Polylogarithmic Approximations for Robust s-t Path

Shi Li[1]
Department of Computer Science and Technology, Nanjing University, Jiangsu, China

Chenyang Xu[1]
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China

Ruilong Zhang[1]
Department of Computer Science, City University of Hong Kong, Hong Kong, China

Abstract

The paper revisits the Robust s-t Path problem, one of the most fundamental problems in robust optimization. In the problem, we are given a directed graph with n vertices and k distinct cost functions (scenarios) defined over edges, and aim to choose an s-t path such that the total cost of the path is always provable no matter which scenario is realized. Viewing each cost function as an agent, our goal is to find a fair s-t path, which minimizes the maximum cost among all agents. The problem is NP-hard to approximate within a factor of $o(\log k)$ unless NP $\subseteq$ DTIME($n^{\text{poly}\log n}$), and the best-known approximation ratio is $\widetilde{O}(\sqrt{n})$, which is based on the natural flow linear program. A longstanding open question is whether we can achieve a polylogarithmic approximation for the problem; it remains open even if a quasi-polynomial running time is allowed.

Our main result is a $O(\log n \log k)$ approximation for the Robust s-t Path problem in quasi-polynomial time, solving the open question in the quasi-polynomial time regime. The algorithm is built on a novel linear program formulation for a decision-tree-type structure, which enables us to overcome the $\Omega(\sqrt{n})$ integrality gap for the natural flow LP. Furthermore, we show that for graphs with bounded treewidth, the quasi-polynomial running time can be improved to a polynomial. We hope our techniques can offer new insights into this problem and other related problems in robust optimization.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Approximation Algorithm, Randomized LP Rounding, Robust s-t Path

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.106

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2305.16439` [25]

Funding Chenyang Xu is supported by the National Key Research Project of China under Grant No. 2023YFA1009402, the National Natural Science Foundation of China (No. 62302166), the Dean's Fund of Shanghai Key Laboratory of Trustworthy Computing, ECNU, and the Key Laboratory of Interdisciplinary Research of Computation and Economics (SUFE), Ministry of Education. Shi Li was supported by the State Key Laboratory for Novel Software Technology, and the New Cornerstone Science Laboratory.

1 Introduction

Robust optimization under uncertainty [5, 6, 17, 26] is one of the most important and challenging computational tasks in the real world. Uncertainty arises in many scenarios. For instance, the travel time for a road segment might be uncertain due to traffic jams. The paper revisits the Robust s-t Path problem [18], a cornerstone problem in the area of robust optimization. In the problem, there are several edge cost functions for a given graph and the

[1] All authors (ordered alphabetically) have equal contributions and are corresponding authors.

goal is to find an s-t path that minimizes the maximum cost across all the cost functions. Apart from serving a model for handling uncertainty, robustness also offers a method to integrate multiple objectives and fairness requirements.

In a routing network, each link (or edge) typically possesses multiple attributes, such as usage cost and delay. By representing each attribute as a cost function, we can formulate the multi-objective routing problem as our model [14, 15, 31]. In the context of fairness computation, diverse edge cost functions can be interpreted as various perspectives of agents on the edges. Our objective is to identify a public path that can accommodate these perspectives under the notion of min-max fairness [1, 27, 30].

The Robust s-t Path problem was initially studied by [18], and since then it has received widespread attention due to its broad applicability. In [18], the authors demonstrated that the problem is strongly NP-Hard even when there are only two scenarios. Later, [2] considers the problem with the constant number of scenarios, and shows that the problem admits a fully polynomial-time approximation scheme (FPTAS). When the number k of scenarios is part of the input, simply computing the shortest path w.r.t the summation of the k cost functions can obtain an approximation ratio of k. Kasperski and Zielinski [20] proved that the problem is hard to approximate within a factor of $o(\log k)$ unless NP $\subseteq$ DTIME($n^{\text{poly}\log n}$). Whether a polylogarithmic approximation can be achieved has been open ever since. A recent breakthrough in the approximation ratio is made by Kasperski and Zielinski [23] in which they gave a flow-LP-based algorithm that is $\widetilde{O}(\sqrt{n})$-approximate, where we use $\widetilde{O}$ to hide a polylogarithmic factor. They further showed that their analysis is nearly tight by proving an integrality gap of $\Omega(\sqrt{n})$ for the flow LP.

It should be noted that Bilò et al. [7] studied the ℓ_q-norm shortest path problem which is a generalized version of robust s-t path, i.e., the problem aims to find a s-t path $\mathcal{P}$ to minimize the value of $\left(\sum_{i\in[k]} c_i(\mathcal{P})^q\right)^{1/q}$, where $c_i(\mathcal{P})$ is agent i's cost for the selected path $\mathcal{P}$. Their algorithm [7, Algorithm 2]) extends the classical Dijkstra algorithm by replacing the distance with the ℓ_q-norm metric. Our directed graph is a DAG, so the Dijkstra-type algorithm becomes a dynamic programming-type algorithm, with nodes processed using a topological order. So, their algorithm just stores the best path in the ℓ_q-norm for every node. It is claimed in [7, Theorem 14] that such an algorithm achieves $O(\min\{q, \log k\})$-approximation for the ℓ_q-norm shortest path problem. However, unfortunately, there exists a crucial error in the analysis. In the full version [25], we give a hard instance on a series-parallel graph for the algorithm and show that the approximation ratio is at least $k^{1-1/q}$. In other words, when $q = O(\log k)$, the proposed algorithm [7, Algorithm 2] is $O(k)$-approximate for the robust s-t problem.

1.1 Our Contributions

This paper makes significant progress in closing the gap between the known upper and the lower bound for Robust s-t Path. We show that for two natural graph classes, a polylogarithmic approximation can be obtained in polynomial time; while for general graphs, there exists a polylogarithmic approximated algorithm running in a quasi-polynomial time. The following formalizes the model and summarizes our main results.

The Robust s-t Path Problem. Consider a directed graph $G(V, E)$ with n vertices and m edges. There are k scenarios (also referred to as "agents" hereafter), where each scenario $i \in [k]$ has an edge cost function $c_i : E \to \mathbb{R}_{\geq 0}$. Given two specified vertices s and t in the graph, the goal is to find an s-t path $\mathcal{P}$ that minimizes $\max_{i\in[k]} c_i(\mathcal{P})$, where $c_i(\mathcal{P}) := \sum_{e\in\mathcal{P}} c_i(e)$.

Main Result 1 (Theorem 1). There is a randomized polynomial-time $O(H \log k)$-approximation algorithm for Robust s-t Path for directed series-parallel graphs, where H is the height of the decomposition tree of the series-parallel graph and k is the number of agents.

Our first result is on the class of *series-parallel graphs* (Section 2), which are used by [20] to demonstrate a lower bound of $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) for the Robust s-t Path problem. We begin by showing that the natural flow linear program (LP) for this class has integrality gaps of $\Omega(k)$ and $\Omega(\sqrt{n})$. The gaps hold even when we integrate the knowledge of the optimum cost to the LP to circumvent some obvious gap instances. This result aligns with the prior findings of [23], but our constructed instance is significantly simpler. It is worth noting that most prior algorithms in the existing literature rely on the flow LP mentioned above, and thus, their approximation ratios cannot be better than $O(\min\{k, \sqrt{n}\})$.

To overcome the gap, we develop a novel linear program based on the *decomposition tree* of series-parallel graphs and demonstrate that a dependent randomized rounding algorithm for the LP obtains an approximation ratio of $O(H \log k)$. Particularly, for the hard instance that leads to a lower bound $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) stated in [20], our algorithm can return a $O(\log \log n \log k)$-approximate solution, which is nearly tight since there is only a $O(\log \log n)$-gap; the detailed discussion can be found in the full version [25].

Main Result 2 (Theorem 8). Given a directed graph with bounded treewidth, there is a randomized algorithm that obtains an approximation ratio of $O(\log n \log k)$ in polynomial time, where n is the number of vertices and k is the number of agents.

We then consider *graphs with bounded treewidth* (Section 3). When the treewidth is 2, it becomes the class of series-parallel graphs. Therefore, combining the above two results gives a $O(\min\{H, \log n\} \cdot \log k)$-approximation for series-parallel graphs. Besides series-parallel graphs, the graph class includes many other common graphs, such as trees, pseudoforests, Cactus graphs, outerplanar graphs, and Halin graphs. In this part, we employ the nice properties provided by the treewidth decomposition of these graphs and obtain a polylogarithmic approximation.

Main Result 3 (Theorem 12). Given any directed graph, there is a randomized algorithm that obtains a $O(\log n \log k)$-approximate solution in quasi-polynomial time, where n is the number of vertices and k is the number of agents. Moreover, any quasi-polynomial time algorithm for Robust s-t Path has an approximation lower bound of $\Omega(\log^{1-\epsilon} k)$ (even on series-parallel graphs) under the assumption that NP $\nsubseteq$ DTIME$(n^{\text{poly} \log n})$.

Finally, we consider general graphs (Section 4). The algorithm is also LP-based, following a similar framework as the algorithm for series-parallel graphs. The main challenge here is that we no longer have a simple tree structure for general graphs. To address this issue, we construct a decision-tree-type tree structure for the given graph and write a linear program based on it. Our algorithm then builds on this new LP to give the first polylogarithmic approximation for general graphs. Additionally, we show that the lower bound of $\Omega(\log^{1-\epsilon} k)$ can be extended to the algorithms running in quasi-polynomial time, i.e., the problem is still hard to approximate within $o(\log k)$ even if we allow quasi-polynomial time algorithms. This part is omitted in this version and can be found in the full version [25].

Main Result 4. For the problems of Robust s-t Path, weighted independent set, and spanning tree under the *maximin criteria*, it is NP-Hard to determine whether their instances have zero-cost optimal solutions or not. This implies that these problems do not admit any polynomial time α-approximate algorithm unless P = NP, where α is an arbitrary function of the input.

The paper also considers the *maximin criteria*, where the goal is to maximize the minimum cost among all agents. By observing that the classic algorithms (e.g., Dijkstra's algorithm) that work for the shortest path problem on DAGs (directed acyclic graphs) also work for the longest path problem on DAGs, one might expect that the maximin criteria is also a candidate objective to investigate the robustness of the s-t path problem, i.e., finding an s-t path $\mathcal{P}$ such that $\min_{i\in[k]} c_i(\mathcal{P})$ is maximized. We demonstrate this is not the case by providing a strong lower bound for the problem under the maximin criteria. Our reduction builds on a variant of the set cover problem. Employing a similar basic idea of the reduction, we also show that the maximin weighted independent set problem on trees or interval graphs is not approximable. This constitutes a strong lower bound for this problem, while the previous works [24, 28] only show the NP-Hardness. Our reduction idea can further be extended to the maximin spanning tree problem, which implies that the robust spanning tree problem is also not approximable under the maximin objective. This part is completely omitted in this version and can be found in the full version [25].

1.2 Other Related Works

Robust Minimax Combinatorial Optimization. Robust minimax optimization under different combinatorial structures has been extensively studied in the past three decades. See [3, 22] for a survey. Many problems that are polynomial-time solvable in the normal setting are shown to be NP-hard in the robust minimax setting: spanning trees, s-t cuts, and perfect matching on bipartite graphs [22]. Besides these fundamental problems, the minimax submodular ranking problem was studied in [10] very recently. For the minimax spanning tree, a $O(\log k/\log\log k)$-approximation algorithm is known [9], which is almost tight by the lower bound of $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) stated in [21]. The problem of minimax perfect matching has a lower bound of $\Omega(\log^{1-\epsilon} k)$ (for any $\epsilon > 0$) [20], while the best upper bound so far is still $O(k)$ which is trivial. In the case where k is a constant, fully polynomial time approximation schemes are known for spanning trees, perfect matching, knapsacks, and s-t paths [3, 4, 22, 29].

Multiobjective s-t Path. Finding an s-t path is a fundamental problem in multi-objective optimization [15]. An Excellent survey of multiobjective combinatorial optimization, including multiobjective s-t path, can be found in [11]. Typically, we are given a directed graph $G := (V, E)$. Each edge $e \in E$ has a positive cost vector $\mathbf{c}(e) := (c_1(e), \ldots, c_k(e))$. For every s-t path $\mathcal{P} \subseteq E$, we have a cost vector $\mathbf{c}(\mathcal{P}) = (c_1(\mathcal{P}), \ldots, c_k(\mathcal{P}))$ with $c_i(\mathcal{P}) = \sum_{e\in\mathcal{P}} c_i(e)$. The goal is to compute an s-t path $\mathcal{P}$ such that $\mathcal{P}$ is *Pareto optimal*. An s-t path is called Pareto optimal if there is no other s-t path that makes one objective better off without making another worse off. Not surprisingly, this problem has been shown to be NP-hard even if the cost vector only has two coordinates [32] in which the problem is called the bi-objective s-t path minimization problem. Bi-objective s-t path minimization has also been studied extensively [14, 31], in which researchers mainly focus on the exact algorithms with exponential running time. In addition, a fully polynomial time approximation scheme is proposed by [29].

Fair Allocation with Public Goods. By observing the minimax objective as a fairness criterion, our problem shares some similarities with the problem of public goods, which was first used to distinguish the previous private goods by Conitzer et al. [12] in the field of fair division. Specifically, there is a multiagent system and different agents hold different opinions about the same goods. And, they aim to select a feasible set of goods to satisfy the various

fairness notions, such as propositional share or its generalization [12, 16]. In [16], they study some constraints of goods, i.e., the selected goods must form a matching or matroid. The minimax criterion is quite different from other fairness measures in the fair division field, which leads to different techniques.

1.3 Roadmap

Section 2 and Section 3 present results on series-parallel graphs and bounded-treewidth graphs, respectively. Subsequently, in Section 4, the general graph case is considered. Section 5 finally concludes the paper. Due to space constraints, all the results on the maximin criteria are deferred to the full version of the paper.Note that we focus on high-level descriptions of our methods in the main body. Some formal descriptions and proofs (including lemma statements) can be found in the corresponding appendices.

2 Series-Parallel Graphs

In this section, we show that there is a randomized algorithm that achieves $O(H \log k)$ approximation for series-parallel graphs, where H is the height of the series-parallel graph's decomposition tree; its meaning will be clear later. The algorithm can be viewed as a warm-up example for the general graph, as the algorithm for the general graph follows a similar algorithmic framework. Formally, we shall show the following theorem (Theorem 1) in this section. We only present the LP formulation and the complete algorithm due to space limits. All proofs can be found in the paper's full version.

▶ **Theorem 1.** *Given any series-parallel graph G, there is a polynomial time algorithm that returns a $O(H \log k)$-approximation solution with probability at least $1 - (\frac{1}{k} + \frac{1}{kH})$ for robust s-t path, where H is the height of G's decomposition tree and k is the number of agents.*

In Section 2.1, we give the basic concepts and properties of the series-parallel graphs, which we will use later to build our linear programming formulation. In Section 2.2, we formally present our LP formulation. We give the complete and rounding algorithm in Section 2.3. Finally, we show the analysis in Section 2.4.

2.1 Basic Concepts

▶ **Definition 2** (Series-Parallel Graph). *A directed graph $G := (V, E, s, t)$ with source s and sink t is called a* series-parallel graph, *if it contains a single edge from s to t, or it can be built inductively using the following* series *and* parallel composition *operations. The series composition of two-terminal graphs $G_1 := (V_1, E_1, s_1, t_1)$ and $G_2 := (V_2, E_2, s_2, t_2)$ is to identify t_1 and s_2, and let s_1 and t_2 be the new source and sink in the resulting graph. The parallel composition of two-terminal graphs $G_1 := (V_1, E_1, s_1, t_1)$ and $G_2 := (V_2, E_2, s_2, t_2)$ is to identify s_1 with s_2 and t_1 with t_2 respectively, and let $s_1 = s_2$ and $t_1 = t_2$ be the new source and sink.*

A series-parallel graph can be represented in a natural way by a tree structure that describes how to assemble some small graphs into a final series-parallel graph through series and parallel composition. Such a tree structure is commonly called the *decomposition tree* of the series-parallel graph in the literature [33]. Formally, a decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ of a series-parallel graph $G := (V, E)$ is a tree such that (i) each leaf node $\mathbf{u} \in \mathbf{V}$ corresponds an edge in E; (ii) each internal node is either a series or parallel node; (iii) the child nodes of a parallel (resp. series) node must be leaf nodes or series (resp. parallel) nodes. The series

(resp. parallel) node corresponds to the series (resp. parallel) composition, and they are used to indicate how to merge the small subgraphs of its child nodes. The subgraph $G_{\mathbf{u}}$ of a node $\mathbf{u}$ is a subgraph of G such that $G_{\mathbf{u}}$ only contains those edges corresponding to the leaf nodes in the subtree rooted at $\mathbf{u}$. Let H be the height of $\mathbf{T}$. Given a series-parallel graph, it is known that its decomposition tree can be built in linear time by the standard series-parallel graph recognition algorithm [33]. An example can be found in Figure 1.

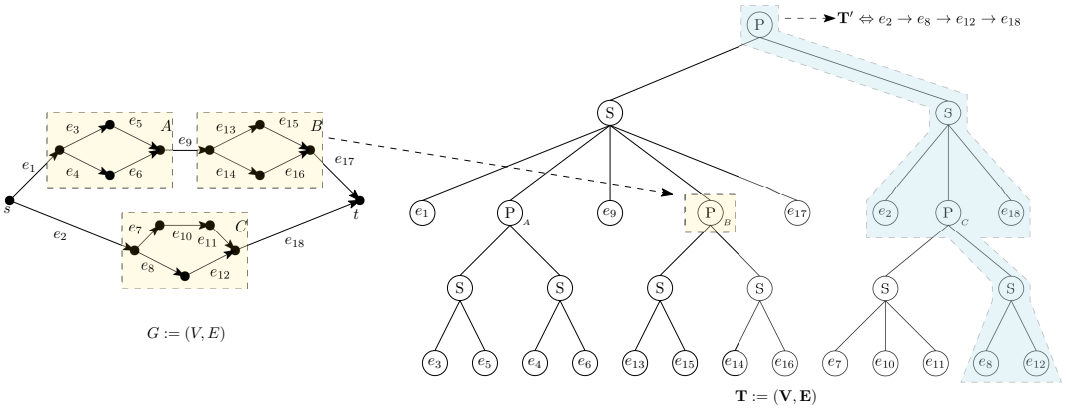


Figure 1 An example of the decomposition tree of a series-parallel graph. The series-parallel graph $G := (V, E)$ is shown on the left and its decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ is shown on the right. Each leaf node in $\mathbf{T}$ corresponds to an edge in E. Each internal node is either a series node S or a parallel node P. And it indicates how to merge the child nodes' subgraph. For example, consider the S node and its two child nodes e_{13} and e_{15}. Then, the subgraph of this S node is $e_{13} \to e_{15}$ which merges its two child nodes' subgraph via the series composition. And also, the subgraph of this S node's parent corresponds to the subgraph B in G, which merges $e_{13} \to e_{15}$ and $e_{14} \to e_{16}$ via the parallel composition. An s-t path corresponds to a feasible subtree (Definition 3). For example, the feasible subtree $\mathbf{T}'$ can be converted to an s-t path $e_2 \to e_8 \to e_{12} \to e_{18}$.

We remark that the children of a parallel node are unordered, and for a series node, the children should be considered as ordered. However, for the s-t path problem, the order is not important, as permuting the children of a series node will lead to an equivalent instance.

We aim to give a linear program based on the decomposition tree $\mathbf{T}$. Clearly, not all subtrees of $\mathbf{T}$ correspond to an s-t path of G. In the following, we introduce the concept of a special subtree of $\mathbf{T}$ called *feasible subtree* (Definition 3), which is able to be converted into an s-t path.

▶ **Definition 3** (Feasible Subtree). *A subtree $\mathbf{T}'$ of $\mathbf{T}$ is called a feasible subtree if and only if (i) $\mathbf{T}'$ includes the root node of $\mathbf{T}$; (ii) for every series node $\mathbf{s}$ in $\mathbf{T}'$, $\mathbf{T}'$ includes all child nodes of $\mathbf{s}$; (iii) for every parallel node $\mathbf{p}$ in $\mathbf{T}'$, $\mathbf{T}'$ includes exactly one child node of $\mathbf{p}$.*

It now remains to define a cost function $f_i : 2^{\mathbf{V}} \to \mathbb{R}_{\geq 0}$ according to the cost function c_i. Since each edge corresponds to a unique leaf node in $\mathbf{T}$, it is easy to define f_i by c_i: for each $\mathbf{v} \in \mathbf{V}$, $f_i(\mathbf{v}) := c_i(e)$ if node $\mathbf{v}$ corresponds to some edge e; otherwise $f_i(\mathbf{v}) := 0$. Formally, we have the following simple observation (Observation 4).

▶ **Observation 4.** *Given any series-parallel graph $G := (V, E)$ and its decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$, fix an arbitrary agent $i \in [k]$, any s-t path $\mathcal{P}$ of G with the cost $c_i(\mathcal{P})$ corresponds to a feasible subtree $\mathbf{T}'$ of $\mathbf{T}$ with the cost $f_i(\mathbf{T}')$ such that $f_i(\mathbf{T}') = c_i(\mathcal{P})$ and vice versa.*

2.2 LP Formulation

Given any series-parallel graph $G := (V, E)$, we first employ the standard doubling technique to enhance the linear program. Given a guess of the optimal objective value GS, we discard those edges e such that there exists an agent $i \in [k]$ with $c_i(e) > \mathsf{GS}$. Clearly, these discarded edges cannot belong to the optimal solution. Then, we run the series-parallel graph recognition algorithm [33] to construct a decomposition tree $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ of the series-parallel graph. See Algorithm 1 for the complete description.

The linear program is shown in (Tree-LP). For an internal node $\mathbf{v} \in \mathbf{V}$, use $\mathsf{child}(\mathbf{v})$ and $\Lambda(\mathbf{v})$ to denote its children and descendants in the tree, respectively. Let $P(\mathbf{T})$ and $S(\mathbf{T})$ be the set of parallel and series nodes in $\mathbf{T}$. For each node $\mathbf{v}$, $x_{\mathbf{v}}$ is a relaxed indicator variable denoting whether $\mathbf{v}$ is selected or not. The three constraints (2), (3) and (4) correspond to the three conditions stated in Definition 3 respectively, in order to ensure that the solution is a feasible subtree. The first constraint type (1) is a bit subtle, and it is the key that allows us to surpass the pessimistic $\Omega(k)$ and $\Omega(\sqrt{n})$ integrality gap. In these constraints, $\sum_{\mathbf{u}\in\Lambda(\mathbf{v})} x_{\mathbf{u}} \cdot f_i(\mathbf{u})$ denotes the cost of the selected subtree rooted at $\mathbf{v}$ with respect to agent i. Thus, when $\mathbf{v} = \mathbf{r}$, the constraint implies that for any agent, the total cost of all the selected nodes must be at most $x_{\mathbf{r}} \cdot \mathrm{OPT} = \mathrm{OPT}$. For the cases that $\mathbf{v} \neq \mathbf{r}$, these constraints do not affect the feasible region of integer solutions since $x_{\mathbf{v}}$ is either 1 or 0, but they can reduce the fractional solution's feasible region dramatically by restricting the contribution of each subtree $\Lambda(\mathbf{v})$. A more detailed discussion is given in the full version of the paper.

(Tree-LP)

$$\sum_{\mathbf{u}\in\Lambda(\mathbf{v})} x_{\mathbf{u}} \cdot f_i(\mathbf{u}) \le x_{\mathbf{v}} \cdot \mathsf{GS}, \qquad \forall i \in [k], \forall \mathbf{v} \in \mathbf{V} \tag{1}$$

$$x_{\mathbf{r}} = 1, \tag{2}$$

$$\sum_{\mathbf{u}\in\mathsf{child}(\mathbf{v})} x_{\mathbf{u}} = x_{\mathbf{v}}, \qquad \forall \mathbf{v} \in P(\mathbf{T}) \tag{3}$$

$$x_{\mathbf{u}} = x_{\mathbf{v}}, \qquad \forall \mathbf{v} \in S(\mathbf{T}), \mathbf{u} \in \mathsf{child}(\mathbf{v}) \tag{4}$$

$$x_{\mathbf{v}} \ge 0, \qquad \forall \mathbf{v} \in \mathbf{V} \tag{5}$$

2.3 Algorithms

This section formally describes the complete algorithm (Algorithm 1) for series-parallel graphs. The main algorithm mainly consists of two steps: the doubling step (lines 2-13 of Algorithm 1) and the rounding algorithm (Algorithm 2). After finishing the doubling step, we obtain a fractional solution x^* with the value of GS that is close to the optimal solution OPT (Observation 5). Then, we shall employ a dependent rounding algorithm to obtain a feasible subtree based on x^*. This dependent randomized rounding algorithm selects nodes level by level, starting from the top of $\mathbf{T}$ and proceeding downwards. For parallel nodes, the algorithm selects one of its child nodes with a probability determined by the optimal fractional solution x^*. For series nodes, the algorithm selects all of its child nodes with a probability of 1, ensuring that the resulting subtree is always feasible. A formal description of the algorithm can be found in Algorithm 2.

2.4 Analysis

This section analyzes the performance of our algorithm. We start by describing a simple observation (Observation 5). Let $\mathbf{T}'$ be the subtree returned by Algorithm 2. Recall that H is the height of $\mathbf{T}$.

Algorithm 1 The Complete Algorithm for Series-Parallel Graphs.

Input: A series-parallel graph $G := (V, E)$ with k cost functions $c_i : 2^E \to \mathbb{R}_{\geq 0}, i \in [k]$.
Output: An s-t path $\mathcal{P} \subseteq E$.

1: $\texttt{flag} \leftarrow \texttt{true}$; $\mathsf{GS} \leftarrow \max_{i\in[k]} \sum_{e\in E} c_i(e)$.
2: **while** $\texttt{flag} = \texttt{true}$ **do**
3: $\quad E' \leftarrow \{\, e \in E \mid \exists i \in [k] \text{ s.t. } c_i(e) > \mathsf{GS} \,\}$.
4: $\quad E \leftarrow E \setminus E'$; $G \leftarrow (V, E')$.
5: $\quad$ Compute the tree decomposition $\mathbf{T} := (\mathbf{V}, \mathbf{E})$ of G by [33].
6: $\quad$ Solve the linear program (Tree-LP).
7: $\quad$ **if** (Tree-LP) has a feasible solution **then**
8: $\quad\quad$ Let x^* be a feasible solution to (Tree-LP).
9: $\quad\quad \mathsf{GS} \leftarrow \frac{\mathsf{GS}}{2}$.
10: $\quad$ **else**
11: $\quad\quad \texttt{flag} \leftarrow \texttt{false}$.
12: $\quad$ **end if**
13: **end while**
14: Run Algorithm 2 with the optimal solution x^* to obtain a feasible subtree $\mathbf{T}'$ of $\mathbf{T}$.
15: Convert $\mathbf{T}'$ into an s-t path $\mathcal{P}$.
16: **return** $\mathcal{P}$.

▶ **Observation 5.** *Let GS^* be the guessing value at the beginning of the last round of the while-loop (lines 2-13 of Algorithm 1). Then, we have $\mathsf{GS}^* \leq 2 \cdot \mathrm{OPT}$*[2].

To show that the approximation ratio is $O(H \log k)$ with a constant probability, a natural step is to first bound the expectation of our solution. We first state some intuition. According to the description of the rounding scheme, it is easy to see that for each agent i,

$$\mathbb{E}[f_i(\mathbf{T}')] = \sum_{e\in E} c_i(e) \Pr[e \in \mathbf{T}'] = \mathsf{GS}^*.$$

Then by Markov inequality, we have for each agent i,

$$\Pr[f_i(\mathbf{T}') \geq H \log k \cdot \mathsf{GS}^*] \leq \frac{1}{H \log k}.$$

However, the above inequality is not sufficient because proving Theorem 1 needs to show that $\Pr[\forall i \in [k], f_i(\mathbf{T}') \geq H \log k \cdot \mathsf{GS}^*]$ is at most $\frac{1}{k} + \frac{1}{kH}$. To address this issue, we need to employ an analysis technique called *Moment Method*, which is widely used in the literature [13, 19]. More formally, we aim to show the following key lemma (Lemma 6); a similar proof can also be found in [13].

▶ **Lemma 6.** *For any agent $i \in [k]$, we have $\mathbb{E}[\exp\left(\ln(1 + \frac{1}{2H}) \cdot f_i(\mathbf{T}')\right)] \leq 1 + \frac{1}{H}$.*

Proof. We prove the theorem inductively. First, consider the case that $H = 1$, i.e., the decomposition tree $\mathbf{T}$ only contains a root $\mathbf{r}$. Since $x_{\mathbf{r}} = 1$, there is no randomness in selecting $\mathbf{T}'$. Thus, we have for any $z \geq 1$,

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] = z^{\frac{x_{\mathbf{r}} \cdot f_i(\mathbf{r})}{\mathsf{GS}^*}} \leq z,$$

where the last inequality uses constraint (1) in (Tree-LP).

[2] One can get a more accurate lower bound of the optimal solution (e.g., $\mathsf{GS}^* \leq (1+\epsilon) \cdot \mathrm{OPT}$ for any $\epsilon > 0$) by the standard binary search technique.

Algorithm 2 Dependent Randomized Rounding.

Input: A tree structure $\mathbf{T}(\mathbf{V}, \mathbf{E})$ rooted at $\mathbf{r}$; a fractional solution $x^* \in [0,1]^{|\mathbf{V}|}$.
Output: A feasible subtree $\mathbf{T}'$.

Initially, set $\mathbf{T}' \leftarrow \emptyset$ and a vertex queue $\mathcal{Q} \leftarrow \{\mathbf{r}\}$.
while $\mathcal{Q} \neq \emptyset$ **do**
  Use $\mathbf{v}$ to represent the front element of $\mathcal{Q}$.
  $\mathbf{T}' \leftarrow \mathbf{T}' \cup \{\mathbf{v}\}$, $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{\mathbf{v}\}$.
  if $\mathbf{v}$ is a parallel node **then**
    Pick one node $\mathbf{u} \in \mathsf{child}(\mathbf{v})$ randomly such that $\mathbf{u}$ is chosen with probability $\frac{x_{\mathbf{u}}}{x_{\mathbf{v}}}$.
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathbf{u}\}$.
  end if
  if $\mathbf{v}$ is a series node **then**
    for each $\mathbf{u} \in \mathsf{child}(\mathbf{v})$ **do**
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathbf{u}\}$.
    end for
  end if
end while
return $\mathbf{T}'$.

To streamline the analysis, we continue considering the case that $H = 2$. We further distinguish two subcases: (i) root $\mathbf{r}$ is a parallel node; (ii) root $\mathbf{r}$ is a series node. For the first subcase, Algorithm 2 selects exactly one child $\mathbf{v} \in \mathsf{child}(\mathbf{r})$ with probability $x_{\mathbf{v}}/x_{\mathbf{r}} = x_{\mathbf{v}}$. According to the law of total expectation and only leaves in $\mathbf{T}$ have non-zero costs, we have

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] = \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} \Pr[\mathbf{v} \in \mathbf{T}'] \cdot \mathbb{E}\left[z^{\frac{f_i(\Lambda'(\mathbf{v}))}{\mathsf{GS}^*}} \middle| \mathbf{v} \in \mathbf{T}'\right],$$

where $\Lambda'(\mathbf{v}) := \Lambda(\mathbf{v}) \cap \mathbf{T}'$. Observing that once conditioned on $\mathbf{v} \in \mathbf{T}'$, the conclusion for the $H = 1$ case can be used to bound the expectation, we have

$$\begin{aligned}
\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] &= \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} \Pr[\mathbf{v} \in \mathbf{T}'] \cdot \mathbb{E}\left[z^{\frac{f_i(\Lambda'(\mathbf{v}))}{\mathsf{GS}^*}} \middle| \mathbf{v} \in \mathbf{T}'\right], \\
&= \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}} \cdot z^{\frac{f_i(\mathbf{v})}{\mathsf{GS}^*}} \\
&\leq \sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}} \cdot \left(1 + (z-1) \cdot \frac{f_i(\mathbf{v})}{\mathsf{GS}^*}\right) \\
&\qquad\qquad \text{(Constraint (1) and } z^r \leq 1 + r(z-1) \forall z > 0, r \in [0,1]\text{)} \\
&= \left(\sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}}\right) + (z-1) \cdot \frac{\sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} x_{\mathbf{v}} \cdot f_i(\mathbf{v})}{\mathsf{GS}^*} \\
&= 1 + (z-1) \cdot \frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}} \cdot f_i(\mathbf{v})}{\mathsf{GS}^*} \qquad \text{(Constraint (3))} \\
&\leq e^{(z-1)\cdot\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}}. \qquad (1 + x \leq e^x)
\end{aligned}$$

For the second subcase, according to Constraint (4), we have $x_{\mathbf{v}} = x_{\mathbf{r}} = 1$ for each $x_{\mathbf{v}} \in \mathsf{child}(\mathbf{r})$, and therefore,

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] = \mathbb{E}\left[z^{\frac{\sum_{\mathbf{v}\in\mathsf{child}(\mathbf{r})} f_i(\Lambda'(\mathbf{v}))}{\mathsf{GS}^*}}\right] \leq z^{\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}}.$$

Since $z \leq e^{z-1}$, combing the two subcases, we have that when $H = 2$,

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] \leq (e^{(z-1)})^{\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}}.$$

The above inequality shows that as the height increases by 1, the upper bound of the expectation grows exponentially. Furthermore, when the height increases from 1 to H, we can obtain a sequence $z_1 = z, z_2, \ldots, z_H$, where $z_h = e^{z_{h-1}-1}$ for each $h > 1$, and have

$$\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right] \leq z_h^{\frac{\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}}\cdot f_i(\mathbf{v})}{\mathsf{GS}^*}},$$

for any $\mathbf{T}$ with height h by almost the same analysis as above. Due to Constraint (1), we have $\sum_{\mathbf{v}\in\Lambda(\mathbf{r})} x_{\mathbf{v}} \cdot f_i(\mathbf{v}) \leq \mathsf{GS}^*$, and thus, $\mathbb{E}\left[z^{\frac{f_i(\mathbf{T}')}{\mathsf{GS}^*}}\right]$ is at most z_h.

Finally, to obtain the claimed upper bound, we set $z = 1 + \frac{1}{2H}$ such that $z_h \leq 1 + \frac{1}{H}$, and complete the proof. ◀

▶ **Lemma 7.** *Consider an arbitrary agent i, Algorithm 2 returns a feasible subtree $\mathbf{T}'$ such that $f_i(\mathbf{T}') \leq 4H \cdot \log k \cdot \mathsf{GS}$ with high probability, where H is the height of the decomposition tree of the series-parallel graph and GS is a guess of the optimal objective value such that the corresponding* (Tree-LP) *admits a feasible solution.*

Lemma 7 can be proved by Lemma 6 and Markov bound. Theorem 1 can be proved by Lemma 7 and union bound. All proofs are deferred to the full version of the paper.

3 Graphs with Bounded Treewidth

This section considers robust s-t path on graphs with bounded treewidth. We mainly show the following theorem. Noting that any series-parallel graph has a treewidth of 2, this result improves upon the above $O(H \log k)$ ratio for series-parallel graphs with large H.

▶ **Theorem 8.** *Given any directed graph G with treewidth $\mathsf{tw}(G) \leq \ell$, there is an algorithm that returns a $O(\log n \log k)$-approximate solution in poly$(n) \cdot n^{O(\ell^2)}$ time with probability at least $1 - (\frac{1}{k} + \frac{1}{k \log n})$ for robust s-t path, where n is the number of vertices and k is the number of agents.*

3.1 Algorithmic Framework

The basic idea of the algorithm is to reduce our problem to *the tree labeling problem* which was proposed by Dinitz et al. [13] very recently. In their paper, they provided a randomized algorithm for the tree label problem. Applying the algorithm to our reduced instance can obtain an s-t path whose expected cost with respect to each agent is bounded. Finally, we employ the concentration inequalities to show that with high probability, the returned path is a polylogarithmic approximation solution. To ensure the reduction's correctness, we also need to utilize some other tools. The complete description of our algorithm can be found in

the full version of the paper. Due to space limitations, in the main body, we only focus on the core of our algorithm – the reduction to tree labeling. Before introducing the reduction, it is necessary to restate the definition of tree labeling and the result proved in [13].

The Tree-labeling Problem. Consider a binary tree $\mathbf{T}(\mathbf{V}, \mathbf{E})$ rooted at $\mathbf{r} \in \mathbf{V}$. For each node $\mathbf{v} \in \mathbf{V}$, there is a finite set $L_{\mathbf{v}}$ of labels for $\mathbf{v}$. Let $L := \bigcup_{\mathbf{v}\in\mathbf{V}} L_{\mathbf{v}}$ be the set of all possible labels. The output is a label assignment $\mathcal{L} := (l_{\mathbf{v}} \in L_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ of the node set $\mathbf{V}$, that satisfies the consistency and cost constraints.

- **(Consistency Constraints)** For every internal node $\mathbf{v}$ of $\mathbf{T}$ with two children $\mathbf{u}$ and $\mathbf{w}$ ($\mathbf{u}$ or $\mathbf{w}$ is possibly empty), we are given a set $\Gamma(\mathbf{v}) \subseteq L_{\mathbf{u}} \times L_{\mathbf{v}} \times L_{\mathbf{w}}$. A valid labeling $\mathcal{L} = (l_{\mathbf{v}} \in L_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ must satisfy $(l_{\mathbf{u}}, l_{\mathbf{v}}, l_{\mathbf{w}}) \in \Gamma(\mathbf{v})$ for every internal node $\mathbf{v}$.
- **(Cost Constraints)** There are k additive cost functions $f_1, \ldots, f_k$ defined over the labels, i.e., for each $i \in [k]$, $f_i : L \to \mathbb{R}_{\geq 0}$. For each $i \in [k]$, a valid labeling $\mathcal{L}$ needs to satisfy $f_i(\mathcal{L}) := \sum_{\mathbf{v}\in\mathbf{V}} f_i(l_{\mathbf{v}}) \leq 1$.

A label assignment $\mathcal{L} := (l_{\mathbf{v}} \in L_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ is called *consistent* if it satisfies the consistency constraints; it is *valid* if it satisfies both the consistency and cost constraints. Let H be the height of $\mathbf{T}$ and let $\Delta := \max_{\mathbf{v}\in\mathbf{V}} |L_{\mathbf{v}}|$ be the maximum size of any label set. Let n be the number of nodes in $\mathbf{T}$. In [13], they show the following result.

▶ **Lemma 9** ([13]). *Given a tree labeling instance such that the instance admits a valid label assignment. There is a randomized algorithm that in time poly$(n) \cdot \Delta^{O(H)}$ outputs a consistent label assignment $\mathcal{L}$ such that for every $i \in [k]$, we have* $\mathbb{E}\left[\exp\left(\ln(1 + \frac{1}{2H}) \cdot f_i(\mathcal{L})\right)\right] \leq 1 + \frac{1}{H}$.

3.2 Reduction Intuition

In this section, we give some intuition of our reduction. The formal description and an example can be found in the next section (Section 3.3). Given any directed graph with bounded treewidth, we aim to construct a tree-labeling instance such that the solution to the constructed tree-labeling instance can be converted into an s-t path of the original graph with some cost-preserved property. Note that the treewidth decomposition $\mathbf{T}(\mathbf{V}, \mathbf{E})$ of any input graph G can be computed efficiently [8]. We directly let the treewidth decomposition $\mathbf{T}$ be the binary tree in the reduced tree-labeling instance. The following shows how to construct the labels and the corresponding constraints such that a feasible label assignment can be successfully transformed into an s-t path.

Label Construction. For a graph's treewidth decomposition, each node $\mathbf{v} \in \mathbf{T}$ corresponds to a node subset $X(\mathbf{v})$ of the original graph. Each edge in the original graph is guaranteed to be covered by some $X(\mathbf{v})$, which is the completeness property of a tree decomposition. Use $C(\mathbf{v})$ to denote the edges covered by node $\mathbf{v} \in \mathbf{T}$. Without loss of generality, we can assume that s and t are included in any node $\mathbf{v} \in \mathbf{T}$ and each edge is assigned to a unique $C(\mathbf{v})$. The label of a node $\mathbf{v}$ is a vector of $|C(\mathbf{v})| + |X(\mathbf{v})| \cdot (|X(\mathbf{v})| - 1)$ dimensions, where the first $|C(\mathbf{v})|$ dimensions correspond to the edges covered by it and the last $|X(\mathbf{v})| \cdot (|X(\mathbf{v})| - 1)$ dimensions correspond to all the vertex pairs in $X(\mathbf{v})$. The intuition is the following. To ensure that a feasible label assignment can be translated to an s-t path, we first need to assign a "choosing indicator" to each edge to imply whether the edge is selected or not. However, having the choosing indicators is not enough because we only know some edges have been picked, but cannot determine whether s and t are connected. Thus, we introduce a "connectivity indicator" for each vertex pair (a, b) in $X(\mathbf{v})$ to indicate whether a is connected to b by the selected edges covered in the subtree rooted at node $\mathbf{v}$.

Constraint Construction. The cost constraints are used to bound the total cost of the selected edges. They can be obtained easily by letting the normalized cost of the edges selected by each label assignment be the corresponding cost. For the consistency constraints, the purpose of designing them is to ensure that s is connected to t, and the connectivity indicators can truthfully reflect the connectivity of the subgraph formed by the selected edges. To achieve the former requirement, we define that a label assignment is feasible only if the connectivity indicator of (s,t) in the root $\mathbf{r}$ is $\mathtt{1}$; while for the latter requirement, the constraint construction is still natural, but we note that proving such "local[3]" constraints are able to capture the global connectivity is non-trivial. Consider an arbitrary node $\mathbf{u}$ and its two children $\mathbf{v}, \mathbf{w}$. We can construct a subgraph where the vertex set is $X(\mathbf{u}) \cup X(\mathbf{v}) \cup X(\mathbf{w})$. Add an edge (a,b) in the subgraph if the connectivity indicator of (a,b) is $\mathtt{1}$ in one of the two children or the choosing indicator of edge (a,b) in node $\mathbf{u}$ is $\mathtt{1}$. A label is feasible if the connectivity indicator of node $\mathbf{u}$ is consistent with the connectivity in this subgraph. Note that this subgraph may not contain all nodes that occur in the subtree rooted at $\mathbf{u}$. Thus, to prove the efficiency of these constraints, we further need to show that there does not exist a vertex pair in $\mathbf{u}$ that is connected by $\mathbf{u}$'s subtree but not connected in the above subgraph. We formally show this claim in the paper's full version. The proof heavily relies on the connectivity property of a tree decomposition. Briefly speaking, a graph's tree decomposition can guarantee that all nodes in the tree containing the same node in the original graph form a connected subtree. This nice property allows us to show that the connectivity between vertices can be continuously propagated between nodes in $\mathbf{T}$.

3.3 Tree-labeling Instance Construction

Given an arbitrary node $\mathbf{v} \in \mathbf{V}$, we also use $\mathbf{v}$ to denote the vertices included in node $\mathbf{v}$, i.e., $\mathbf{v} \subseteq V$. Let $E_{\mathbf{v}} \subseteq E$ be the set of edges such that, for any edge (a,b) in $E_{\mathbf{v}}$, node $\mathbf{v}$ is the highest node that contains (a,b). Note that an edge (a,b) may be included in more than one node in $\mathbf{T}$ but the highest node that includes (a,b) is unique. For any node $\mathbf{v} \in \mathbf{V}$, we have two types of labels: choosing label and connectivity label. For each edge (a,b) (or e) in $E_{\mathbf{v}}$, the choosing label $\mathtt{chng}(a,b) = \mathtt{1}$ or (or $\mathtt{chng}(e) = \mathtt{1}$) indicates that the edge is chosen in current label assignment; otherwise, the edge is not chosen. For each pair of vertices (p,q) in $\mathbf{v}$, the connectivity label $\mathtt{conn}(p,q) = \mathtt{1}$ indicates that there is a $\mathcal{P} \subseteq E$ path from p to q such that every edge in $\mathcal{P}$ is chosen in some nodes of the subtree rooted at node $\mathbf{v}$; otherwise, p and q are not connected. Note that $\mathtt{conn}(p,q)$ and $\mathtt{conn}(q,p)$ are two different labels since G is a directed graph. Let $L_{\mathbf{v}}$ be the set of all possible labels and $l_{\mathbf{v}} \in L_{\mathbf{v}}$ be a specific label of node $\mathbf{v}$. We remark that the size of $L_{\mathbf{v}}$ is related to the treewidth of G. Since the treewidth of G is a constant, $|L_{\mathbf{v}}|$ is also constant. See the proof of Theorem 8 for details.

To ensure the feasibility of label assignments for obtaining an s-t path in $\mathbf{T}$, arbitrarily picking labels for each node is not a viable solution. Instead, we define a local constraint that applies to every adjacent set of three nodes $\mathbf{u}, \mathbf{v}, \mathbf{w}$ in $\mathbf{T}$, where $\mathbf{u}$ and $\mathbf{w}$ are the child nodes of $\mathbf{v}$. The purpose of this local constraint is to guarantee that all label assignments are capable of producing an s-t path. For every node $\mathbf{v}$ and its two children $\mathbf{u}$ and $\mathbf{w}$, let $\mathtt{CP}(\mathbf{v}) := L_{\mathbf{u}} \times L_{\mathbf{v}} \times L_{\mathbf{w}}$ be the set of all possible label combinations of these three nodes, i.e., $\mathtt{CP}(\mathbf{v})$ is the Cartesian product of $L_{\mathbf{u}}, L_{\mathbf{v}}, L_{\mathbf{w}}$. Note that $\mathbf{u}$ or $\mathbf{w}$ may not exist. In this case, we refer to $\mathbf{u}$ or $\mathbf{w}$ as empty nodes and $\mathtt{CP}(\mathbf{v})$ is defined as the $L_{\mathbf{v}} \times L_{\mathbf{w}}$ (or $L_{\mathbf{u}} \times L_{\mathbf{v}}$ or $L_{\mathbf{v}}$).

[3] Consistency constraints in the tree-labeling problem are "local" constraints because the feasibility of a node $\mathbf{u}$'s label is only influenced by its child nodes.

▶ **Definition 10** (Feasible Label Assignment). *A label assignment $\mathcal{L} := (l_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$ is a feasible label assignment if, for each $\mathbf{v} \in \mathbf{V}$ and its two children $\mathbf{u}$ and $\mathbf{w}$ ($\mathbf{u}$ and $\mathbf{w}$ maybe empty nodes), $l_{\mathbf{v}} \in \texttt{CP}(\mathbf{v})$ satisfies the following three constraints:*

(C1) **(Choosing Constraints)** *For each edge $(a, b) \in E_{\mathbf{v}}$, if (a, b) is chosen, then a and b are connected and vice versa. Namely, $\texttt{chng}(a, b) = \texttt{1}$ if and only if $\texttt{conn}(a, b) = \texttt{1}$.*

(C2) **(Connectivity Constraints)** *For each vertex pair (p, q) in $\mathbf{v}$, vertex p and vertex q are connected (i.e. $\texttt{conn}(p, q) = \texttt{1}$) if and only if the following statement is true: there is a vertex sequence $(p, v_1, \ldots, v_d, q)$ such that every two adjacent vertices (a, b) in the sequence are connected in some nodes in $\mathbf{u}, \mathbf{v}, \mathbf{w}$, i.e., $\texttt{conn}(a, b) = \texttt{1}$ in some nodes in $\mathbf{u}, \mathbf{v}, \mathbf{w}$ for each pair of adjacent vertices.*

(C3) **(Feasibility Constraints)** *If $\mathbf{v}$ is the root of $\mathbf{T}$, then the source s and sink t are connected, i.e., $\texttt{conn}(s, t) = \texttt{1}$ must be true in the root.*

Given a feasible label assignment $\mathcal{L}$, an edge $(p, q) \in E$ is *chosen* by $\mathcal{L}$ if (p, q)'s choosing label is 1 in $\mathcal{L}$. (C1) defines the connectivity of each edge in E. If an edge (a, b) is chosen by a label assignment, then vertex a and b are connected. (C2) is the most important constraint which defines the connectivity of each pair of vertices in $\mathbf{v}$. In the case where $\mathbf{v}$ is a leaf node, $\mathbf{u}$ and $\mathbf{w}$ would be empty nodes and thus this constraint is equivalent to (C1). In the case where $\mathbf{v}$ is not a leaf node, an arbitrary vertices pair (a, b) in $\mathbf{v}$ are connected if the connected segments in $\mathbf{u}, \mathbf{v}, \mathbf{w}$ can be merged into a path from a to b. In Lemma 11, we show that such a local constraint is sufficient to describe the connectivity of vertex a and b in the subtree rooted at $\mathbf{v}$. (C3) ensures that a feasible label assignment must contain an s-t path, i.e., source s and sink t are connected.

Consider an arbitrary feasible label assignment $\mathcal{L}$, then $\mathcal{L}$ has the following crucial property (Lemma 11) by our definition. We shall use this property later to show that any feasible label assignment can be converted into an s-t path of the original graph. It is worth noting that an s-t path corresponds to a unique feasible label assignment, but a feasible label assignment may contain multiple s-t paths. An example is shown in Figure 2.

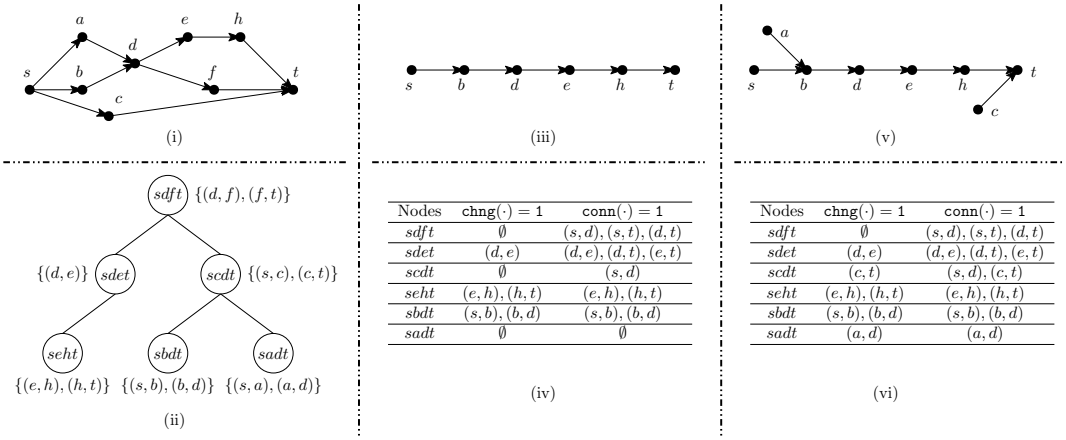
▶ **Lemma 11.** *Given an arbitrary feasible label assignment $\mathcal{L} := (l_{\mathbf{v}})_{\mathbf{v}\in\mathbf{V}}$, consider an arbitrary node $\mathbf{v} \in \mathbf{V}$. For any vertices pair (p, q) in $\mathbf{v}$, there is a path $\mathcal{P} \subseteq E$ from p to q such that every edge in $\mathcal{P}$ is chosen in some nodes in the subtree rooted at $\mathbf{v}$ if and only if (a, b) has a connectivity label of 1 in $\mathbf{v}$.*

To complete the instance construction, we also need to define an appropriate label cost function for each node in $\mathbf{T}$. It shall be used to connect the cost of our problem to the tree-labeling problem. This part, together with the proofs of Lemma 11 and Theorem 8, are deferred to the paper's full version.

4 General Graphs

We shall follow the same algorithmic framework stated in Section 2 and show the following result (Theorem 12). Namely, we first construct a tree structure and set up a linear program based on the tree. Then, we employ the same rounding algorithm (Algorithm 2) to obtain a feasible subtree and convert it back to an s-t path in the original graph. The following states some intuition to construct such a tree. We defer the formal descriptions of the construction, the LP formulation, the algorithm, and the analysis to the full version of the paper.

▶ **Theorem 12.** *Given any directed graph G, there is an algorithm that returns a $O(\log n \log k)$-approximate solution in $poly(n) \cdot n^{O(\log n)}$ time with probability at least $1 - (\frac{1}{k} + \frac{1}{k \log n})$ for robust s-t path, where n is the number of vertices and k is the number of agents.*



Nodes	chng(·) = 1	conn(·) = 1
sdft	∅	$(s,d),(s,t),(d,t)$
sdet	(d,e)	$(d,e),(d,t),(e,t)$
scdt	∅	(s,d)
seht	$(e,h),(h,t)$	$(e,h),(h,t)$
sbdt	$(s,b),(b,d)$	$(s,b),(b,d)$
sadt	∅	∅

Nodes	chng(·) = 1	conn(·) = 1
sdft	∅	$(s,d),(s,t),(d,t)$
sdet	(d,e)	$(d,e),(d,t),(e,t)$
scdt	(c,t)	$(s,d),(c,t)$
seht	$(e,h),(h,t)$	$(e,h),(h,t)$
sbdt	$(s,b),(b,d)$	$(s,b),(b,d)$
sadt	(a,d)	(a,d)

Figure 2 An example of the reduction. The subfigure (i) is the given directed graph. Then, we compute a tree decomposition with the logarithmic depth and add s and t to all nodes, which is shown in subfigure (ii). The edge set next to each node in subfigure (ii) is its corresponding $E_{\mathbf{v}}$. For example, the edge set $E_{\mathbf{r}}$ for root node $\mathbf{r}$ is $\{(d,f),(f,t)\}$ since $\mathbf{r}$ is the highest node that contains edge (d,f) and (f,t). Subfigure (iii) is an s-t path of the given directed graph and subfigure (iv) is the corresponding label assignment of the s-t path in (iii) in which we only list these labels with the value of `1`. The complete label of each node is obtained by merging these single labels, e.g., for the root $\mathbf{r}$, $l_{\mathbf{r}}$ consists of 14 bits (2 choosing labels and 12 connectivity labels). In these 14 bits, only $\texttt{conn}(s,d), \texttt{conn}(s,t), \texttt{conn}(d,t)$ has a value of `1` and all the remaining 11 labels have a value of `0`. Subfigure (v) shows another example and subfigure (vi) is its corresponding label assignment.

In Section 2, we use a tree-based linear program to break through the $\Omega(k)$ and $\Omega(\sqrt{n})$ integrality gap of the flow LP. However, on general graphs, we no longer have such a natural tree structure as in series-parallel graphs.To still obtain a polylogarithmic approximation, we construct a decision-tree-type metatree that maps every s-t path in the graph to a corresponding subtree in the metatree.

Metatree Construction Intuition. The basic idea of the meta tree construction is to iteratively guess the possible middle vertex of an s-t path. There are at most n possibilities for this middle vertex. Once we determine the middle vertex of the s-t path, say it is a, the whole path can be partitioned into two subpaths – path s-a and path a-t. We then recur on the obtained subpaths till level $O(\log n)$; the sufficiency of $O(\log n)$'s levels will be clear later. This process gives us a natural tree structure $\mathbf{T}$ with $O(\log n)$ depth. We define two types of nodes in $\mathbf{T}$. The first node type is referred to as *splitting node*. Each splitting node corresponds to a (sub-)path. It has n children, where each child represents a choice of the (sub-)path's middle vertex. The algorithm needs to ensure that only one of these children can be selected. The second node type is called *merging node*. A merging node has to be a child of a splitting node in $\mathbf{T}$ and represents a scheme for selecting a middle vertex. Further, a merging node has two splitting nodes as its children, corresponding to the two obtained subpaths by this scheme. We can view such a node as being used to merge its two children (subpaths). The algorithm needs to ensure that both children are selected simultaneously. See the paper's full version for more details of the construction.

As one may observe, a splitting (resp. merging) node in our metatree plays the same role as a parallel (resp. series) node in the decomposition tree of the series-parallel graph. Thus, the LP for the general graph is similar to (Tree-LP) and we can still use the same rounding algorithm. Consider an s-t path $\mathcal{P}$ of length n. If we write $\mathcal{P}$ in the form of a binary tree by

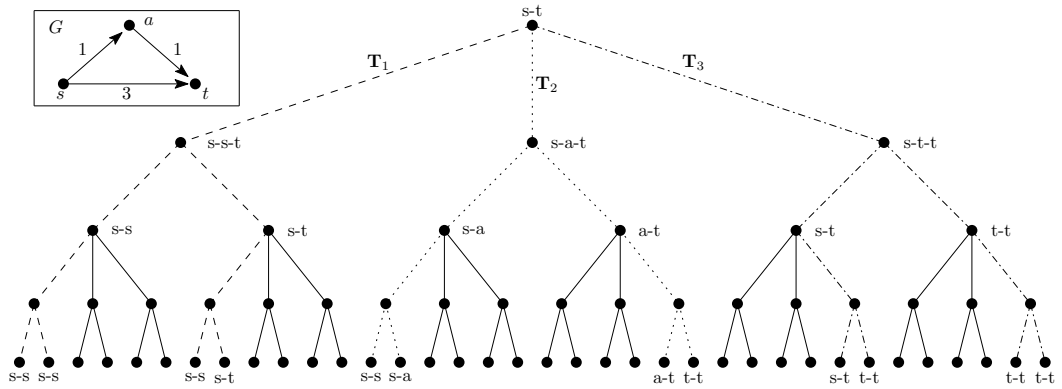


Figure 3 An example for the metatree $\mathbf{T}$ construction. The given directed graph G is shown in the up-left corner. Since there are three vertices in G, $\mathbf{T}$ will consist of five levels because the height of $\mathbf{T}$ is $2\lceil \log n \rceil + 1$. The dashed subtree $\mathbf{T}_1$ corresponds to the path $s \to t$ of G. The dotted subtree $\mathbf{T}_2$ represents the path $s \to a \to t$ of G. The dash-dotted subtree $\mathbf{T}_3$ also corresponds to the path $s \to t$.

guessing the middle vertex of each subpath, it will have at most $\lceil \log n \rceil$ levels. Thus, we can let the recursive tree $\mathbf{T}$ terminate at level $O(\log n)$, and therefore, its size is quasi-polynomial $O(n^{\log n})$ since each node has at most n children. From Section 2, we know that the rounding algorithm is able to find a $O(H \log k)$-approximate solution where H is the height of the tree. This also provides the intuition for the approximation ratio $O(\log n \log k)$ since the height of $\mathbf{T}$ is $O(\log n)$ (specifically, $H = 2\lceil \log n \rceil + 1$).

We now define a *feasible subtree* for $\mathbf{T}$. Recalling the feasible subtrees (Definition 3) on series-parallel graphs, unfortunately, we cannot use the same definition for general graphs. This is because not all leaf nodes in the constructed tree $\mathbf{T}$ correspond to edges in G. We refer to a subtree that satisfies three conditions in Definition 3 as a *consistent subtree*. To ensure that the subtree can be translated to an s-t path, one more condition is needed.

▶ **Definition 13** (Feasible Subtree for General Graphs). *A subtree $\mathbf{T}' \subseteq \mathbf{T}$ is called feasible if and only if (i) $\mathbf{T}'$ is consistent; (ii) each leaf node corresponds to either an edge or a single vertex in G.*

An example can be found in Figure 3. As one might be concerned, using a different definition of the feasible subtree may require a different LP formulation for general graphs, since a natural adaptation of (Tree-LP) can only find a consistent subtree. This issue can be fixed easily by directly disabling the infeasible leaf nodes in the linear program. See the paper's full version for more details of the LP formulation.

5 Conclusion

This paper considers the robust s-t path problem and proposes polylogarithmic approximation algorithms on different graph classes. For graphs with bounded treewidth, we obtain a $O(\log n \log k)$-approximate polynomial algorithm which partially answers the open question in [23]. For general graphs, we prove that there is a quasipolynomial algorithm that is $O(\log n \log k)$-approximate which leaves a logarithmic gap. Our approaches are based on a novel linear program that enables us to get rid of the $\Omega(k)$ and $\Omega(\sqrt{n})$ integrality gap from the natural linear program. We also investigate the robustness of the s-t path, weighted independent set, and spanning tree under the maximin criteria and show some hardness results.

There leave several future works. Closing the gap for robust s-t path still remains open. It is thus interesting to investigate whether there exists a better approximation upper bound or a tighter lower bound. We can also look at other robust optimization problems, e.g., the robust perfect matching problem. The best approximation known to date for robust matching is still $O(k)$ which can be achieved by a trivial algorithm. Since there exists a strong connection between s-t path and min-cost perfect matching, it would be intriguing to explore whether our methods can be applied to improving the upper bound of the robust matching problem.

References

1 Jacob D. Abernethy, Pranjal Awasthi, Matthaus Kleindessner, Jamie Morgenstern, Chris Russell, and Jie Zhang. Active sampling for min-max fairness. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 53–65. PMLR, 2022.

2 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Approximation of min-max and min-max regret versions of some combinatorial optimization problems. *Eur. J. Oper. Res.*, 179(2):281–290, 2007.

3 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438, 2009.

4 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. General approximation schemes for min-max (regret) versions of some (pseudo-)polynomial problems. *Discret. Optim.*, 7(3):136–148, 2010.

5 Yang An and Rui Gao. Generalization bounds for (wasserstein) robust optimization. In *NeurIPS*, pages 10382–10392, 2021.

6 Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*, volume 28 of *Princeton Series in Applied Mathematics*. Princeton University Press, 2009.

7 Vittorio Bilò, Ioannis Caragiannis, Angelo Fanelli, Michele Flammini, and Gianpiero Monaco. Simple greedy algorithms for fundamental multidimensional graph problems. In *ICALP*, volume 80 of *LIPIcs*, pages 125:1–125:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

8 Hans L. Bodlaender. Nc-algorithms for graphs with small treewidth. In *WG*, volume 344 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1988.

9 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*, pages 575–584. IEEE Computer Society, 2010.

10 Qingyun Chen, Sungjin Im, Benjamin Moseley, Chenyang Xu, and Ruilong Zhang. Min-max submodular ranking for multiple agents. *AAAI 2023, to appear*, 2023.

11 Altannar Chinchuluun and Panos M. Pardalos. A survey of recent developments in multiobjective optimization. *Ann. Oper. Res.*, 154(1):29–50, 2007.

12 Vincent Conitzer, Rupert Freeman, and Nisarg Shah. Fair public decision making. In *EC*, pages 629–646. ACM, 2017.

13 Michael Dinitz, Guy Kortsarz, and Shi Li. Degrees and network design: New problems and approximations. *CoRR*, abs/2302.11475, 2023. `arXiv:2302.11475`.

14 Daniel Duque, Leonardo Lozano, and Andrés L. Medaglia. An exact method for the biobjective shortest path problem for large-scale road networks. *Eur. J. Oper. Res.*, 242(3):788–797, 2015.

15 Matthias Ehrgott. *Multicriteria Optimization (2. ed.)*. Springer, 2005.

16 Brandon Fain, Kamesh Munagala, and Nisarg Shah. Fair allocation of indivisible public goods. In *EC*, pages 575–592. ACM, 2018.

17 Virginie Gabrel, Cécile Murat, and Aurélie Thiele. Recent advances in robust optimization: An overview. *Eur. J. Oper. Res.*, 235(3):471–483, 2014.

18 Yu Gang and Yang Jian. On the robust shortest path problem. *Comput. Oper. Res.*, 25(6):457–468, 1998.

19 Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. $O(\log^2 k \,/\, \log\log k)$-approximation algorithm for directed steiner tree: a tight quasi-polynomial-time algorithm. In *STOC*, pages 253–264. ACM, 2019.

20 Adam Kasperski and Pawel Zielinski. On the approximability of minmax (regret) network optimization problems. *Inf. Process. Lett.*, 109(5):262–266, 2009.

21 Adam Kasperski and Pawel Zielinski. On the approximability of robust spanning tree problems. *Theor. Comput. Sci.*, 412(4-5):365–374, 2011.

22 Adam Kasperski and Paweł Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. *Robustness analysis in decision aiding, optimization, and analytics*, pages 113–143, 2016.

23 Adam Kasperski and Pawel Zielinski. Approximating some network problems with scenarios. *CoRR*, abs/1806.08936, 2018. `arXiv:1806.08936`.

24 Ana Klobucar and Robert Manger. Solving robust weighted independent set problems on trees and under interval uncertainty. *Symmetry*, 13(12):2259, 2021.

25 Shi Li, Chenyang Xu, and Ruilong Zhang. Polylogarithmic approximation for robust s-t path. *CoRR*, abs/2305.16439, 2023. `arXiv:2305.16439`.

26 Xian Li and Hongyu Gong. Robust optimization for multilingual translation with imbalanced data. In *NeurIPS*, pages 25086–25099, 2021.

27 Natalia Martínez, Martín Bertrán, and Guillermo Sapiro. Minimax pareto fairness: A multi objective perspective. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 6755–6764. PMLR, 2020.

28 Fabrice Talla Nobibon and Roel Leus. Robust maximum weighted independent-set problems on interval graphs. *Optim. Lett.*, 8(1):227–235, 2014.

29 Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92. IEEE Computer Society, 2000.

30 Bozidar Radunovic and Jean-Yves Le Boudec. A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Trans. Netw.*, 15(5):1073–1083, 2007.

31 Antonio Sedeño-Noda and Marcos Colebrook. A biobjective dijkstra algorithm. *Eur. J. Oper. Res.*, 276(1):106–118, 2019.

32 Paolo Serafini. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent Advances and Historical Development of Vector Optimization: Proceedings of an International Conference on Vector Optimization*, pages 222–232. Springer, 1987.

33 Jacobo Valdes, Robert Endre Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.