leq 2 \sum_{S \subset V} y_S.$$

To prove this, we compare the increase in each side during each step of PCSF3. Since both sides are initially equal to zero, showing that the increase in the left-hand side is no more than the increase in the right-hand side establishes the desired inequality.

Now, let us consider a specific step of the procedure PCSF3 where the y_S values of the active sets in $ActS$ are increased by Δ. In this step, the increase in the left-hand side can be written as $\sum_{S \in ActS} d_{F'}(S) \cdot \Delta$ while the increase in the right-hand side is $2\Delta \cdot |ActS|$. Therefore, we want to prove that

$$\sum_{S \in ActS} d_{F'}(S) \leq 2 \cdot |ActS|.$$

Consider the graph H formed from F' by contracting each connected component in FC at this step in the algorithm. As the edges of forest F at this step and F' are a subset of the forest F at the end of PCSF3, the graph H should be a forest. If H contains a cycle, it contradicts the fact that F at the end of PCSF3 is a forest.

In forest H, each vertex represents a set $S \in FC$, and the neighboring edges of this vertex are exactly the edges in $\delta(S) \cap F'$. We refer to the vertices representing active sets as active vertices, and the vertices representing inactive sets as inactive vertices. To simplify the analysis, we remove any isolated inactive vertices from H.

Now, let us focus on the inactive vertices in H. Each inactive vertex must have a degree of at least 2 in H. Otherwise, if an inactive vertex v has a degree of 1, consider the only edge in H connected to this vertex. For this edge not to be removed in the final step of Algorithm 1 at Line 26, there must exist a pair outside of Q that would be disconnected after deleting this edge. However, since vertex v is inactive, its corresponding set S becomes tight before this step. According to Lemma 2.11, S will remain tight afterward. As a result, by Lemma 2.13, any pair cut by S will also be tight and will be included in Q. By applying Lemma 1.2, we can conclude that the only pairs disconnected by removing this edge would be the pairs cut by S, which we have shown to be in Q. Therefore, an inactive vertex cannot have a degree of 1, and all inactive vertices in H have a degree of at least 2. Let V_a and V_i represent the sets of active and inactive vertices in H, respectively. We have

$$\begin{aligned}
\sum_{S \in Act S} d_{F'}(S) &= \sum_{v \in V_a} d_H(v) \\
&= \sum_{v \in V_a \cup V_i} d_H(v) - \sum_{v \in V_i} d_H(v) \\
&\leq 2(|V_a| + |V_i|) - \sum_{v \in V_i} d_H(v) && (H \text{ is a forest}) \\
&\leq 2(|V_a| + |V_i|) - 2|V_i| && (d_H(v) \geq 2 \text{ for } v \in V_i) \\
&\leq 2(|V_a|) = 2|Act S|.
\end{aligned}$$

This completes the proof. □

Finally, we prove that the running time of PCSF3 is polynomial.

Lemma 2.16. *The runtime of PCSF3 is polynomial.*

Proof. By Corollary 2.4 and Lemma 2.7, we can conclude that the number of iterations of the while loop is polynomial. In each iteration, Lemma 2.3, in addition to Corollaries 2.5, 2.6, and 2.8, proves that the number of operations is polynomial. Last, FindMinimalAssignment is polynomial due to Corollary 2.10. Therefore, PCSF3 runs in polynomial time. □

3 The Iterative Algorithm

In this section, we present our iterative algorithm which uses the PCSF3 procedure from Algorithm 1 as a building block. We then provide a proof of its 2-approximation guarantee in Section 3.1. Finally, in Section 3.2, we provide a brief overview of a more refined analysis to establish a $(2 - \frac{1}{n})$-approximation for an n vertex input graph.

Our algorithm, described in Algorithm 3, considers two solutions for the given PCSF instance I. The first solution, denoted as (Q_1, F_1'), is obtained by invoking the PCSF3 procedure (Line 1). If the total penalty of this solution, $\pi(Q_1)$, is equal to 0, the algorithm returns it immediately as the solution.

Otherwise, a second solution, denoted as (Q_2, F_2'), is obtained through a recursive call on a simplified instance R. The simplified instance is created by adjusting penalties: penalties are limited to pairs that Algorithm 1 does not pay for, and the penalties for other pairs are set to 0 (Lines 6–14). Essentially, we assume that penalties paid in the first solution will indeed be paid, and our objective is to find a solution for the remaining pair connection demands. We note that setting the penalties

ALGORITHM 3: IPCSF($I = (G,\ \pi)$): Iterative PCSF Algorithm

Input: An undirected graph $G = (V, E, c)$ with edge costs $c : E \to \mathbb{R}_{\geq 0}$ and penalties $\pi : V \times V \to \mathbb{R}_{\geq 0}$.
Output: A set of pairs Q with a forest F' that connects the endpoints of every pair $(i, j) \notin Q$.

1: $(Q_1, F_1') \leftarrow$ PCSF3(I)
2: **if** $\pi(Q_1) = 0$ **then**
3: **return** (Q_1, F_1')
4: **end if**
5: $cost_1 \leftarrow c(F_1') + \pi(Q_1)$
6: Initialize π' as a new all-zero penalty vector
7: **for** $(i, j) \in V \times V$ **do**
8: **if** $(i, j) \in Q_1$ **then**
9: $\pi'_{ij} \leftarrow 0$
10: **else**
11: $\pi'_{ij} \leftarrow \pi_{ij}$
12: **end if**
13: **end for**
14: Construct instance R of the PCSF problem consisting of G and π'
15: $(Q_2, F_2') \leftarrow$ IPCSF(R)
16: $cost_2 \leftarrow c(F_2') + \pi(Q_2)$
17: **if** $cost_1 \leq cost_2$ **then**
18: **return** (Q_1, F_1')
19: **else**
20: **return** (Q_2, F_2')
21: **end if**

for these pairs to 0 guarantees their inclusion in Q_2. This is because Q_2 represents the set of tight pairs for a subsequent invocation of PCSF3, and any pair with a penalty of 0 is trivially tight.

To compare the two solutions, the algorithm computes the values $cost_1 = c(F_1') + \pi(Q_1)$ and $cost_2 = c(F_2') + \pi(Q_2)$, which represent the costs of the solutions (Lines 5 and 16). In the final step, the algorithm simply selects and returns the solution with the lower cost.

3.1 Analysis

We now analyze the approximation guarantee of Algorithm 3. In the following, we consider an arbitrary instance $I = (G, \pi)$ of the PCSF problem, and analyze the solutions found by the IPCSF algorithm. In our analysis, we focus on *the first call* of IPCSF. By the output of PCSF3, we refer to the result of the first call of PCSF3 on instance I at Line 1. Similarly, when we mention the output of the recursive call, we are referring to the output of IPCSF on instance R at Line 15. We compare the output of IPCSF on I, which is the minimum of the output of PCSF3 and the output of the recursive call, with an optimal solution OPT of the instance I. We denote the forest selected in OPT as F^* and use Q^* to refer to the set of pairs not connected in F^*, for which OPT pays the penalties. Then, the cost of OPT is given by $cost(OPT) = c(F^*) + \pi(Q^*)$. The values y_S, y_{Sij}, and y_{ij} used in the analysis all refer to the corresponding values in the call to PCSF3 on instance I. In particular, the y_S values are the final values of the set variables in the primal-dual algorithm, the y_{Sij} values refer to the final assignment in the call to FindMinimalAssignment, and the y_{ij} values are calculated from y_{Sij} values.

Definition 3.1. For an instance I, we define four sets to categorize the pairs based on their connectivity in both the optimal solution OPT of I and the result of PCSF3(I), denoted as (Q_1, F_1'):

— Set CC contains pairs (i, j) that are connected in the optimal solution and are not in the set Q_1 returned by PCSF3.
— Set $C\mathcal{P}$ contains pairs (i, j) that are connected in the optimal solution and are in the set Q_1 returned by PCSF3.
— Set $\mathcal{P}C$ contains pairs (i, j) that are not connected in the optimal solution and are not in the set Q_1 returned by PCSF3.
— Set $\mathcal{P}\mathcal{P}$ contains pairs (i, j) that are not connected in the optimal solution and are in the set Q_1 returned by PCSF3.

Based on the final dual assignment of PCSF3(I), we define the following values to represent the sum of dual values for each set.

$$cc = \sum_{(i,j)\in CC} y_{ij}, \qquad cp = \sum_{(i,j)\in C\mathcal{P}} y_{ij}$$
$$pc = \sum_{(i,j)\in \mathcal{P}C} y_{ij}, \qquad pp = \sum_{(i,j)\in \mathcal{P}\mathcal{P}} y_{ij}$$

The following table illustrates the connectivity status of pairs in each set.

		PCSF3	
		Connect	Penalty
Optimal Solution	Connect	CC	$C\mathcal{P}$
	Penalty	$\mathcal{P}C$	$\mathcal{P}\mathcal{P}$

Next, we categorize sets that cut the optimal forest OPT into two categories based on the number of edges cut: sets that cut exactly one edge of the optimal solution, and those that cut at least two. Since pairs in $C\mathcal{P}$ are connected in the optimal solution, any set cutting these pairs must cut at least one edge of OPT. Based on this, we allocate the value of cp between cp_1 and cp_2, representing the contribution from each category.

Definition 3.2 (Single-Edge and Multi-Edge Sets). For an instance I, we define a set $S \subset V$ as a single-edge set if it cuts exactly one edge of OPT (i.e., $d_{F^*}(S) = 1$) and as a multi-edge set if it cuts at least two edges of OPT (i.e., $d_{F^*}(S) > 1$). Let cp_1 and cp_2 denote the contribution to cp from y_{Sij} values corresponding to single-cut and multi-cut sets, respectively. These values are formally defined as follows:

$$cp_1 = \sum_{(i,j)\in C\mathcal{P}} \sum_{\substack{S:S\odot(i,j),\\ d_{F^*}(S)=1}} y_{Sij}$$
$$cp_2 = \sum_{(i,j)\in C\mathcal{P}} \sum_{\substack{S:S\odot(i,j),\\ d_{F^*}(S)>1}} y_{Sij}.$$

Figure 1 displays a single-edge set on the left and a multi-edge set on the right.

Lemma 3.1. *For an instance I, we have $cp_1 + cp_2 = cp$.*

Proof. Since pairs in $C\mathcal{P}$ are connected by the optimal solution OPT, any set S cutting a pair in $C\mathcal{P}$ must cut at least one edge of OPT. Therefore, any set S contributing to cp is either a single-edge set or a multi-edge set. Hence, we have $cp_1 + cp_2 = cp$. □

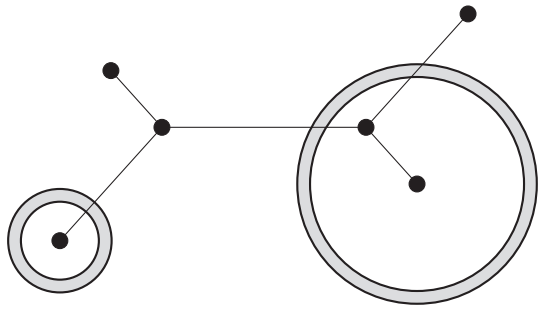

Fig. 1. A comparison between a single-cut set (left) and a multi-cut set (right).

Now, we use these definitions and categorizations to analyze our algorithm. All of the following lemmas are based on the assumption that IPCSF is executed on instance I. First, in Lemma 3.2, we provide a lower bound on the cost of the optimal solution, which is $cost(OPT) \geq cc + cp + cp_2 + pc + pp$. Next, in Lemma 3.3, we present an upper bound on the output of PCSF3(I), which is $cost_1 \leq 2cc + 2pc + 3cp + 3pp$. Moreover, in Lemma 3.4, we show that this value is at most $2cost(OPT) + cp_1 - cp_2 + pp$.

Next, we want to upper bound the output of the recursive call within IPCSF. In Lemma 3.6, we initially prove that $cost_R(OPT_R) \leq cost(OPT) - pp - cp_1$, where $cost_R(OPT_R)$ represents the cost of the optimal solution for the instance R defined at Line 14. Finally, in Theorem 3.7, we employ induction to demonstrate that $cost(\text{IPCSF}) \leq 2cost(OPT)$. Here, $cost(\text{IPCSF})$ denotes the cost of the output produced by IPCSF on instance I. To accomplish this, we use the same induction to bound the cost of the solution obtained through the recursive call at Line 16 by $cost_2 \leq 2cost_R(OPT_R) + cp + pp$, and by utilizing Lemma 3.6, we can then conclude that $cost_2 \leq 2cost(OPT) - cp_1 + cp_2 - pp$. Taking the average of $cost_1$ and $cost_2$ results in a value that is at most $2cost(OPT)$. Consequently, the minimum of these two values, corresponding to the cost of IPCSF(I), is at most $2cost(OPT)$.

Lemma 3.2. *For an instance I, we can derive a lower bound for the cost of the optimal solution OPT as follows:*

$$cost(OPT) \geq cc + cp + cp_2 + pc + pp.$$

Proof. The optimal solution pays penalties for pairs with labels $\mathcal{PC}$ and $\mathcal{PP}$ as it does not connect them. Since $y_{ij} \leq \pi_{ij}$ for each pair (i, j) by Corollary 2.2, we can lower bound the penalty paid by OPT as

$$\pi(Q^*) = \sum_{(i,j)\in(\mathcal{PC}\cup\mathcal{PP})} \pi_{ij} \geq \sum_{(i,j)\in(\mathcal{PC}\cup\mathcal{PP})} y_{ij} = pc + pp.$$

Now, we want to lower bound the cost of the forest in the optimal solution by $cc + cp + cp_2$. We show this using the properties of the primal-dual algorithm.

$$\begin{aligned} c(F^*) &= \sum_{e\in F^*} c_e \\ &\geq \sum_{e\in F^*} \sum_{\substack{S\subset V \\ e\in\delta(S)}} y_S && \text{(Corollary 2.1)} \\ &= \sum_{S\subset V} \sum_{\substack{e\in F^* \\ e\in\delta(S)}} y_S && \text{(Change the order of summations)} \end{aligned}$$

$$
\begin{aligned}
&= \sum_{S \subset V} d_{F^*}(S) \cdot y_S \\
&= \sum_{S \subset V} \sum_{(i,j): S \odot (i,j)} d_{F^*}(S) \cdot y_{Sij} && (y_S = \textstyle\sum_{(i,j): S \odot (i,j)} y_{Sij}) \\
&= \sum_{(i,j) \in V \times V} \sum_{S: S \odot (i,j)} d_{F^*}(S) \cdot y_{Sij} && \text{(Change the order of summations)} \\
&\geq \sum_{(i,j) \in CC} \sum_{S \odot (i,j)} d_{F^*}(S) \cdot y_{Sij} + \sum_{(i,j) \in C\mathcal{P}} \sum_{S \odot (i,j)} d_{F^*}(S) \cdot y_{Sij}. && (CC \cap C\mathcal{P} = \emptyset)
\end{aligned}
$$

For each pair $(i, j) \in (CC \cup C\mathcal{P})$, we know that i and j are connected in the optimal solution OPT. This implies that for every set S satisfying $S \odot (i, j)$, the set S cuts the forest of OPT—that is, $d_{F^*}(S) \geq 1$. Based on this observation, we bound the two terms in the preceding summation separately. For pairs in CC, we have

$$
\sum_{(i,j) \in CC} \sum_{S \odot (i,j)} d_{F^*}(S) \cdot y_{Sij} \geq \sum_{(i,j) \in CC} \sum_{S \odot (i,j)} y_{Sij} = \sum_{(i,j) \in CC} y_{ij} = cc.
$$

For pairs in $C\mathcal{P}$, we have

$$
\begin{aligned}
\sum_{(i,j) \in C\mathcal{P}} \sum_{S \odot (i,j)} d_{F^*}(S) \cdot y_{Sij} &= \sum_{(i,j) \in C\mathcal{P}} \sum_{\substack{S \odot (i,j), \\ d_{F^*}(S)=1}} d_{F^*}(S) \cdot y_{Sij} + \sum_{(i,j) \in C\mathcal{P}} \sum_{\substack{S \odot (i,j), \\ d_{F^*}(S)>1}} d_{F^*}(S) \cdot y_{Sij} \\
&\geq \sum_{(i,j) \in C\mathcal{P}} \sum_{\substack{S \odot (i,j), \\ d_{F^*}(S)=1}} y_{Sij} + \sum_{(i,j) \in C\mathcal{P}} \sum_{\substack{S \odot (i,j), \\ d_{F^*}(S)>1}} 2y_{Sij} \\
&= cp_1 + 2cp_2 \\
&= cp + cp_2. && \text{(Lemma 3.1)}
\end{aligned}
$$

Summing up all the components, we have

$$
cost(OPT) = c(F^*) + \pi(Q^*) \geq cc + cp + cp_2 + pc + pp.
$$

□

Lemma 3.3. *For an instance I, during the first iteration of IPCSF(I) where PCSF3(I) is invoked, we can establish an upper bound on the output of PCSF3 as follows:*

$$
cost_1 \leq 2cc + 2pc + 3cp + 3pp.
$$

Proof. Since $cost_1$ is the total cost of PCSF3(I), we should bound $\pi(Q_1) + c(F_1')$. First, let us observe that PCSF3 pays the penalty for exactly the pairs (i, j) in $C\mathcal{P} \cup \mathcal{P}\mathcal{P}$, where $C\mathcal{P} \cup \mathcal{P}\mathcal{P} = Q_1$. Since every pair in Q_1 is tight, we have $\pi_{ij} = y_{ij}$ for these pairs. Therefore, the total penalty paid by PCSF3 can be bounded by

$$
\pi(Q_1) = \sum_{(i,j) \in (C\mathcal{P} \cup \mathcal{P}\mathcal{P})} \pi_{ij} = \sum_{(i,j) \in (C\mathcal{P} \cup \mathcal{P}\mathcal{P})} y_{ij} = cp + pp.
$$

Now, it suffices to show that $c(F_1') \leq 2(cc + cp + pc + pp)$. Since each pair belongs to exactly one of the sets CC, $C\mathcal{P}$, $\mathcal{P}C$, and $\mathcal{P}\mathcal{P}$, we can observe that

$$
cc + cp + pc + pp = \sum_{(i,j) \in V \times V} y_{ij} = \sum_{S \subset V} y_S.
$$

Therefore, the desired statement is implied by Lemma 2.15. □

Lemma 3.4. *For an instance I, during the first iteration of IPCSF(I) where PCSF3(I) is invoked, we can establish an upper bound on the output of PCSF3 as follows:*

$$cost_1 \leq 2cost(OPT) + cp_1 - cp_2 + pp.$$

Proof. We can readily prove this by referring to the previous lemmas:

$$\begin{aligned} cost_1 &\leq 2cc + 2pc + 3cp + 3pp && \text{(Lemma 3.3)} \\ &= 2(cc + cp + cp_2 + pc + pp) + cp - 2cp_2 + pp \\ &\leq 2cost(OPT) + (cp - cp_2) - cp_2 + pp && \text{(Lemma 3.2)} \\ &= 2cost(OPT) + cp_1 - cp_2 + pp. && \text{(Lemma 3.1)} \end{aligned}$$

□

Lemma 3.5. *For an instance I, it is possible to remove a set of edges from F^* with a total cost of at least cp_1 while ensuring that the pairs in CC remain connected.*

Proof. Consider a single-edge set S that cuts some pair (i, j) in $\mathcal{CP}$ with $y_{Sij} > 0$. Since (i, j) is in $\mathcal{CP}$, it is also in Q_1 and therefore tight. By Corollary 2.9, any other pair cut by S will also be tight. Consequently, the pairs in CC will not be cut by S since they are not tight. Furthermore, according to Lemma 1.2, if S cuts only one edge e of F^*, then the only pairs that will be disconnected by removing edge e from F^* are the pairs that are cut by S. However, we have already shown that no pair in CC is cut by S. Therefore, all pairs in CC will remain connected even after removing edge e. This is illustrated in Figure 2.

For any single-edge set S that cuts a pair (i, j) in $\mathcal{CP}$ with $y_{Sij} > 0$, we can safely remove the single edge of F^* that is cut by S. Let D_{single} denote the collection of such single-edge sets and E_{single} be the set of removed edges attached to sets in D_{single}. Then we have

$$\begin{aligned} c(E_{single}) &= \sum_{e \in E_{single}} c_e \\ &\geq \sum_{e \in E_{single}} \sum_{\substack{S \subset V \\ e \in \delta(S)}} y_S && \text{(Corollary 2.1)} \\ &= \sum_{S \subset V} \sum_{\substack{e \in E_{single} \\ e \in \delta(S)}} y_S && \text{(Change the order of summations)} \\ &= \sum_{S \subset V} |\delta(S) \cap E_{single}| y_S && \text{(Simplified)} \\ &\geq \sum_{S \in D_{single}} y_S && (\delta(S) \cap E_{single} \neq \emptyset \text{ for } S \in D_{single}) \\ &= \sum_{S \in D_{single}} \sum_{(i,j): S \odot (i,j)} y_{Sij} && (y_S = \textstyle\sum_{(i,j): S \odot (i,j)} y_{Sij}) \\ &\geq \sum_{S: d_{F^*}(S) = 1} \sum_{\substack{(i,j) \in \mathcal{CP} \\ S \odot (i,j) \\ y_{Sij} > 0}} y_{Sij} \\ &= \sum_{S: d_{F^*}(S) = 1} \sum_{\substack{(i,j) \in \mathcal{CP} \\ S \odot (i,j)}} y_{Sij} \\ &= cp_1. \end{aligned}$$

Therefore, the total length of the removed edges will be at least cp_1. □

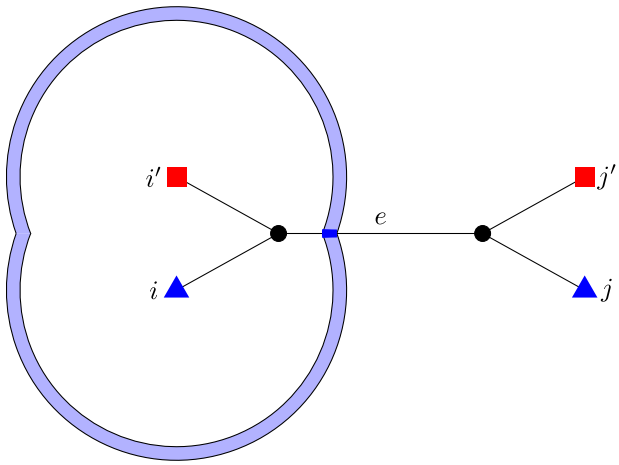

Fig. 2. The figure shows the graph of F^* with pairs (i, j) and (i', j'), and a single-edge set S with $y_{Sij} > 0$. Tightness of (i, j) implies tightness of (i', j'), and removing edge e does not disconnect pairs in CC.

Now, we introduce some useful notation to analyze the output of the recursive call. During the execution of IPCSF on an instance I, it generates a modified instance R at Line 14, where the penalties for pairs in Q_1 are set to 0. We use the notation π' to represent the penalties in the instance R as they are defined in Lines 6 through 11. Since Line 2 ensures that $\pi(Q_1) \neq 0$, we can conclude that R is a reduced instance compared to I, meaning that the number of pairs with non-zero penalties is smaller in R than in I. Given that we recursively call IPCSF on instance R, we can bound the output of the recursive call by the optimal solution of R using induction. Let OPT_R be an optimal solution for R. We denote the forest of OPT_R as F^*_R and the set of pairs not connected by F^*_R as Q^*_R. The cost of OPT_R is given by $cost_R(OPT_R) = c(F^*_R) + \pi'(Q^*_R)$. We will use these notations in the following lemmas.

Lemma 3.6. *For an instance I and the instance R constructed at Line 14 during the execution of IPCSF(I), we have*

$$cost_R(OPT_R) \leq cost(OPT) - pp - cp_1.$$

Proof. To prove this lemma, we first provide a solution for the instance R given the optimal solution of the instance I, denoted as OPT, and we show that the cost of this solution is at most $cost(OPT) - pp - cp_1$. Since OPT_R is a solution for the instance R with the minimum cost, we can conclude that $cost_R(OPT_R) \leq cost(OPT) - pp - cp_1$.

To provide the aforementioned solution for the instance R, we start with the solution OPT consisting of the forest F^* and the set of pairs for which penalties were paid, denoted as Q^*. We create a new set $Q'_R = Q^* \cup \mathcal{CP} = \mathcal{PC} \cup \mathcal{PP} \cup \mathcal{CP}$ and a forest F'_R initially equal to F^*. Since F^* connects pairs in CC and $\mathcal{CP}$, but we add pairs in $\mathcal{CP}$ to Q'_R and pay their penalties, we can remove edges from F'_R that do not connect pairs in CC.

Let us focus on Q'_R first. Since the penalties for pairs in $\mathcal{CP}$ and $\mathcal{PP}$ are set to 0 in π', we have

$$\begin{aligned} \pi'(Q'_R) &= \pi'(\mathcal{CP}) + \pi'(\mathcal{PC}) + \pi'(\mathcal{PP}) && (Q'_R = \mathcal{CP} \cup \mathcal{PC} \cup \mathcal{PP}) \\ &= \pi'(\mathcal{PC}) && (\pi'(\mathcal{CP}) = \pi'(\mathcal{PP}) = 0) \\ &= \pi(\mathcal{PC}) \end{aligned}$$

$$
\begin{aligned}
&= \pi(Q^*) - \pi(\mathcal{PP}) && (Q^* = \mathcal{PC} \cup \mathcal{PP}) \\
&= \pi(Q^*) - \sum_{(i,j)\in\mathcal{PP}} \pi_{ij} \\
&= \pi(Q^*) - \sum_{(i,j)\in\mathcal{PP}} y_{ij} && (\text{Pairs in } \mathcal{PP} \text{ are tight}) \\
&= \pi(Q^*) - pp.
\end{aligned}
$$

Moreover, using Lemma 3.5, we construct F'_R from F^* by removing a set of edges with a total length of at least cp_1, while ensuring that the remaining forest still connects all the pairs in CC. Therefore, we can bound the cost of F'_R as

$$c(F'_R) \leq c(F^*) - cp_1.$$

Summing it all together, we have

$$cost_R(OPT_R) \leq c(F'_R) + \pi'(Q'_R) \leq (c(F^*) - cp_1) + (\pi(Q^*) - pp) = cost(OPT) - pp - cp_1,$$

where the first inequality comes from the fact that OPT_R is the optimal solution for the instance R, whereas (Q'_R, F'_R) gives a valid solution—that is, F'_R connects every pair that is not in Q'_R. □

Finally, we can upper bound the cost of the output of IPCSF. For an instance I, let us denote the cost of the output of IPCSF(I) as $cost$(IPCSF). In Theorem 3.7, we prove that the output of IPCSF is a 2-approximate solution for the PCSF problem.

Theorem 3.7. *For an instance I, the output of IPCSF(I) is a 2-approximate solution to the optimal solution for I, meaning that*

$$cost(IPCSF) \leq 2cost(OPT).$$

Proof. We will prove the claim by induction on the number of pairs (i, j) with penalty $\pi_{ij} > 0$ in instance I.

First, the algorithm makes a call to the PCSF3 procedure to obtain a solution (Q_1, F'_1). If $\pi(Q_1) = 0$ for this solution, which means no cost is incurred by paying penalties, the algorithm terminates and returns this solution at Line 3. This will always be the case in the base case of our induction where for all pairs $(i, j) \in Q_1$, penalties π_{ij} are equal to 0. Since every pair $(i, j) \in Q_1$ is tight, we have $y_{ij} = \pi_{ij} = 0$. Given that $\mathcal{CP}$ and $\mathcal{PP}$ are subsets of Q_1, we can conclude that $cp = cp_1 = cp_2 = pp = 0$. Now, by Lemma 3.4, we have

$$cost_1 \leq 2cost(OPT) + (cp_1 - cp_2) + pp = 2cost(OPT).$$

Therefore, when IPCSF returns at Line 3, we have

$$cost(\text{IPCSF}) = cost_1 \leq 2cost(OPT),$$

and we obtain a 2-approximation of the optimal solution.

Now, let us assume that PCSF3 pays penalties for some pairs—that is, $\pi(Q_1) \neq 0$. Therefore, since we set the penalty of pairs in Q_1 equal to 0 for instance R at Line 9, the number of pairs with non-zero penalty in instance R is less than in instance I. By induction, we know that the output of IPCSF on instance R, denoted as (Q_2, F'_2), has a cost of at most $2cost_R(OPT_R)$. That means

$$c(F'_2) + \pi'(Q_2) \leq 2cost_R(OPT_R).$$

In addition, we have

$$\pi(Q_2) = \pi(Q_2 \setminus Q_1) + \pi(Q_2 \cap Q_1) \leq \pi'(Q_2 \setminus Q_1) + \pi(Q_1) \leq \pi'(Q_2) + \pi(Q_1),$$

where we use the fact that $\pi'_{ij} = \pi_{ij}$ for $(i,j) \notin Q_1$. Now we can bound the cost of the solution (Q_2, F'_2), denoted as $cost_2$, by

$$\begin{aligned}
cost_2 &= c(F'_2) + \pi(Q_2) \\
&\le c(F'_2) + \pi'(Q_2) + \pi(Q_1) \\
&\le 2cost_R(OPT_R) + \pi(Q_1) && \text{(By induction)} \\
&\le 2\left(cost(OPT) - pp - cp_1\right) + \sum_{(i,j)\in Q_1} \pi_{ij} && \text{(Lemma 3.6)} \\
&= 2\left(cost(OPT) - pp - cp_1\right) + \sum_{(i,j)\in Q_1} y_{ij} && \text{(Pairs in } Q_1 \text{ are tight)} \\
&= 2cost(OPT) - 2pp - 2cp_1 + cp + pp \\
&= 2cost(OPT) - cp_1 + cp_2 - pp. && \text{(Lemma 3.1)}
\end{aligned}$$

Furthermore, according to Lemma 3.4, the cost of the solution (Q_1, F'_1), denoted as $cost_1$, can be upper bounded by

$$cost_1 \le 2OPT + cp_1 - cp_2 + pp.$$

Finally, in Line 17, we return the solution with the smaller cost between (Q_1, F'_1) and (Q_2, F'_2). Based on the upper bounds above on both solutions, we know that

$$\begin{aligned}
cost(\text{IPCSF}) = \min(cost_1, cost_2) &\le \frac{1}{2}(cost_1 + cost_2) \\
&\le \frac{1}{2}(2cost(OPT) + cp_1 - cp_2 + pp + 2cost(OPT) - cp_1 + cp_2 - pp) \\
&= \frac{1}{2}(4cost(OPT)) = 2cost(OPT),
\end{aligned}$$

and we obtain a 2-approximation of the optimal solution. This completes the induction step and the proof of the theorem. □

It is interesting to note that as we use the average of $cost_1$ and $cost_2$ as an upper bound for the smaller one, the same analysis can be applied to a similar algorithm that chooses between the two solutions uniformly at random instead of choosing the one with the minimum cost.

Theorem 3.8. *The runtime of the IPCSF algorithm is polynomial.*

Proof. Let n be the number of vertices in the input graph. There are $O(n^2)$ pairs of vertices in total. Whenever IPCSF calls itself recursively, the number of pairs with non-zero penalties decreases by at least 1, and otherwise IPCSF will return at Line 3. Thus, the recursion depth is polynomial in n. At each recursion level, the algorithm only runs PCSF3 on one instance of the problem and performs $O(n^2)$ additional operations. By Lemma 2.16, we know that PCSF3 runs in polynomial time. Therefore, the total runtime of IPCSF will also be polynomial. □

3.2 Improving the Approximation Ratio

In this section, we briefly explain how a tighter analysis can be used to show that the approximation ratio of the IPCSF algorithm is at most $2 - \frac{1}{n}$, where n is the number of vertices in the input graph G. This approximation ratio more closely matches the approximation ratio of $2 - \frac{2}{n}$ for the Steiner forest problem.

We first introduce an improved version of Lemmas 3.3 and 3.4.

Lemma 3.9. *For an instance I, during the first iteration of IPCSF(I) where PCSF3(I) is invoked, we have the following upper bound:*

$$cost_1 \leq \left(2 - \frac{2}{n}\right) \cdot cc + \left(2 - \frac{2}{n}\right) \cdot pc + \left(3 - \frac{2}{n}\right) \cdot cp + \left(3 - \frac{2}{n}\right) \cdot pp.$$

Proof. We proceed similarly to the proof of Lemma 3.3 and make a slight change. In one of the last steps of that proof, we use the following inequality:

$$\sum_{v \in V_a \cup V_i} d_H(v) - \sum_{v \in V_i} d_H(v) \leq 2(|V_a| + |V_i|) - \sum_{v \in V_i} d_H(v).$$

This is true, as H is a forest and its number of edges is less than its number of vertices. However, as the number of edges in a forest is strictly less than the number of vertices, we can lower the right-hand side of this inequality to $2(|V_a| + |V_i| - 1) - \sum_{v \in V_i} d_H(v)$. Rewriting the main inequality in this step with this change gives us

$$\begin{aligned}
\sum_{S \in ActS} d_{F_1'}(S) &\leq 2(|V_a| + |V_i| - 1) - \sum_{v \in V_i} d_H(v) \\
&\leq 2(|V_a| + |V_i| - 1) - 2|V_i| && (d_H(v) \geq 2 \text{ for } v \in V_i) \\
&\leq 2(|V_a| - 1) = 2|ActS| - 2 && (|V_a| = |ActS|) \\
&= \left(2 - \frac{2}{|ActS|}\right) |ActS| \\
&\leq (2 - \frac{2}{n})|ActS|. && (|ActS| \leq n)
\end{aligned}$$

Based on the steps in the proof of Lemma 3.3, this leads to the desired upper bound. □

Lemma 3.10. *For an instance I, during the first iteration of IPCSF(I) where PCSF3(I) is invoked, we can establish an upper bound on the output of PCSF3 as follows:*

$$cost_1 \leq \left(2 - \frac{2}{n}\right) \cdot cost(OPT) + cp_1 - \left(1 - \frac{2}{n}\right) \cdot cp_2 + pp.$$

Proof. We prove this lemma similarly to Lemma 3.4, except we use Lemma 3.9 instead of Lemma 3.3:

$$\begin{aligned}
cost_1 &\leq \left(2 - \frac{2}{n}\right) \cdot cc + \left(2 - \frac{2}{n}\right) \cdot pc + \left(3 - \frac{2}{n}\right) \cdot cp + \left(3 - \frac{2}{n}\right) \cdot pp && \text{(Lemma 3.9)} \\
&= \left(2 - \frac{2}{n}\right)(cc + cp + cp_2 + pc + pp) + cp - \left(2 - \frac{2}{n}\right) \cdot cp_2 + pp \\
&\leq \left(2 - \frac{2}{n}\right) \cdot cost(OPT) + (cp - cp_2) - \left(1 - \frac{2}{n}\right) cp_2 + pp && \text{(Lemma 3.2)} \\
&= \left(2 - \frac{2}{n}\right) \cdot cost(OPT) + cp_1 - \left(1 - \frac{2}{n}\right) \cdot cp_2 + pp. && \text{(Lemma 3.1)}
\end{aligned}$$

□

Finally, we improve Theorem 3.7.

Theorem 3.11. *For an instance I, the output of IPCSF(I) is a $(2 - \frac{1}{n})$-approximate solution to the optimal solution for I, meaning that*

$$cost(IPCSF) \leq \left(2 - \frac{1}{n}\right) \cdot cost(OPT).$$

Proof. Similarly to the proof of Theorem 3.7, we use induction on the number of non-zero penalties. If the algorithm terminates on Line 3, then by Lemma 3.10 we have

$$cost_1 \leq \left(2 - \frac{2}{n}\right) \cdot cost(OPT) + cp_1 - \left(1 - \frac{2}{n}\right) \cdot cp_2 + pp = \left(2 - \frac{2}{n}\right) \cdot cost(OPT)$$

since cp_1, cp_2, and pp are all 0 in this case. As $2 - \frac{2}{n} \leq 2 - \frac{1}{n}$, the desired inequality holds in this case. This establishes our base case for the induction.

Using the same reasoning as the proof of Theorem 3.7, based on the induction we have

$$\begin{aligned}
cost_2 &\leq \left(2 - \frac{1}{n}\right) \cdot cost_R(OPT_R) + \pi(Q_1) \\
&= \left(2 - \frac{1}{n}\right) \cdot cost_R(OPT_R) + cp + pp \\
&\leq \left(2 - \frac{1}{n}\right)(cost(OPT) - cp_1 - pp) + cp + pp && \text{(By Lemma 3.6)} \\
&\leq \left(2 - \frac{1}{n}\right) \cdot cost(OPT) - \left(1 - \frac{1}{n}\right) \cdot cp_1 + cp_2 - \left(1 - \frac{1}{n}\right) \cdot pp.
\end{aligned}$$

We can combine this with the following upper bound from Lemma 3.10:

$$cost_1 \leq \left(2 - \frac{2}{n}\right) \cdot cost(OPT) + cp_1 - \left(1 - \frac{2}{n}\right) \cdot cp_2 + pp.$$

As the algorithm chooses the solution with the lower cost between $cost_1$ and $cost_2$, we have

$$\begin{aligned}
cost(\text{IPCSF}) = \min(cost_1, cost_2) &\leq \frac{1}{2}(cost_1 + cost_2) \\
&\leq \frac{1}{2}\left[\left(2 - \frac{2}{n}\right) \cdot cost(OPT) + cp_1 - \left(1 - \frac{2}{n}\right) \cdot cp_2 + pp \right. \\
&\quad \left. + \left(2 - \frac{1}{n}\right) \cdot cost(OPT) - \left(1 - \frac{1}{n}\right) \cdot cp_1 + cp_2 - \left(1 - \frac{1}{n}\right) \cdot pp\right] \\
&= \frac{1}{2}\left(\left(4 - \frac{3}{n}\right) \cdot cost(OPT) + \frac{2}{n}cp_2 + \frac{1}{n}cp_1 + \frac{1}{n}pp\right) \\
&\leq \frac{1}{2}\left(\left(4 - \frac{2}{n}\right) \cdot cost(OPT) + \frac{1}{n}[2cp_2 + cp_1 + pp - cost(OPT)]\right) \\
&\leq \frac{1}{2}\left(4 - \frac{2}{n}\right) \cdot cost(OPT) && (cost(OPT) \geq 2cp_2 + cp_1 + pp \text{ by Lemma 3.2}) \\
&= \left(2 - \frac{1}{n}\right) \cdot cost(OPT).
\end{aligned}$$

Therefore, the algorithm obtains a $(2 - \frac{1}{n})$-approximation of the optimal solution. □

References

[1] Ajit Agrawal, Philip Klein, and R. Ravi. 1991. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC '91)*. ACM, New York, NY, USA, 134–144. https://doi.org/10.1145/103418.103437

[2] Ajit Agrawal, Philip N. Klein, and R. Ravi. 1995. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM Journal on Computing* 24, 3 (1995), 440–456. https://doi.org/10.1137/S0097539792236237

[3] Ali Ahmadi, Iman Gholami, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Mohammad Mahdavi. 2024. 2-Approximation for prize-collecting Steiner forest. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA '24)*. 669–693. https://doi.org/10.1137/1.9781611977912.25

[4] Ali Ahmadi, Iman Gholami, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Mohammad Mahdavi. 2024. Prize-collecting Steiner tree: A 1.79 approximation. In *Proceedings of the 56th Annual ACM symposium on Theory of Computing (STOC '24)*. ACM, New York, NY, USA, 1641–1652. https://doi.org/10.1145/3618260.3649789

[5] Aaron Archer, MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Howard J. Karloff. 2011. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM Journal on Computing* 40, 2 (2011), 309–332. https://doi.org/10.1137/090771429

[6] Egon Balas. 1989. The prize collecting traveling salesman problem. *Networks* 19, 6 (1989), 621–636. https://doi.org/10.1002/net.3230190602

[7] MohammadHossein Bateni and MohammadTaghi Hajiaghayi. 2012. Euclidean prize-collecting Steiner forest. *Algorithmica* 62, 3-4 (2012), 906–929. https://doi.org/10.1007/s00453-011-9491-8

[8] Marshall W. Bern and Paul E. Plassmann. 1989. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters* 32, 4 (1989), 171–176. https://doi.org/10.1016/0020-0190(89)90039-2

[9] Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David P. Williamson. 1993. A note on the prize collecting traveling salesman problem. *Mathematical Programming* 59 (1993), 413–420. https://doi.org/10.1007/BF01581256

[10] Jannis Blauth, Nathan Klein, and Martin Nägele. 2023. A better-than-1.6-approximation for prize-collecting TSP. *CoRR abs/2308.06254* (2023). https://doi.org/10.48550/ARXIV.2308.06254

[11] Jannis Blauth and Martin Nägele. 2023. An improved approximation guarantee for prize-collecting TSP. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC '23)*. ACM, New York, NY, USA, 1848–1861. https://doi.org/10.1145/3564246.3585159

[12] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanità. 2010. An improved LP-based approximation for Steiner tree. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*. ACM, New York, NY, USA, 583–592. https://doi.org/10.1145/1806689.1806769

[13] Miroslav Chlebík and Janka Chlebíková. 2008. The Steiner tree problem on graphs: Inapproximability results. *Theoretical Computer Science* 406, 3 (2008), 207–214. https://doi.org/10.1016/j.tcs.2008.06.046

[14] Michel X. Goemans. 2009. Combining approximation algorithms for the prize-collecting TSP. *CoRR abs/0910.0553* (2009). http://arxiv.org/abs/0910.0553

[15] Michel X. Goemans and David P. Williamson. 1992. A general approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '92)*. 307–316.

[16] Michel X. Goemans and David P. Williamson. 1995. A general approximation technique for constrained forest problems. *SIAM Journal on Computing* 24, 2 (1995), 296–317. https://doi.org/10.1137/S0097539793242618

[17] Anupam Gupta, Jochen Könemann, Stefano Leonardi, R. Ravi, and Guido Schäfer. 2007. An efficient cost-sharing mechanism for the prize-collecting Steiner forest problem. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. 1153–1162. http://dl.acm.org/citation.cfm?id=1283383.1283507

[18] Anupam Gupta and Amit Kumar. 2015. Greedy algorithms for Steiner forest. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC '15)*. ACM, New York, NY, USA, 871–878. https://doi.org/10.1145/2746539.2746590

[19] MohammadTaghi Hajiaghayi and Arefeh A. Nasri. 2010. Prize-collecting Steiner networks via iterative rounding. In *LATIN 2010: Theoretical Informatics*. Lecture Notes in Computer Science, Vol. 6034. Springer, 515–526. https://doi.org/10.1007/978-3-642-12200-2_45

[20] Mohammad Taghi Hajiaghayi and Kamal Jain. 2006. The prize-collecting generalized Steiner tree problem via a new approach of primal-dual schema. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*. ACM, New York, NY, USA, 631–640. http://dl.acm.org/citation.cfm?id=1109557.1109626

[21] Mohammad Taghi Hajiaghayi, Rohit Khandekar, Guy Kortsarz, and Zeev Nutov. 2012. Prize-collecting Steiner network problems. *ACM Transactions on Algorithms* 9, 1 (2012), Article 2, 13 pages. https://doi.org/10.1145/2390176.2390178

[22] Dorit S. Hochbaum (Ed.). 1996. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company..

[23] Richard M. Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, Raymond E. Miller and James W. Thatcher (Eds.). IBM Research Symposia Series. Plenum Press, New York, NY, USA, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[24] Marek Karpinski and Alexander Zelikovsky. 1997. New approximation algorithms for the Steiner tree problems. *Journal of Combinatorial Optimization* 1, 1 (1997), 47–65. https://doi.org/10.1023/A:1009758919736

[25] Jochen Könemann, Neil Olver, Kanstantsin Pashkovich, R. Ravi, Chaitanya Swamy, and Jens Vygen. 2017. On the integrality gap of the prize-collecting Steiner forest LP. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques (APPROX/RANDOM 2017)*, Klaus Jansen, José D. P. Rolim, David Williamson,

and Santosh S. Vempala (Eds.). Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Article 17, 13 pages. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2017.17

[26] Gabriel Robins and Alexander Zelikovsky. 2005. Tighter bounds for graph Steiner tree approximation. *SIAM Journal on Discrete Mathematics* 19, 1 (2005), 122–134. https://doi.org/10.1137/S0895480101393155

[27] Yogeshwer Sharma, Chaitanya Swamy, and David P. Williamson. 2007. Approximation algorithms for prize collecting forest problems with submodular penalty functions. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. 1275–1284. http://dl.acm.org/citation.cfm?id=1283383.1283520

[28] Alexander Zelikovsky. 1993. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica* 9, 5 (1993), 463–470. https://doi.org/10.1007/BF01187035

Received 30 November 2023; revised 7 January 2025; accepted 1 March 2025