Isotonic Optimization with Fixed Costs

Xin Chen[1], Shuai Li[1], Weijun Xie[1]
[1] H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA
xin.chen@isye.gatech.edu, sli884@gatech.edu, wxie@gatech.edu

This paper introduces a generalized isotonic optimization framework over an arborescence graph, where each node incurs state-dependent convex costs and a fixed cost upon strict increases. We begin with the special case in which the arborescence is a path and develop a dynamic programming (DP) algorithm with an initial complexity of $O(n^3)$, which we improve to $O(n^2 \log n)$ by exploiting problem structure, and further to $O(n)$ under an isotonic minima assumption via the Monge property. For general arborescences, where the path-based DP is inapplicable, we identify structural conditions under which the problem remains tractable. In particular, we present efficient polynomial-time algorithms for various special cases, such as with piecewise linear convex costs, having dominating fixed costs at parent nodes, and arborescences with restricted branching. Numerical experiments validate the computational advantage of our proposed algorithms over generic solvers.

Key words: isotonic optimization, fixed costs, dynamic programming, arborescence

1. Introduction

We consider a mixed integer nonlinear optimization problem on an arborescence graph. Specifically, we define an arborescence $\mathcal{T} = (N_{\mathcal{T}}, A_{\mathcal{T}})$ as a directed tree where all edges are directed away from the root. The node set and edge set are given by $N_{\mathcal{T}} = 0 \cup [n]$ and $A_{\mathcal{T}} = \{(p(i), i) \mid i \in [n]\}$, where $[n] = \{1, ..., n\}$ and $p(i)$ denote the parent node of node i. Without loss of generality, we assume that the root node 0 is only connected to node 1. For each $i \in [n]$, we attach it with an associated one-dimensional state x_i and a corresponding convex state cost $f_i(x_i)$. We impose isotonic constraints for variables whose indices form an arc in $A_{\mathcal{T}}$, i.e., $x_{p(i)} \le x_i$ for all $i \in [n]$. In addition, if $x_{p(i)} < x_i$, a fixed cost $K_i \ge 0$ is incurred. The objective is to minimize the overall state costs and fixed costs incurred by all nodes in $\mathcal{T}$. The isotonic requirements and the need to incorporate a fixed cost whenever a strict increase is made arise in many applications. We refer to Section 1.1 for several more detailed motivating examples. The above generalized isotonic optimization model can be formulated as follows:

$$\begin{aligned} \min_{(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{R}^n \times \{0,1\}^n} \ & \sum_{i \in [n]} K_i y_i + \sum_{i \in [n]} f_i(x_i) \\ \text{s.t.} \quad & (x_{p(i)} - x_i)(1 - y_i) = 0, \quad \forall i \in [n], \\ & x_{p(i)} \le x_i, \quad \forall i \in [n], \end{aligned} \tag{Arborescence}$$

where x_0 is a given parameter, and for each node $i \in [n]$, we let $y_i = 1$ indicate that the quantity x_i is strictly greater than that of its parent, $x_{p(i)}$. In the Arborescence problem, the first set of constraints ensures that a fixed cost is incurred whenever a strict increase occurs from a parent node. The second set of constraints enforces the isotonic inequalities on the arborescence. When the graph $\mathcal{T}$ degenerates into a path, the problem simplifies to the following:

$$\begin{aligned} \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathbb{R}^n\times\{0,1\}^n} & \sum_{i\in[n]} K_i y_i + \sum_{i\in[n]} f_i(x_i) \\ \text{s.t.} \quad & (x_i - x_{i-1})(1-y_i) = 0, \quad \forall i \in [n], \\ & x_0 \le x_1 \le x_2 \le \cdots \le x_n. \end{aligned} \tag{Path}$$

Both the Arborescence and Path problems are mixed-integer nonlinear programs (MINLPs), which are, in general, hard to solve. The purpose of this paper is to design efficient polynomial-time algorithms for the Path problem and a broad family of the Arborescence problem.

1.1. Motivating Examples

This subsection presents three motivating examples from the existing literature, each of which can be viewed as a special case of the Arborescence and Path problems.

Generalized Reduced Isotonic Regression: Isotonic optimization, also known as the isotonic regression, is a well-established problem in statistics and optimization (see, e.g., Barlow and Brunk 1972, Robertson and Wright 1975, Barlow and Ubhaya 1971, Bacchetti 1989). Given a sequence of univariate data points, the objective of the isotonic regression problem is to determine the best monotone nondecreasing sequence that optimally fits the data with respect to a specified convex loss function. Formally, suppose there are n data points, and let $f_i(x_i) : \mathbb{R} \to \mathbb{R}_+$ be a convex (loss) function that measures the discrepancy between the fitted output x_i and the observation. The isotonic regression problem can be expressed in the following general form:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ \sum_{i\in[n]} f_i(x_i) \mid x_1 \le x_2 \le \cdots \le x_n \right\}. \tag{1}$$

Although the most classical isotonic regression model (1) receives much attention in the literature, it is sometimes problematic. The number of markups is allowed to be any number and is at most n times if $x_1 < x_2 < \cdots < x_n$, which receives criticism that the isotonic regression may overfit the data (Niculescu-Mizil and Caruana 2005, Salanti and Ulm 2003, Schell and Singh 1997). Besides, a fitted step functions with too many steps occupy a lot of storage space on the computer. To address the overfitting issue in the isotonic optimization (1), the *reduced isotonic regression* (Schell and Singh 1997) is introduced to restrict the number of strict increases to be within a threshold $s \in \mathbb{N}$; i.e., an additional constraint $\sum_{i=2}^{n} \mathbb{1}\{x_i > x_{i-1}\} \le s$ is added. The authors show that the reduced

isotonic regression can be solved in polynomial time, and subsequent research has improved the time complexity (Haiminen et al. 2008, Hardwick and Stout 2014). Here, we adopt another formulation to impose sparsity. Instead of restricting the number of changes to be within a cardinality, we add a penalty term $\sum_{i=2}^{n} K_i \mathbb{1}\{x_i > x_{i-1}\}$ to the objective of the isotonic optimization (1) with appropriate penalty parameters $K_i \in \mathbb{R}_+$. We end up with the following more general problem:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^n, \boldsymbol{z}\in\{0,1\}^{n-1}} \quad & \sum_{i\in[n]} K_i y_i + \sum_{i\in[n]} f_i(x_i) \\ \text{s.t. } & (x_i - x_{i-1})(1-y_i) = 0, \quad \forall i \in [n]\backslash\{1\}, \\ & x_1 \le x_2 \le \cdots \le x_n, \end{aligned} \tag{GIR}$$

where $K_i \in \mathbb{R}_+$'s are fixed costs which are only incurred when $x_i > x_{i-1}$. We call (GIR) the *generalized reduced isotonic regression* problem, which is a special case of the Path problem. The advantage of this formulation is that it allows us to use local penalty (i.e., different penalty terms for different indicator functions), which is also studied in a separate stream of literature on functional model selection, e.g., Tu et al. (2012), Wang and Kai (2015). The local penalty enables us to fit the model with high accuracy in more important regions, while maintaining low accuracy in less important regions, depending on the application requirements. Besides, the optimization problem remains tractable, as we will show later. Despite the above advantages, to the best of our knowledge, the local sparsity is rarely studied in isotonic regression.

Open-loop Stochastic Lot-Sizing: The stochastic lot-sizing problem is extensively studied in the inventory management literature (see a comprehensive overview in Tempelmeier 2013). Consider a periodic review open-loop stochastic lot-sizing problem over a time horizon of length T. Let the random variable $d_t \in \mathbb{R}^+$ represent the stochastic demand at time t, with a cumulative distribution function $F_t(\cdot)$ for each $t \in [T]$. Let the decision variable $x_t \in \mathbb{R}^+$ denote the ordering quantity placed at time t for each $t \in [T]$. At time t, three types of costs are incurred. The fixed ordering cost $K_t \in \mathbb{R}^+$ applies whenever $x_t > 0$. The holding cost $h_t \in \mathbb{R}^+$ is associated with inventory storage, while the backlog cost $b_t \in \mathbb{R}_+$ accounts for unmet demand. The unit ordering cost can be subsumed into the inventory cost and can be normalized to zero here. The stochastic lot-sizing problem can be formulated as follows:

$$\max_{\boldsymbol{x}\in\mathbb{R}^n_+} \sum_{t\in[T]} \left(K_t \mathbb{1}\{x_t > 0\} + \mathbb{E}\left[h_t \left(\sum_{i\in[t]} x_i - \sum_{i\in[t]} d_i \right)^+ + b_t \left(\sum_{i\in[t]} d_i - \sum_{i\in[t]} x_i \right)^+ \right] \right). \tag{2}$$

It is worth noting that Vargas (2009) designed a dynamic programming algorithm to solve a special case of the stochastic lot-sizing problem, where the ratio of backlog and holding costs must be non-decreasing. The general stochastic lot-sizing problem can be treated as a special case of the Path problem. To see this, we define $y_t = \mathbb{1}\{x_t > 0\} \in \{0, 1\}$ as an indicator variable denoting whether

an order is placed at time t. We further define $D_t = \sum_{i\in[t]} d_i$ as the cumulative stochastic demand from time 1 to t and $X_t = \sum_{i\in[t]} x_i$ as the cumulative ordering quantity over the same period. The cost can then be represented in terms of the cumulative ordering quantity X_t and the cumulative demand D_t. By definition, the cumulative ordering quantities satisfy $0 = X_0 \le X_1 \le X_2 \le \cdots \le X_T$. If $y_t = 1$, a fixed cost K_t is incurred at time t, and the cumulative ordering quantity X_t can increase strictly beyond X_{t-1} by placing an order. Additionally, we define the function $f_t(X_t) := \mathbb{E}[h_t(X_t - D_t)^+ + b_t(D_t - X_t)^+]$, which is convex. Thus, the stochastic lot-sizing problem (2) can be equivalently expressed as:

$$\begin{aligned} \min_{\boldsymbol{X}\in\mathbb{R}^T, \boldsymbol{y}\in\{0,1\}^T} \quad & \sum_{t\in[T]} \Big(K_t y_t + f_t(X_t) \Big) \\ \text{s.t. } & (X_t - X_{t-1})(1-y_t) = 0, \quad \forall t \in [T], \\ & 0 = X_0 \le X_1 \le X_2 \le \ldots \le X_T, \end{aligned} \tag{SLS-O}$$

which is a special case of the Path problem.

Close-loop Stochastic Lot-Sizing: The aforementioned SLS-O is a two-stage open-loop model; that is, the order quantities are determined at the beginning of the planning horizon and do not rely on a particular realization of random demands. One multi-stage *closed-loop* stochastic lot-sizing model based on a scenario tree has been studied by Guan and Miller (2008), Guan (2011). Different from SLS-O, in the scenario-tree-based model, the order quantity decision at the current stage depends on past realizations. Suppose that there is a T-stage scenario tree $\mathcal{T} = (N_\mathcal{T}, A_\mathcal{T})$, where each node $i \in N_\mathcal{T} := \{0\} \cup [n]$ corresponds to a particular realization. Each node $i \in N_\mathcal{T}$ represents a stage, containing: a set of its child nodes $\mathcal{C}(i)$, a realization probability p_i (note that $p_i = \sum_{j\in\mathcal{C}(i)} p_j$), a demand realization d_i, a fixed ordering cost K_i, a unit holding cost h_i, and a unit backlog cost b_i. Let a binary decision $y_i \in \{0,1\}$ indicate whether to order at node i or not, and let the continuous variable $x_i \in \mathbb{R}_+$ denote the order quantity at node i. Let $\mathcal{P}(i,j)$ denote as the set of all nodes in the sub-path of tree $\mathcal{T}$ that starts from node $i \in N_\mathcal{T}$ and ends at node $j \in N_\mathcal{T}$. At each node i, there are fixed ordering cost $K_i y_i$ and the inventory cost $h_i(\sum_{\ell\in\mathcal{P}(1,i)} x_\ell - \sum_{\ell\in\mathcal{P}(1,i)} d_\ell)^+ + b_i(\sum_{\ell\in\mathcal{P}(1,i)} d_\ell - \sum_{\ell\in\mathcal{P}(1,i)} x_\ell)^+$. We can also incorporate linear ordering cost $c_i x_i$ at each node, which can be expressed as $\sum_{i\in[n]} c_i x_i$ as $\sum_{i\in[n]}(c_i - \sum_{j\in\mathcal{C}(i)})\sum_{\ell\in\mathcal{P}(1,i)} x_\ell$. Subsuming each term $(c_i - \sum_{j\in\mathcal{C}(i)})\sum_{\ell\in\mathcal{P}(1,i)} x_\ell$ for each $i \in [n]$ into the inventory cost at i, we assume without loss of generality that the linear ordering cost is zero. Similar to SLS-O model, by defining $X_i = \sum_{\ell\in\mathcal{P}(1,i)} x_\ell$ and $D_i = \sum_{\ell\in\mathcal{P}(1,i)} d_\ell$, the optimization problem can be reformulated as:

$$\begin{aligned} \min_{(\boldsymbol{X},\boldsymbol{y})\in\mathbb{R}^n\times\{0,1\}^n} \quad & \sum_{i\in[n]} K_i y_i + \sum_{i\in[n]} \Big(h_i(X_i - D_i)^+ + b_i(D_i - X_i)^+ \Big) \\ \text{s.t.} \quad & (X_{p(i)} - X_i)(1-y_i) = 0, \quad \forall i \in [n], \\ & X_{p(i)} \le X_i, \quad \forall i \in [n], \\ & X_0 = 0. \end{aligned} \tag{SLS-C}$$

We observe that the SLS-C problem is a special case of Arborescence problem where each f_i is piece-wise linear convex with only one kink point.

Joint Inventory Control and Pricing with Costly Price Markdown: Let us consider a deterministic joint inventory control and pricing model with finite horizon T studied by Chen and Hu (2012). At the beginning of each time $t \in [T]$, the retailer decides the ordering quantity $x_t \in \mathbb{R}_+$ and the price $p_t \in \mathbb{R}_+$ of the product. Let $c_t \in \mathbb{R}_+$ denote the unit ordering cost and d_t denote the price-dependent demand at time t. Let the remaining inventory at the beginning of time t be I_{t-1}. At time t, the cost $h_t(I_{t-1} + x_t - d_t)^+ + b_t(d_t - I_{t-1} - x_t)^+$ is incurred, where $h_t \in \mathbb{R}+$ and $b_t \in \mathbb{R}+$ are the unit holding and backlog costs, respectively, with $b_t > c_t$. We can easily figure out how demand d_t at each period t should be fulfilled and denote the corresponding marginal cost as $\widetilde{c}_t$, which is affine in c_t's, h_t's, and b_t's and is independent of the specific values of d_t's. Hence, the overall lot sizing cost is $\sum_{t\in[T]} \widetilde{c}_t d_t$. Let the demand function be $D_t(p_t) = \alpha_t D(p_t)$ where $\alpha_t \geq 0$ is the time-dependent multiplier and $D : \mathbb{R}_+ \to \mathbb{R}_+$ is the time-independent demand function for all times, which is a decreasing function in price. Then we can equivalently express the price function in demand d_t as $P_t(d) = D^{-1}(d_t/\alpha_t)$ and the revenue function as $R_t(d_t) = d_t \times P_t(d_t)$. If we let $d_t/\alpha_t = \widetilde{d}_t$, then $R_t(d_t) = \alpha_t R(\widetilde{d}_t)$, where $R(\widetilde{d}_t) = \widetilde{d}_t \times D^{-1}(\widetilde{d}_t)$ and is assumed to be a concave function with bounded maxima. We assume the price markdown is allowed, i.e., we impose the constraint $p_1 \geq p_2 \geq \ldots \geq p_n$, which can be equivalently expressed as $\widetilde{d}_1 \leq \widetilde{d}_2 \leq \ldots \leq \widetilde{d}_n$. We further assume that price change is costly, and the costs consist of menu cost and management cost. Assume a fixed cost $A_t \in \mathbb{R}_+$ is charged when $p_t > p_{t-1}$ or $\widetilde{d}_{t-1} < \widetilde{d}_t$. Hence, we can formulate the model as:

$$\begin{aligned} \min_{\widetilde{\boldsymbol{d}}\in\mathbb{R}^T, \boldsymbol{z}\in\{0,1\}^T} \quad & \sum_{t\in[T]} A_t z_t + \sum_{t\in[T]} \alpha_t \left(\widetilde{c}_t \widetilde{d}_t - R(\widetilde{d}_t) \right) \\ \text{s.t.} \quad & (\widetilde{d}_t - \widetilde{d}_{t-1})(1 - z_t) = 0, \quad \forall t \in [T], \\ & 0 = \widetilde{d}_0 \leq \widetilde{d}_1 \leq \widetilde{d}_2 \leq \ldots \leq \widetilde{d}_T. \end{aligned} \tag{CPM}$$

Since $\alpha_t(\widetilde{c}_t \widetilde{d}_t - R(\widetilde{d}_t))$ is a convex function with bounded minima, we can see that the CPM problem is a special case of the Path problem.

1.2. Literature Review

There are three streams of relevant literature: isotonic regression, general isotonic optimization problems, and application problems with isotonic constraints. Most existing works do not incorporate fixed costs into their models and can be solved in polynomial time using well-developed algorithms.

The literature on isotonic regression primarily focuses on problems of the form (1), where the convex function $f_i(x_i)$ is typically of the form $w_i|x_i - b_i|^p$ for $1 \leq p < \infty$, with $w_i \geq 0$ for each $i \in [n]$. When $p = 1$, the problem is referred to as isotonic median regression (Robertson and Wright 1973,

Chakravarti 1989). When $p = 2$, the problem corresponds to the standard isotonic regression problem, which has been widely studied in the literature (Barlow and Brunk 1972, Best and Chakravarti 1990, Dykstra 1981). For the general case where each convex function $f_i(x_i)$ has an $O(1)$ evaluation oracle, Best et al. (2000) showed that the Pool Adjacent Violators (PAV) algorithm could be used to solve the isotonic regression problem optimally in polynomial time. Subsequent research has aimed at improving the computational efficiency of solving isotonic regression problems. For example, Ahuja and Orlin (2001) introduced a cost scaling approach to accelerate the PAV algorithm, while Stout (2013) developed a partition-based technique to achieve the same state-of-the-art complexity. Beyond these standard formulations, the literature has explored alternative convex and non-convex objectives for problem (1). For instance, L_∞-norm isotonic regression considers an objective of the form $\max_{i\in[n]} w_i|\widehat{y}_i - y_i|$ and has been studied in Stout (2018, 2014, 2015). Similarly, L_0-norm isotonic regression minimizes the number of misclassified values, with the objective $\sum_{i\in[n]} w_i\delta(\widehat{y}_i \neq y_i)$, where $\delta(\widehat{y}_i \neq y_i) = 1$ if $\widehat{y}_i \neq y_i$ and 0, otherwise (Stout 2021a,b). For a comprehensive overview of the fastest known algorithms for these isotonic regression problems, readers may refer to Stout (2019, 2023). However, none of these existing algorithms can be directly applied to isotonic optimization with fixed costs and this paper fills this gap.

Existing literature also studies isotonic optimization problems in more general settings, extending the classical isotonic regression formulations. One extension involves "soft" isotonic models, where isotonic constraints can be violated with a penalty cost. Examples of such works include nearly isotonic regression (Tibshirani et al. 2011), the convex s-excess problem (Hochbaum 2001), and the convex dual minimum-cost network flow problem (Ahuja et al. 2003). Another line of research enforces isotonic constraints on the edges of general directed acyclic graphs (DAGs). For example, Stout (2008) studied unimodal regression, where the non-decreasing constraints in (1) are replaced with unimodal constraints of the form $\widehat{y}_1 \leq \ldots \leq \widehat{y}_m \geq \widehat{y}_{m+1} \geq \ldots \geq \widehat{y}_n$ for some $m \in [n]$. Other works investigate isotonic regression on arborescences (Pardalos and Xue 1999, Stout 2013, 2015) and on general DAGs (Wang et al. 2022). Additionally, some studies apply soft isotonic models to more general structures, such as paths (Yu et al. 2023) and directed trees (Chen et al. 2023), designing optimal dynamic programming or PAV-based algorithms to efficiently solve these problems.

Beyond isotonic regression, isotonic constraints play a significant role in modeling various classical operations research problems. One notable example is the extension of the Economic Order Quantity (EOQ) model to multistage production systems, including serial, assembly, and distribution systems, as well as more general system structures (Schwarz and Schrage 1975, Crowston et al. 1973, Love 1972, Maxwell and Muckstadt 1985). Beyond multistage inventory systems, isotonic constraints also arise in other operational problems, such as the center location problem (Kaufman and Tamir 1993, Tamir 1993) and various other applications surveyed in Hochbaum (2004). However, none of these existing works incorporate fixed costs imposed on strict monotonicity into their models.

1.3. Summary of Contributions

The contributions of this paper are summarized as follows:

- *Modeling Power*: The proposed Arborescence problem and Path models can cover many problems from the literature, including generalized reduced isotonic regression, stochastic lot sizing, joint inventory control, and pricing with costly price markdowns.
- *Efficient Algorithms for the Path Problem*: To efficiently solve the Path problem, we develop a dynamic programming (DP) algorithm by leveraging optimality conditions. While a naive DP implementation has a time complexity of $O(n^3)$, we improve it to $O(n^2 \log n)$ by exploiting the problem's structural properties. Furthermore, when the convex cost functions exhibit isotonic minima, we prove that the DP algorithm can be further improved to achieve a linear runtime $O(n)$.
- *Efficient Algorithms for the Arborescence Problem*: The DP algorithms developed for the Path problem may not be directly applicable to the Arborescence problem. To address this, we design new DP algorithms specifically for solving the Arborescence problem. In particular, we show that when cost functions are convex piecewise linear with a constant number of pieces, the Arborescence problem can be solved in $O(n^2)$ time. This result recovers the best-known complexity for the scenario-tree-based lot-sizing problem but with a much simpler algorithm. For general convex cost functions, we prove that the corresponding value-to-go functions are piecewise convex in their states. Under mild conditions on the fixed costs or the structure of arborescences, we further show that the number of convex pieces is polynomial in n. Hence, the Arborescence problem remains polynomial-time solvable under these conditions.
- *Effective Numerical Experiments*: Compared to the off-the-shelf solver Gurobi, our proposed algorithms efficiently find optimal solutions, whereas Gurobi struggles to solve most instances within the time limit. For the general Arborescence problem, although we are unable to show a polynomial-time complexity for our DP algorithms, numerical experiments demonstrate that they remain highly efficient on randomly generated instances. These results highlight the practical effectiveness of our approach in handling large-scale problem instances.

Paper Structure and Notations: The structure of the paper is as follows. Section 2 investigates a DP algorithm for solving the Path problem and enhances its time complexity for both the general case and a special case where the functions f_i exhibit isotonic minima. Section 3 and Section 4 extend the study to the more general Arborescence problem and develop efficient polynomial-time DP algorithms for special cases, where Section 3 considers two special cost structures and Section 4 considers special arborescence structures. Section 5 presents numerical experiments to validate the superior computational performance of the proposed algorithms. Finally, Section 6 summarizes the findings and concludes the paper.

In this paper, all vectors are written in boldface, while scalars and matrices are not. $[n] := \{1, \ldots, n\}$ for any $n \in \mathbb{Z}_{++}$. $\mathbb{1}\{\text{condition}\}$ equals 1 if condition = YES and equals 0 if condition = NO. Denote $f'_-(x)$ and $f'_+(x)$ as the left and right derivatives of a real-valued function $f : \mathbb{R} \to \mathbb{R}$ at x if they exist. A function $f(n)$ is said to be $O(g(n))$ as $n \to \infty$ if there exist positive constants M and n_0 such that $|f(n)| \le M|g(n)|$ for all $n \ge n_0$. For an arborescence graph $\mathcal{T} = (N_\mathcal{T}, A_\mathcal{T})$, let $\mathcal{T}(i) = (N_{\mathcal{T}(i)}, A_{\mathcal{T}(i)})$ be the sub-arborescence of $\mathcal{T}$ rooted at $i \in N_\mathcal{T}$, and $\mathcal{P}(i, j)$ be the set of all nodes in the sub-path of $\mathcal{T}$ that starts from node $i \in N_\mathcal{T}$ and ends at node $j \in N_\mathcal{T}$. We denote $\mathcal{L}$ as the set of leaf nodes in $\mathcal{T}$, and $\mathcal{L}(i)$ as the set of leaf nodes in $\mathcal{T}(i)$. All the proofs in this paper are delegated in the appendix.

2. DP Algorithms for the Path Problem

This section first presents a DP algorithm to solve the Path problem, leveraging its optimality condition. We then enhance this DP algorithm by eliminating redundant computations to improve its efficiency. Additionally, we introduce a new DP algorithm with significantly improved time complexity for a special case.

For ease of presentation, we assume that $x_0 = -\infty$. Note that this is without loss of generality since when $x_0 = \ell$ instead of $-\infty$, we can insert an index $0'$ between 0 and 1 and associate a fixed cost $K_{0'} = 0$ and a convex cost function $f_{0'}(x_{0'}) = \delta(x_{0'} = \ell)$ to the index, where $\delta(A) = 0$ if $A =$ YES and ∞ otherwise, to recover the original (Path) problem.

2.1. A DP Algorithm and Its Efficient Implementation

We first provide an optimality condition for the Path problem, which serves as the foundation for designing a DP algorithm to solve it. To begin with, we introduce key notations that are frequently used throughout this paper. For each pair (i, j) satisfying $1 \le i \le j \le n$, define $\overline{x}^*_{i,j}$ as (i, j)-opt solution and $c_{i,j}$ as its associated cost, which admit the following expressions:

$$\overline{x}^*_{i,j} = \min \left\{ x^* | x^* \in \arg\min \sum_{\ell=i}^{j} f_\ell(x) \right\}, c_{i,j} = K_i + \sum_{\ell=i}^{j} f_\ell(\overline{x}^*_{i,j}). \tag{3}$$

Any optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ of the Path problem partitions $[n]$ into distinct blocks, each of which consists of a subset of consecutive indices, and the corresponding x^*_i values in a block are the same. Denote $B(i, j)$ to be the block containing indices from i to j. For any two consecutive blocks $B(i, j)$ and $B(j + 1, k)$ in the partition, the values must satisfy the monotonicity condition $x^*_i = \ldots = x^*_j < x^*_{j+1} = \ldots = x^*_k$. Thus, $\overline{x}^*_{i,j}$ can be interpreted as the smallest optimal solution of the unconstrained problem for the block $B(i, j)$, with $c_{i,j}$ representing its corresponding cost. This result is formally summarized below. For simplicity, we refer to a solution of the Path problem as *optimal* if it optimally solves the Path problem using the minimum number of increases (i.e., making the minimum number of partitions), and in each block of the optimal partition, the value of the optimal x^*_i's is minimized in case with multiple optimal solutions.

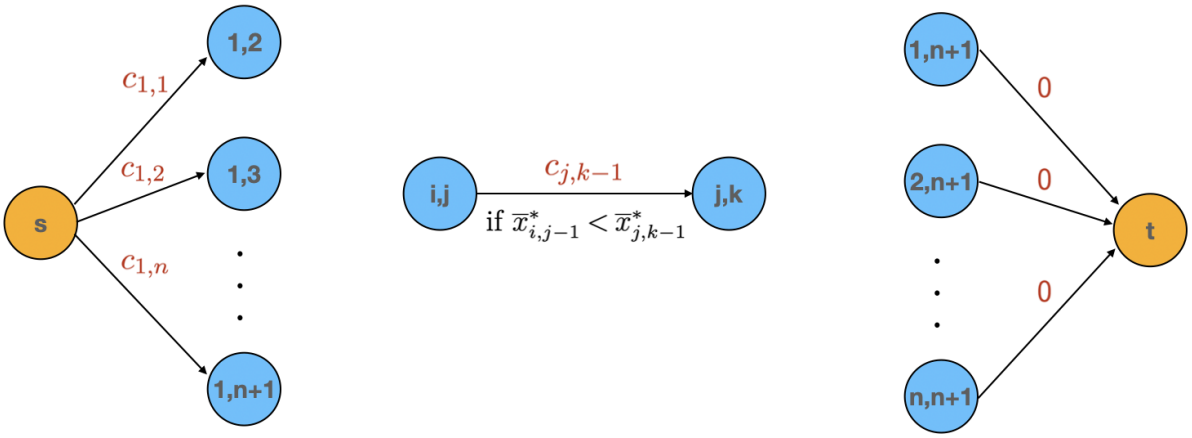

Figure 1 Illustration of the graph where finding the shortest (s,t)-path in it is equivalent to solving (DP-original). Each blue node (i,j) with $1 \le i < j \le n+1$ represents the block $B(i,j-1)$ and two yellow nodes are s and t. The arcs pointing out from s only towards $(1,2),(1,3),\ldots,(1,n+1)$ whose costs are $c_{1,1},c_{1,2},\ldots,c_{1,n}$, and arcs pointing to t only from $(1,n+1),(2,n+1),\ldots,(n,n+1)$ whose costs are all 0. Between any two blue nodes $(i,j),(j,k)$, an arc can only be placed between them if they satisfy $\overline{x}^*_{i,j-1} < \overline{x}^*_{j,k-1}$, and the cost is $c_{j,k-1}$.

THEOREM 1. *Suppose* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *is an optimal solution for the Path problem such that* $y^*_i = 1$ *if* $i \in \{i_0, i_1, i_2, \ldots, i_R\} \subseteq [n]$ *where* $1 = i_0 < i_1 < i_2 < \ldots < i_R < i_{R+1} = n+1$ *and* $y^*_i = 0$. *Then*

(a) $x^*_i = \overline{x}^*_{i_{r-1}, i_r - 1}$ *for all* $i \in B(i_{r-1}, i_r - 1)$; *and*

(b) The strict inequality $\overline{x}^*_{i_{r-1}, i_r - 1} < \overline{x}^*_{i_r, i_{r+1}-1}$ *must hold for any* $r \in [R]$.

The optimality condition in Theorem 1 allows us to design an exact DP algorithm. The algorithm begins by precomputing all values of $\overline{x}^*_{i,j}$ and $c_{i,j}$. The goal of the DP algorithm is to determine the optimal partition of the set $[n]$ into blocks $B(1, i_1 - 1), B(i_1, i_2 - 1), \ldots, B(i_R, n)$, such that the sequence satisfies $\overline{x}^*_{1,i_1-1} < \overline{x}^*_{i_1,i_2-1} < \ldots < \overline{x}^*_{i_R,n}$, while minimizing the total cost $\sum_{r \in [R+1]} c_{i_{r-1}, i_r - 1}$.

For each (j,k) pair such that $1 \le j < k \le n+1$, we define the value-to-go function $V(j,k)$ as the minimum cost incurred by the first $k-1$ indexes in the Path problem, given that the last block in the partition of the index set $[k]$ is $B(j, k-1)$, or equivalently, the last index that has a strict increase before k is j. That is, we consider

$$V(j,k) = \begin{cases} c_{1,k-1}, & \text{if } 1 = j < k \le n+1, \\ c_{j,k-1} + \min_{i:1 \le i < j, \overline{x}^*_{i,j-1} < \overline{x}^*_{j,k-1}} V(i,j), & \text{if } 2 \le j < k \le n+1. \end{cases} \tag{DP-original}$$

The value $\min_{j \in [n]} V(j, n+1)$ corresponds to the optimal value of the Path problems. Note that the formulation (DP-original) can also be interpreted as finding the (s,t) shortest path of the graph illustrated in Figure 1.

The formulation (DP-original), assuming all $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s are precomputed, has a time complexity of $O(n^3)$. This complexity arises because there are $O(n^2)$ states, and for each state, the naive computation of the value-to-go function requires $O(n)$ arithmetic operations (i.e., additions and comparisons). To accelerate the naive implementation, we observe that for each $j=2,\dots,n$, the evaluations of $V(j,k)$ for any $k=j+1,\dots,n+1$ are structurally similar, differing only in the placement of different upper bounds when selecting the optimal i to evaluate $\min_{i:1\le i<j,\overline{x}^*_{i,j-1}<\overline{x}^*_{j,k-1}} V(i,j)$. Specifically, for any $k=j+1,\dots,n+1$, computing $V(j,k)$ requires identifying the optimal preceding index i such that $V(i,j)$ is minimized over all i's with $\overline{x}^*_{i,j-1}$ strictly less than $\overline{x}^*_{j,k-1}$. We observe that although the valid i's for each k depend on $\overline{x}^*_{j,k-1}$, they are all drawn from a common pool. Algorithm 1 takes advantage of this by first sorting the combined set of values $\{\overline{x}^*_{i,j-1}\}_{i=1}^{j-1}\cup\{\overline{x}^*_{j,k-1}\}_{k=j}^{n}$ in ascending order. Suppose $\overline{x}^*_{j,k-1},\overline{x}^*_{j,k'-1}$ are two consecutive elements in the sorted list with $\overline{x}^*_{j,k-1}\le\overline{x}^*_{j,k'-1}$ and $k,k'\ge j$. When $V(j,k)$ has already been computed, to compute $V(j,k')$ we only need to take the minimum over all $V(i,j)$'s with i's such that $\overline{x}^*_{j,k-1}\le\overline{x}^*_{i,j-1}<\overline{x}^*_{j,k'-1}$, and compare it with $\min_{i:1\le i<j,\overline{x}^*_{i,j-1}<\overline{x}^*_{j,k-1}} V(i,j)$ which has already been computed and recorded when computing $V(j,k)$. This prevents us from scanning and doing comparisons over $V(i,j-1)$ for the same i multiple times, and will ultimately save computation. The implementation details are provided in Algorithm 1, with correctness and time complexity analyzed in Proposition 1. Notably, this optimization reduces the time complexity from $O(n^3)$ to $O(n^2\log n)$, significantly improving computational efficiency. Additionally, the time complexity of precomputing $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s must be considered when evaluating the overall complexity of the algorithm. This aspect will be analyzed in the subsequent subsections.

PROPOSITION 1. *Suppose that $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s are precomputed. Then, the DP recursion* (DP-original) *can be efficiently implemented using Algorithm 1 in $O(n^2\log n)$ time.*

2.2. Acceleration with Isotonic Minima

In this subsection, we study a special case of the Path problem that satisfies the following assumption.

ASSUMPTION 1 **(Isotonic Minima)**. *Let $\overline{x}^*_{i,i}=\min\{x\mid x\in\arg\min f_i(x)\}$ for each $i\in[n]$. We refer to the sequence $\{\overline{x}^*_{i,i}\}_{i\in[n]}$ as isotonic minima if it satisfies the condition $\overline{x}^*_{1,1}\le\overline{x}^*_{2,2}\le\ldots\le\overline{x}^*_{n,n}$.*

According to Assumption 1 and the convexity of the functions $f_i(x)$, it follows that $\overline{x}^*_{i,j-1}\le\overline{x}^*_{j,j}\le\overline{x}^*_{j,k-1}$ for any $1\le i<j<k\le n+1$. This holds because $\overline{x}^*_{i,j-1}$ minimizes a sum of convex functions whose individual minimizers are all less than or equal to $\overline{x}^*_{j,j}$, while $\overline{x}^*_{j,k-1}$ minimizes a sum of convex functions with individual minimizers are all greater than or equal to $\overline{x}^*_{j,j}$. This property allows us to simplify the DP recursion by reducing the state space. Specifically, using this property and the

Algorithm 1 Efficient Implementation of DP Recursion (DP-original)

Input: $\overline{x}^*_{i,j}$'s, $c_{i,j}$'s , for (i,j) such that $1 \le i \le j \le n$
for $k = 2, \ldots, n+1$ **do**
    $V(1,k) \leftarrow c_{1,k-1}$
end for
for $j = 2, \ldots, n$ **do**
    Index $\overline{x}^*_{1,j-1}, \ldots, \overline{x}^*_{j-1,j-1}, \overline{x}^*_{j,j}, \ldots, \overline{x}^*_{j,n}$ to be $1, \ldots, j-1, j, \ldots, n$
    Sort $\overline{x}^*_{1,j-1}, \ldots, \overline{x}^*_{j-1,j-1}, \overline{x}^*_{j,j}, \ldots, \overline{x}^*_{j,n}$ in the ascending order, and let $\pi : [n] \to [n]$ be the corresponding permutation, with $\pi(\ell)$ denoting its ℓ-th element.
    $V_{\min}(0) \leftarrow \infty$
    for $\ell = 1, \ldots, n$ **do**
        if $\pi(\ell) \ge j$ **then**
            $V_{\min}(\ell) \leftarrow V_{\min}(\ell-1)$
            $V(j, \pi(\ell)+1) \leftarrow c_{j,\pi(\ell)} + V_{\min}(\ell)$
        else
            $V_{\min}(\ell) \leftarrow \min\{V(\pi(\ell), j), V_{\min}(\ell-1)\}$
        end if
    end for
end for
Output: $\min_{j \in [n]} V(j, n+1)$ and the solution towards it

observation that in the formulation (DP-original), the strict inequality $\overline{x}^*_{i,j-1} < \overline{x}^*_{j,k-1}$ can be replaced by a less than or equal to inequality, the value-to-go function $V(j,k)$ can be simplified as

$$V(j,k) = \begin{cases} c_{1,k-1}, & \text{if } 1 = j < k \le n+1, \\ c_{j,k-1} + \min_{i \in [j-1]} V(i,j), & \text{if } 2 \le j < k \le n+1. \end{cases}$$

Redefining $\hat{V}(j-1) := \min_{i \in [j-1]} V(i,j)$ for each $j \in [n+1]$ with $V(0) = 0$, we can rewrite the formulation (DP-original) as

$$\hat{V}(j) = \begin{cases} 0, & j = 0, \\ \min_{i:1 \le i \le j} \{c_{i,j} + \hat{V}(i-1)\} & j \in [n]. \end{cases} \qquad \text{(DP-isotonic)}$$

This is equivalent to the DP recursion for finding the shortest (s,t)-path in the graph shown in Figure 2, where each node represents an index. The terminal cost $\hat{V}(n)$ corresponds to the optimal value of the Path problem.

A simple observation is that the new DP recursion (DP-isotonic) can be computed in $O(n^2)$ time, improving upon the $O(n^2 \log n)$ complexity from the previous subsection. Furthermore, we can show

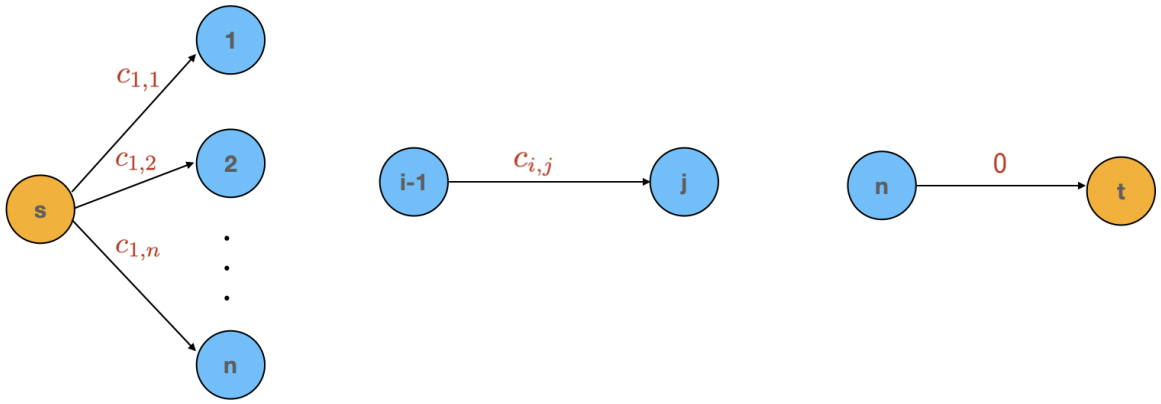

Figure 2 Illustration of the graph where finding the shortest (s,t)-path is equivalent to solving the DP recursion (DP-isotonic). Each index i with $1 \le i \le n$ is represented by a blue node. The only two yellow nodes are s and t, with arcs pointing out from s only towards $1,2,\ldots,n$ whose distances are $c_{1,1}, c_{1,2}, \ldots, c_{1,n}$, and arcs pointing to t only from n with distance 0. Between any two blue nodes $i-1$ and j with $i \le j$, an arc is directed from $i-1$ to j with distance $c_{i,j}$.

that the DP recursion (DP-isotonic) admits a significantly better $O(n)$ time complexity by leveraging the Monge property of the costs $c_{i,j}$'s.

Definition 1 (Monge Property of a Matrix). An $m \times n$ matrix M is *Monge* if for any $i \in [m-1], j \in [n-1]$:

$$M_{i,j} + M_{i+1,j+1} \le M_{i+1,j} + M_{i,j+1};$$

i.e., for any $i < i' \in [m]$ and $j < j' \in [n]$, a Monge matrix M must satisfy $M_{ij} + M_{i',j'} \le M_{i',j} + M_{i,j'}$.

The Monge property, in fact, implies *total monotonicity*. The $m \times n$ matrix M is *monotone* if $i_M(j) \le i_M(j')$ for any $j < j'$ where $i_M(j)$ is the smallest row index of the minimum element in the jth column of the matrix M, and is *totally monotone* if every 2×2 submatrix of M is monotone. When $\boldsymbol{M}$ is Monge, we have $M_{i',j'} - M_{i,j'} \le M_{i',j} - M_{i,j}$ for any $i < i'$ and $j < j'$, which implies that if $M_{i',j} \le M_{i,j}$, then $M_{i',j'} \le M_{i,j'}$. That is, a Monge matrix must be totally monotone. For a general matrix M, finding all column minima requires $O(mn)$ time since finding the minimum of one column takes $O(m)$ time and there are n columns. When the matrix $\boldsymbol{M}$ is Monge and therefore totally monotone, Aggarwal et al. (1986) showed that the *SMAWK algorithm* can be used to find all column minima in $O(m+n)$ time. The SMAWK algorithm applies two reductions, named *REDUCE* and *INTERPOLATE*, recursively and alternatively, whose efficient implementations rely on the total monotonicity property. Readers can refer to the original paper for details of the algorithm.

We show that the Monge property can accelerate our DP algorithm. For each $i, j \in [n]$, let us define $\widetilde{c}_{i,j}$ as

$$\widetilde{c}_{i,j} = \begin{cases} c_{i,j} & i \le j, \\ \infty & i > j, \end{cases} \tag{4}$$

and $u_{i,j} = \widetilde{c}_{i,j} + \hat{V}(i-1)$. The DP formulation (DP-isotonic) can be equivalently transformed to be:

$$\hat{V}(j) = \begin{cases} 0, & j = 0 \\ \min_{i:1\le i\le j} u_{i,j}, & j \in [n]. \end{cases}$$

Since the DP algorithm operates in a forward propagation manner, each entry $u_{i,j}$ can be evaluated in $O(1)$ time. Specifically, at step j, the value $\hat{V}(i)$ is available if its index $i \le j$. Conversely, if $i > j$, the entry $u_{i,j}$ is set to ∞. Thus, computing the values $\hat{V}(j)$ is equivalent to recursively identifying all column minima of the matrix $\boldsymbol{U} = (u_{i,j})_{i\in[n],j\in[n]}$. In particular, if the matrix $\boldsymbol{U}$ satisfies the Monge property, and all the elements of $\boldsymbol{U}$ are known in advance, the SMAWK algorithm (Aggarwal et al. 1986) can be applied here to compute all column minima in $O(n)$ time. However, it is important to note that the original SMAWK (Aggarwal et al. 1986) algorithm is not directly applicable in our DP setting because the evaluation of $u_{i,j}$ depends on the prior computation of $\hat{V}(i-1)$ when $i \le j$. Nevertheless, we can still use the modified SMAWK algorithm (Galil and Park 1989) to solve the problem in $O(1)$ time by guessing and correcting value-to-go functions along the way.

It remains to show that the matrix $\boldsymbol{U}$ is Monge, i.e., to prove that $u_{i,j} + u_{i+1,j+1} \le u_{i+1,j} + u_{i,j+1}$ for every $i \in [n-1], j \in [n-1]$, which is equivalent to

$$\begin{aligned} & \widetilde{c}_{i,j} + \hat{V}(i-1) + \widetilde{c}_{i+1,j+1} + \hat{V}(i) \le \widetilde{c}_{i+1,j} + \hat{V}(i) + \widetilde{c}_{i,j+1} + \hat{V}(i-1) \\ \iff \quad & \widetilde{c}_{i,j} + \widetilde{c}_{i+1,j+1} \le \widetilde{c}_{i+1,j} + \widetilde{c}_{i,j+1}. \end{aligned}$$

Hence, the matrix $\boldsymbol{U}$ is Monge if and only if the matrix $\widetilde{\boldsymbol{C}} = (\widetilde{c}_{i,j})_{i\in[n],j\in[n]}$ is. In fact, we show that the matrix $\widetilde{\boldsymbol{C}}$ indeed has Monge property under the isotonic minima assumption (i.e., Assumption 1 holds).

THEOREM 2. *Under Assumption 1, matrix $\widetilde{C} = (\widetilde{c}_{i,j})_{i\in[n],j\in[n]}$ defined by (4) satisfies the Monge property, i.e.,*

$$\widetilde{c}_{i,j} + \widetilde{c}_{i+1,j+1} \le \widetilde{c}_{i+1,j} + \widetilde{c}_{i,j+1}$$

for each $i \in [n-1], j \in [n-1]$.

When the matrix $\widetilde{C}$ satisfies the Monge property, the Path problem can be solved in $O(n)$ time, as summarized below.

COROLLARY 1. *Suppose that Assumption 1 holds and each $c_{i,j}$ for $1 \le i \le j \le n$ can be queried in $O(1)$ time. Then the Path problem can be solved in $O(n)$ time.*

Hence, if the pre-computation time for the $c_{i,j}$ values is $O(n)$, then the Path problem can be solved in $O(n)$ time under Assumption 1. Importantly, during the pre-processing, it is not necessary to compute all $c_{i,j}$ values explicitly. Instead, it suffices to preprocess the data such that each $c_{i,j}$ can be queried in $O(1)$ time on demand because the SMAWK algorithm only requires $O(n)$ such queries in total. The $O(n)$ pre-computation time holds for certain special cases, as discussed in the next subsection.

REMARK 1. The classical lot-sizing problem with backlog is a special case of our Path formulation and satisfies Assumption 1. Aggarwal and Park (1993) accelerated the naive $O(n^3)$ DP to $O(n \log n)$ via a divide-and-conquer method that exploits the Monge property, but their approach differs fundamentally from ours. Their DP recursion is formed by selecting the preceding ordering and zero-inventory epochs, without precomputing $x^*_{i,j}$'s or $c_{i,j}$'s, while our DP recursion is computed by selecting the last index at which an increase is made and needs to do pre-computing. Hence, the ideas behind the acceleration are also different. While our method incurs a higher overall complexity of $O(n^2 \log n)$ (comprising $O(n)$ for the DP recursion and $O(n^2 \log n)$ for precomputing arc costs as established in the second proposition of Proposition 2 in Section 2.3), it is conceptually simpler and easier to implement. Moreover, unlike their method, our framework extends naturally to broader settings, including stochastic lot-sizing.

2.3. Computation of $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s

In the previous subsections, we have discussed the time complexity of the DP algorithm for solving the Path problem and its acceleration under the assumption with isotonic minima of f_i's. However, the overall time complexity of the DP algorithm also depends on the computation of $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s. This subsection addresses this issue.

The General Binary Search Algorithm: Recall that for each $i < j \in [n]$, the value $\overline{x}^*_{i,j}$ is the minimum minimizer of $\sum_{\ell=i}^{j} f_\ell(x)$. Thus, computing $\overline{x}^*_{i,j}$ requires solving a convex minimization problem, which can be efficiently handled using the binary search algorithm. Without loss of generality, we assume that each function $f_i(x)$ has a bounded minimum within the interval $[\ell, u]$ and define $U = u - \ell$. Let $\epsilon \in \mathbb{R}_+$ denote the tolerance level for each $\overline{x}^*_{i,j}$. Since evaluating $f_i(x)$ at any point $x \in [\ell, u]$ takes $O(1)$ time, computing $\sum_{\ell=i}^{j} f_\ell(x)$ at a point $x \in [\ell, u]$ requires $O(n)$ time. The binary search procedure to obtain an ϵ-optimal solution $\overline{x}^*_{i,j}$ requires $\log(U/\epsilon)$ search steps. Consequently, finding an ϵ-optimal solution for each pair (i, j) takes $O(n \log(U/\epsilon))$ time. According to (3), the overall complexity of computing all $\overline{x}^*_{i,j}$ and $c_{i,j}$ values is $O(n^3 \log(U/\epsilon))$ time.

This computation complexity dominates the $O(n^2 \log n)$ of the DP algorithm and worsens the overall complexity. Ahuja and Orlin (2001) used the separability structure to accelerate those computations, but their method can only be implemented under the framework of the PAV algorithm

and cannot be applied to our setting. Although we may not be able to improve this complexity, we show that efficient implementations exist for the motivating examples mentioned at the beginning of the paper.

Efficient Computation for Motivating Examples: To efficiently compute $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s for the motivating examples, we first introduce a lemma that allows us to develop efficient computation when the functions f_i are quadratic or piecewise linear convex with only a single kinking point.

LEMMA 1. *To compute $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s defined in (3) for all pairs (i,j) such that $1 \le i \le j \le n$, we have*

(i) If $f_i(x) = p_i x^2 - 2q_i x + r_i$ with $p_i > 0$ for any $i \in [n]$, then the computation can be done in $O(n^2)$ time; and

(ii) If $f_i(x) = p_i(r_i - x)^+ + q_i(x - r_i)^+$ with $p_i, q_i > 0$ for any $i \in [n]$, then the computation can be done in $O(n^2 \log n)$ time.

According to Lemma 1, combined with the results in Proposition 1, we conclude that the overall DP algorithm can be implemented in $O(n^2 \log n)$ time when the functions f_i's are either quadratic or piecewise linear convex with a single kinking point.

REMARK 2. When the functions $\{f_i\}_{i\in[n]}$ have isotonic minima, leveraging the Monge property in Theorem 2 and the SMAWK algorithm, the DP algorithm can be implemented in $O(n)$ time, provided that preprocessing allows each $\overline{x}^*_{i,j}$ and $c_{i,j}$ to be queried in $O(1)$ time. Importantly, this approach eliminates the need to explicitly compute all $\overline{x}^*_{i,j}$ and $c_{i,j}$ values beforehand. For the case where f_i's are quadratic functions, as shown in the proof of Lemma 1, an $O(n)$ preprocessing time suffices to ensure that each $\overline{x}^*_{i,j}$ and $c_{i,j}$ can be queried in $O(1)$ time. Consequently, the overall computational complexity for this case is only $O(n)$.

We now show the connection between the two cases in Lemma 1 and the three motivating examples introduced in Section 1.1.

- For the generalized reduced isotonic optimization problem (GIR), the weighted ℓ_1 norm (i.e., $f_i(\widehat{y}_i) = w_i|\widehat{y}_i - y_i|$ for each $i \in [n]$) and the weighted squared ℓ_2 norm (i.e., $f_i(\widehat{y}_i) = w_i(\widehat{y}_i - y_i)^2$ for each $i \in [n]$) are two commonly used norms in the literature. These norms align directly with the two cases in Lemma 1.
- For the stochastic lot-sizing problem (SLS-O), consider the standard scenario tree model (Guan and Miller 2008) with S scenarios, where each scenario $s \in [S]$ occurs with probability p_s, has demand $d_t(s)$ at time t, and cumulative demand $D_t(s)$ from time 1 to t. The cost function can be expressed as $f_t(X_t) = \sum_{s\in[S]} p_s(h_t(X_t - D_t(s))^+ + b_t(D_t(s) - X_t)^+)$. To ensure that each $f_t(X_t)$ can be queried in $O(1)$ time for any $X_t \in \mathbb{R}+$, we assume that $S = O(1)$ and that all $D_t(s)$ values are precomputed.

Since each $f_t(X_t)$ is the summation of S piecewise linear convex functions, each with a single kinking point, we can modify the algorithm for Case (ii) in Lemma 1 by inserting S nodes into the augmented binary search tree at each step. This allows us to compute all $\overline{x}^*_{i,j}$ and $c_{i,j}$ values in $O(Sn^2\log(Sn)) = O(n^2\log n)$ since $S = O(1)$.

- For the joint inventory control and pricing problem with costly price markdown (CPM), when the demand functions are linear in price, the revenue function $R(\widetilde{d})$ is quadratic, which corresponds to Case (i) of Lemma 1.

These observations are summarized in the following proposition.

PROPOSITION 2. *To compute $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s defined by (3) for all (i,j) such that $1 \le i \le j \le n$, we have*

(GIR): for ℓ_1 and ℓ_2 generalized reduced isotonic regression, the computation requires $O(n^2\log n)$ and $O(n^2)$ time, separately;

(SLS-O): for the stochastic lot sizing problem with $S = O(1)$ number of scenarios, the computation requires $O(n^2\log n)$ time; and

(CPM): for the joint inventory control and pricing problem with costly price markdown, when the demand functions are linear in prices, the computation requires $O(n^2)$ time.

3. Solving the Arborescence Problem Part I: Two Special Cost Structures

This section extends the Path problem to the more general Arborescence problem. We present two special cases that can be solved in polynomial time. For the same reason stated at the beginning of Section 2, we assume $x_0 = -\infty$ without loss of generality.

To begin with, we develop an optimality condition for the Arborescence problem, similar to that in Theorem 1. Recall that we define $\mathcal{T}(i)$ as the sub-arborescence of $\mathcal{T}$ rooted at the node i.

PROPOSITION 3. *Suppose $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is an optimal solution to the Arborescence problem and let $M \subseteq [n]$ denote the support of the vector $\boldsymbol{y}^*$. Then the optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ partitions $\mathcal{T}$ into several induced arborescences $\mathcal{T}^i = (N_{\mathcal{T}^i}, A_{\mathcal{T}^i})$ for all $i \in M$, where each $\mathcal{T}^i$ is rooted at node i and $N_{\mathcal{T}^i} = N_{\mathcal{T}(i)} \backslash (\cup_{j \in N_{\mathcal{T}(i)} \cap M, j \neq i} N_{\mathcal{T}(j)})$, and $A_{\mathcal{T}^i}$ are arcs of $A_{\mathcal{T}}$ restricted on $N_{\mathcal{T}^i}$. Let $\overline{x}^*_{\mathcal{T}^i} = \min\{x^* | x^* \in \arg\min \sum_{\ell \in N_{\mathcal{T}^i}} f_\ell(x)\}$ and $c_{\mathcal{T}^i} = \sum_{\ell \in N_{\mathcal{T}^i}} f_\ell(\overline{x}^*_{\mathcal{T}^i})$. We have that:*

(a) $x^*_\ell = \overline{x}^*_{\mathcal{T}^i}$ for all $\ell \in N_{\mathcal{T}^i}$, and

(b) $\overline{x}^*_{\mathcal{T}^i} < \overline{x}^*_{\mathcal{T}^j}$ for all $i \in M$ and $j \in N_{\mathcal{T}(i)} \cap M$.

The implication of Proposition 3 is similar to that of Theorem 1, indicating that the optimal solution for (Arborescence) partitions $\mathcal{T}$ into several small arborescences, with all nodes in any single one share the same x_i value, and x_i values for any two adjacent ones follows a strict inequality relation. In the case of the path graph, the DP algorithm defines its states based on all possible blocks $B(i,j)$

for $1 \le i \le j \le n$, resulting in $O(n^2)$ blocks/states. To compute each value-to-go function, we only need to compare the values of $O(n)$ possible preceding blocks. However, in the case of the arborescence graph, each block is no longer a set of consecutive indices, but rather an induced arborescence of $\mathcal{T}$, which can be exponentially many in the worst case. This implies that the DP recursion must compute value-to-go functions for exponentially many states. Furthermore, computing each value-to-go function requires comparing the values of all preceding induced arborescences, which are also exponentially many in the worst case. Thus, even though Proposition 3 shows a structural similarity between the Arborescence problem and the Path problem, a DP algorithm similar to (DP-original) may be infeasible due to the curse of dimensionality. Therefore, we focus on exploring special cases where computationally efficient algorithms can be developed.

3.1. Special Case I: Piece-wise Linear Convex Cost Functions

This subsection studies the case where f_i's are piecewise linear convex. Specifically, we assume that the following condition holds throughout this subsection.

Assumption 2. *For any $i \in [n]$, f_i is piece-wise linear convex with m_i number of kink points $\{k_1^i, \ldots, k_{m_i}^i\}$, where $k_1^i < \ldots < k_{m_i}^i$ with $m_i = O(1)$ such that each $f_i(x_i)$ can be queried in $O(1)$ time for any x_i.*

Many application problems, including problem (SLS-C), satisfy Assumption 2. Additionally, piecewise linear convex functions often serve as effective approximations for general convex functions.

When Assumption 2 holds, we can develop an efficient DP algorithm whose running time is polynomial in the total number of kink points across all f_i's. Specifically, the complexity depends on $\sum_{i\in[n]} m_i$, which is of order $O(n)$.

DP Recursion: From Theorem 3, we know that an optimal solution $(\boldsymbol{x^*}, \boldsymbol{y^*})$ for the Arborescence problem can partition $\mathcal{T}$ into several induced arborescences $\mathcal{T}^i = (N_{\mathcal{T}^i}, A_{\mathcal{T}^i})$ for all $i \in M$, where $\overline{x}^*_{\mathcal{T}^i} < \overline{x}^*_{\mathcal{T}^j}$ for all $i \in M$ and $j \in N_{\mathcal{T}(i)} \cap M$, and $x^*_\ell \equiv \overline{x}^*_{\mathcal{T}^i}$ for all $\ell \in N_i$ (recall that $x^*_{\mathcal{T}^i} = \min\{x^* | x^* \in \arg\min \sum_{\ell\in N_{\mathcal{T}^i}} f_\ell(x)\}$). Since f_i's are assumed to be piece-wise linear, we know that $\sum_{\ell\in N_{\mathcal{T}^i}} f_\ell(x)$ is also piece-wise linear convex with kink points $\bigcup_{\ell\in N_{\mathcal{T}^i}} \{k_1^\ell, \ldots, k_{m_\ell}^\ell\}$. Hence, $\overline{x}^*_{\mathcal{T}^i}$ can be one of the points from $\bigcup_{\ell\in N_{\mathcal{T}^i}} \{k_1^\ell, \ldots, k_{m_\ell}^\ell\}$. Taking all possible induced arborescences $\mathcal{T}^i$ containing node r into consideration, we know that $x^*_r \in \mathcal{K} = \bigcup_{\ell\in[n]} \{k_1^\ell, \ldots, k_{m_\ell}^\ell\}$, which is of $\sum_{\ell\in[n]} m_\ell = O(n)$ many, i.e., only linear in n.

Thus, this motivates us to design an efficient DP algorithm to optimally solve the Arborescence problem when f_i's are piece-wise linear convex functions. For each $i \in [n]$ and $x \in [\ell, u]$, let the value-to-go function $V(i,x)$ be defined as follows:

$$\begin{aligned} V(i,x) = \min_{(\boldsymbol{x},\boldsymbol{y})\in\mathbb{R}^{|\mathcal{T}(i)|}\times\{0,1\}^{|\mathcal{T}(i)|}} & \sum_{\ell\in N_{\mathcal{T}(i)}} K_\ell y_\ell + \sum_{\ell\in N_{\mathcal{T}(i)}} f_\ell(x_\ell) \\ \text{s.t.} \quad & (x_{p(\ell)} - x_\ell)(1-y_\ell) = 0, \quad \forall \ell \in N_{\mathcal{T}(i)}, \\ & x_{p(\ell)} \le x_\ell, \quad \forall \ell \in N_{\mathcal{T}(i)}, \\ & x_{p(i)} = x, \end{aligned} \tag{5}$$

which denotes the minimum cost of the isotonic optimization restricted to the sub-arborescence $\mathcal{T}(i)$, given that $x_{p(i)} = x$. Without loss of generality, we only need to consider the value of x to be k, where k belongs to the set $\{-\infty\} \cup \mathcal{K}$, since at optimality, $x^*_{p(i)}$ must take a value from $\{-\infty\} \cup \mathcal{K}$. We include $-\infty$ because x can only take the value $-\infty$ when $i = 1$. At the node i with $x_{p(i)} = k$, we have two choices: (i) keep x_i unchanged, setting $x_i = x_{p(i)} = k$, or (ii) increase the value of x_i to one element in $\mathcal{K}$ that is strictly greater than k. Recall that $\mathcal{C}(i)$ is defined as the set of all child nodes of i. In the first case, the value-to-go function is

$$V_\rightarrow(i,k) = f_i(k) + \sum_{j\in\mathcal{C}(i)} V(j,k).$$

In the second case, where x_i is strictly increased, the value-to-go function is

$$V_\uparrow(i,k) = \min_{k'\in\mathcal{K}:k'>k} \left\{ K_i + f_i(k') + \sum_{j\in\mathcal{C}(i)} V(j,k') \right\}.$$

Hence, the DP recursion can be recast as:

$$V(i,k) = \begin{cases} V_\uparrow(1,-\infty), & i = 1, k = -\infty, \\ \min\{V_\rightarrow(i,k), V_\uparrow(i,k)\}, & \forall i \in [n]\backslash\{1\}, k \in \mathcal{K}. \end{cases} \tag{DP-PL}$$

Note that $V(1,-\infty)$ equals the optimal value of the Arborescence problem. We next compute the overall complexity of (DP-PL). For each state (i,k), we need to first compute $V_\rightarrow(i,k)$ using $O(|\mathcal{C}(i)|)$ operations. Then we should compute $V_\uparrow(i,k)$, which needs us to compare $O(\sum_{i\in[n]} m_i)$ possible decisions, and evaluating each decision requires $O(|\mathcal{C}(i)|)$ operations. Therefore, the total computation across all (i,k) pairs is $\sum_{k'\in\mathcal{K}} \sum_{i\in[n]} O(\sum_{i\in[n]} m_i |\mathcal{C}(i)|) = O(n(\sum_{i\in[n]} m_i)^2)$. It is at most $O(n^3)$ under the assumption that $m_i = O(1)$ for all $i \in [n]$. We can further accelerate the DP algorithm to run in $O(n^2)$ time by adopting a similar approach as in Algorithm 1.

The Efficient Implementation: To accelerate the DP algorithm, note that for each $i \in [n]$, in the DP recursion (DP-PL), the computation of $V_\uparrow(i,k)$ for different $k \in \mathcal{K}$ can be efficiently implemented. Namely, we first sort all kink points from set $\{-\infty\} \cup \mathcal{K}$ in the descending order for each $i \in [n]$. For

each $i \in [n]$, by scanning through the sorted values and maintaining a running minimum over the scanned values, we can also calculate all $V(i,k)$'s together and save computation. Hence, the total time complexity is $O((n+\log(\sum_{i\in[n]} m_i))\sum_{i\in[n]} m_i)$. The detailed implementation can be found in Algorithm 2 in Appendix C. The result is summarized in the following proposition.

PROPOSITION 4. *Under Assumption 2, the Arborescence problem can be solved in* $O((n+\log(\sum_{i\in[n]} m_i))\sum_{i\in[n]} m_i)$ *time by efficiently implementing* (DP-PL)*, which is of order* $O(n^2)$ *if we assume* $m_i \equiv O(1)$ *for all* $i \in [n]$.

REMARK 3. Note that the problem (SLS-C) is a special case of the (Arborescence) problem, where each function f_i has only one kink point. According to Proposition 4, the problem (SLS-C) can be efficiently solved using Algorithm 2 with a time complexity of $O(n^2)$. This result recovers the best-known computational complexity shown in Guan (2011), but through a simpler algorithm and a more straightforward analysis.

REMARK 4. Proposition 4 also implies that for the Path problem, when the f_i's are piecewise linear with a constant number of kink points, the $O(n^2 \log n)$ complexity of Algorithm 1 can be further improved to $O(n^2)$ by adopting a more efficient alternative Algorithm 2.

3.2. Special Case II: General Convex Cost Functions with Dominating Fixed Costs in Parent Nodes

In this subsection, we study another special case of the Arborescence problem, where the fixed cost at each parent node is greater than or equal to the sum of all fixed costs in its child nodes. That is, we assume that

ASSUMPTION 3. *For each node* $i \in [n]$*, we have* $K_i \geq \sum_{j\in\mathcal{C}(i)} K_j$.

In (SLS-C), the condition that $K_i \geq K_j$ for any $i \in [n]$ and $j \in \mathcal{C}(i)$ is a generalization of the non-increasing fixed costs assumption in the path case, which is frequently made in inventory problems (Chen and Simchi-Levi 2004a,b).

This subsection shows that under Assumption 3, the Arborescence problem can be solved in polynomial time by characterizing the value-to-go functions as piecewise convex functions, each consisting of a few pieces. For simplicity, we assume that $f_i(x_i) \to \infty$ as $|x_i| \to \infty$ throughout this subsection. This assumption ensures that the optimal solution remains bounded and is satisfied by all the motivating examples introduced in the introduction section.

DP Formulation: Different from the previous special case, our value-to-go functions have continuous states; i.e., the state variable x in the function $V(i,x)$ in (5) is continuous. Therefore, the DP recursion can be expressed as:

$$\begin{aligned} V(i,x) &= \min_{x' \geq x} \left\{ K_i \mathbb{1}\{x' > x\} + U(i,x') \right\}, \\ U(i,x) &= f_i(x) + \sum_{j \in \mathcal{C}(i)} V(j,x) \end{aligned} \tag{6}$$

The value $V(1,-\infty)$ provides the optimal value. Since the value-to-go function has a continuous state x, its computation is generally difficult. We will show that $V(i,x)$ has a well-structured characterization using the concept called K-convexity, under the assumption that $K_i \geq \sum_{j \in \mathcal{C}(i)} K_j$ for any $i \in [n]$. More specifically, we show that $V(i,x)$ is a piecewise convex function in x with at most $O(|N_{\mathcal{T}(i)}|)$ convex pieces, and that its representation can be computed efficiently.

The notion of K-convexity dates back to Scarf et al. (1960), where it was used to demonstrate that the well-known (s,S) policy is optimal for a classical stochastic inventory problem with fixed costs. Formally, K-convexity is defined as follows:

Definition 2. A real-valued function f is called K-convex for $K \geq 0$ if, for any $x_0 \leq x_1$ and $\lambda \in [0,1]$,

$$f((1-\lambda)x_0 + \lambda x_1) \leq (1-\lambda) f(x_0) + \lambda f(x_1) + \lambda K.$$

Characterization of $V(i,x)$'s and the Optimal Policy: The following theorem characterizes the value-to-go function $V(i,x)$ as a piece-wise convex function with at most $|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|$ number of convex pieces (recall that $\mathcal{L}(i)$ is the leaf node set of the subtree $\mathcal{T}(i)$), which is of order $O(|N_{\mathcal{T}(i)}|)$, by leveraging its K_i-convexity property, for each $i \in [n]$. Intuitively, the piecewise convexity arises because, at each node, the cost-to-go function is defined as the minimum of two piecewise convex functions: one corresponding to keeping the value unchanged, and the other to increasing it and incurring a fixed cost. Each option contributes a distinct convex segment to the overall function. As we propagate upward through the tree, the DP recursion creates more and more new convex pieces of value-to-go functions. The K_i-convexity ensures each $V(i,x)$ introduces at most one more convex piece compared with $U(i,x)$. Consequently, the total number of convex pieces grows at most linearly with the number of nodes in the subtree.

Theorem 3. *Under Assumption 3, for each $i \in [n]$, the value-to-go function $V(i,x)$ defined in (5) satisfies the following three properties:*

(a) $V(i,x)$ is K_i-convex;

(b) there exists a pair of real numbers (s_i, S_i) satisfying

$$S_i \in \arg\min U(i,x), \tag{7a}$$

$$s_i \in \left\{ x | x \le S_i, U(i,x) = U(i,S_i) + K_i \right\}, \tag{7b}$$

such that an optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *to the Arborescence problem is*

$$x_i^* = \begin{cases} S_i, & x^*_{p(i)} \le s_i, \\ x^*_{p(i)}, & o.w. \end{cases}, \quad y_i^* = \begin{cases} 1, & x^*_{p(i)} \le s_i, \\ 0, & o.w; \end{cases}$$

(c) $V(i,x)$ *can be expressed as*

$$V(i,x) = \begin{cases} K_i + U(i,S_i), & x \le s_i, \\ U(i,x), & o.w.; \end{cases}$$

(d) $V(i,x)$ *is piece-wise convex with at most* $|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|$ *number of convex pieces.*

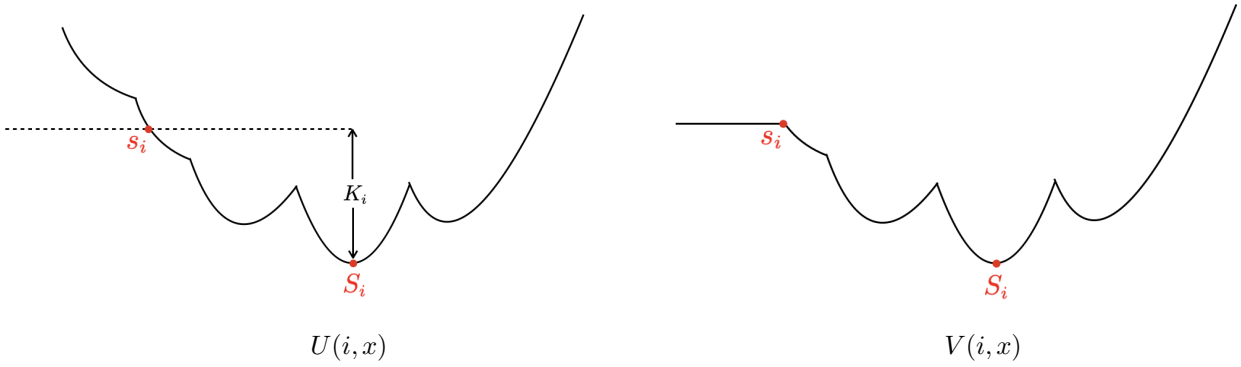

Figure 3 The changing dynamics from $U(i,x)$ to $V(i,x)$ when $K_i \ge \sum_{j \in \mathcal{C}(j)} K_j$

From Theorem 3(b), we know that computing an optimal solution to the Arborescence problem relies on determining the corresponding (s_i, S_i) thresholds. The resulting optimal policy resembles the classical (s_t, S_t) policy in stochastic inventory control (Scarf et al. 1960): the decision at each node i involves checking whether the value inherited from its parent node is below a threshold s_i; if so, the value is increased to S_i, otherwise it remains unchanged. We refer to this as the (s_i, S_i) policy. While such a policy is known to be optimal, computing the exact values of (s_i, S_i)'s is difficult in the traditional inventory setting due to the non-convexity of the value-to-go functions $V(i,x)$. However, for our setting, leveraging Theorem 3(d), which shows that each $V(i,x)$ is a piecewise convex function with at most $O(|N_{\mathcal{T}(i)}|)$ pieces, we can efficiently obtain the expression of each $V(i,x)$. Consequently, the corresponding (s_i, S_i) values can also be computed efficiently.

Algorithm and Complexity: We present an algorithm for obtaining the explicit functional representation of the value-to-go functions $V(i,x)$ and efficiently computing the optimal thresholds (s_i, S_i). According to Theorem 3(c), each $V(i,x)$ is a piece-wise convex function, which allows for a compact representation through a structured list. Specifically, each entry of this list corresponds to one convex

piece of $V(i,x)$ and includes relevant information about its functional form. Formally, we represent each function $V(i,x)$ through the following list structure:

$$\text{List}_i = [(-\infty, b_1^i, L_1^i, \mathcal{J}_1^i), (b_1^i, b_2^i, L_2^i, \mathcal{J}_2^i), \ldots, (b_{R_i-1}^i, \infty, L_{R_i}^i, \mathcal{J}_{R_i}^i)], \tag{8}$$

where the points $b_1^i \le b_2^i \le \ldots \le b_{R_i-1}^i$ represent the breakpoints that separate the convex segments of $V(i,x)$. We define the intervals $(-\infty, b_1^i], (b_1^i, b_2^i], \ldots, (b_{R_i-1}^i, \infty)$ to correspond directly to these convex segments. For notational convenience, we denote $b_0^i = -\infty$ and $b_{R_i}^i = \infty$. From Theorem 3(d), it follows that the number of convex pieces satisfies $R_i \le |N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|$. Additionally, each convex piece $r \in [R_i]$ is characterized by a scalar $L_r^i \in \mathbb{R}$ and an associated subset $\mathcal{J}_r^i \subseteq [n]$ which together completely describe the functional form of $V(i,x)$. Specifically, the value-to-go function $V(i,x)$ can be represented as follows:

$$V(i,x) = \begin{cases} L_1^i + \sum_{j\in\mathcal{J}_1^i} f_j(x), & x \in (-\infty, b_1^i], \\ L_2^i + \sum_{j\in\mathcal{J}_2^i} f_j(x), & x \in (b_1^i, b_2^i], \\ \cdots & \\ L_{R_i}^i + \sum_{j\in\mathcal{J}_{R_i}^i} f_j(x), & x \in (b_{R_i-1}^i, \infty). \end{cases} \tag{9}$$

We next introduce a proposition addressing three important aspects: (i) the validity of the representation provided in (9), (ii) an efficient method for constructing these piece-wise convex representations via DP recursion, and (iii) the computational complexity of obtaining the lists. A byproduct of the proof is that it directly provides an efficient algorithm for solving the Arborescence problem under the assumption $K_i \ge \sum_{j\in\mathcal{C}(i)} K_j$ for all $i \in N_{\mathcal{T}}$. The optimal thresholds (s_i, S_i) will also be computed, allowing a straightforward reconstruction of the optimal solution to the Arborescence problem. The DP computes a list representation (8) for each $V(i,x)$ in $O(|N_{\mathcal{T}(i)}|^2 \log(U/\varepsilon))$ time, and hence induces an overall $O(n^3 \log(U/\epsilon))$ time complexity of the algorithm.

PROPOSITION 5. *Suppose $K_i \ge \sum_{j\in\mathcal{C}(i)} K_j$ for any $i \in [n]$. Assume S_i defined by (7a) and s_i defined by (7b) are located on a one-dimensional ϵ-grid of the domain (i.e., a discretized grid over the domain with resolution $\epsilon > 0$) for any $i \in [n]$, then*

(a) There exists a list representation (8) for each $V(i,x)$, whose functional form can be recovered from the list representation according to (9).

(b) The optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ for the Arborescence problem can be obtained in $O(n^3 \log(U/\epsilon))$ time.

4. Solving the Arborescence Problem Part II: Special Arborescence Structures

This section analyzes different arborescence structures, such as balanced arborescences and a special class of unbalanced arborescences. We show that the Arborescence problem can be solved in

polynomial time for these special classes. We also discuss why the current analysis fails to extend to general unbalanced arborescences. Throughout this section, we also assume $x_0 = -\infty$ without loss of generality. To begin with, we formally define balanced and unbalanced arborescences. Recall that $\mathcal{L}$ is the set of all leaf nodes of the tree $\mathcal{T}$.

DEFINITION 3. Given an arborescence $\mathcal{T} = (N_\mathcal{T}, A_\mathcal{T})$ with graph size $|N_\mathcal{T}| = n$ and height $h = \max_{j\in\mathcal{L}} |\mathcal{P}(1,j)|$. Then $\mathcal{T}$ is called *balanced* if $h = O(\log n)$, and is called *unbalanced* if $h = \omega(\log n)$.

4.1. General Techniques

We first describe the general technique used in this section. Our main strategy is to demonstrate that the value-to-go function $V(i,x)$ specified by (5) is a piece-wise convex function for each node $i \in N_\mathcal{T}$. Consequently, for any given problem instance, if we can bound the number of convex pieces for each $V(i,x)$ by a polynomial function of n, then we can obtain the optimal solution $x_i^* = \arg\min_{x\geq x^*_{p(i)}} V(i,x)$ in polynomial time. This is done by comparing the values of $V(i,x)$ at $x^*_{p(i)}$ and at the minima of all convex segments of $V(i,x)$ greater than $x^*_{p(i)}$.

LEMMA 2. *The value-to-go function $V(i,x)$ is piece-wise convex for any $i \in N_\mathcal{T}$, regardless of the arborescence structure.*

The piece-wise convexity of $V(i,x)$ comes from an inductive structure. The function $U(i,x)$ is the sum of $V(j,x)$ for all child nodes $j \in \mathcal{C}(i)$, so it is piece-wise convex if each $V(j,x)$ is. Then, $V(i,x)$ is constructed from $U(i,x)$ by introducing new flat segments in the absence of the fixed cost, but preserves piece-wise convexity. Repeating this from the leaves up to the root proves the claim by induction. Having shown that each value-to-go function $V(i,x)$ is piece-wise convex, our next goal is to bound the number of convex pieces for each $V(i,x)$ by poly(n). In Section 3.2, we showed that under the assumption $K_i \geq \sum_{j\in\mathcal{C}(i)} K_j$ for each $i \in N_\mathcal{T}$, the number of convex pieces for each $V(i,x)$ is bounded by $|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)| = O(n)$, where we recall that $\mathcal{L}(i)$ is the set of leaf nodes in $\mathcal{T}(i)$. In this section, we extend our analysis and demonstrate that, even without this assumption, the number of convex pieces for $V(i,x)$ remains bounded by poly(n) for several classes of arborescence graphs.

We now introduce the procedure for upper-bounding the number of convex pieces of $V(i,x)$ according to its DP recursion (6), assuming that each child node's value-to-go function $V(j,x)$, for $j \in \mathcal{C}(i)$, has at most C_j convex pieces:

- **Summing piece-wise convex functions:** We first analyze the number of convex pieces in $U(i,x)$. Given that each $V(j,x)$ has at most C_j convex pieces, the number of breakpoints separating different convex segments in each $V(j,x)$ is $C_j - 1$. Since the set of breakpoints for $\sum_{j\in\mathcal{C}(i)} V(j,x)$ is the union of breakpoints from each $V(j,x)$ for all $j \in \mathcal{C}(i)$, the total number of breakpoints in $\sum_{j\in\mathcal{C}(i)} V(j,x)$ is at most $\sum_{j\in\mathcal{C}(i)}(C_j - 1)$. Consequently, $\sum_{j\in\mathcal{C}(i)} V(j,x)$ has at most $\sum_{j\in\mathcal{C}(i)}(C_j -$

$1)+1 \le \sum_{j\in\mathcal{C}(i)} C_j$ convex pieces. Additionally, since $f_i(x)$ is convex, the function $U(i,x) = f_i(x) + \sum_{j\in\mathcal{C}(i)} V(j,x)$ is also piece-wise convex, having at most $\sum_{j\in\mathcal{C}(i)} C_j$ convex pieces.

- **Cutting out new convex pieces:** Next, we analyze the number of convex pieces in $V(i,x)$. From the proof of Lemma 2, we observe that $V(i,x)$ is obtained from $U(i,x)$ by introducing new flat segments. These flat segments correspond to intervals in which increasing x is preferable to keeping it unchanged, leading to the addition of new convex pieces. To bound the total number of convex pieces of $V(i,x)$, we must restrict the number of these newly introduced flat segments. We will subsequently show that bounding these segments can equivalently be achieved by bounding the number of "quasi-convex" segments in $U(i,x)$, a notion we introduce next. To do so, let us first define a function $f:[a,b]\to\mathbb{R}$ to be *weakly unimodal* if there exists a value x^* for which it is non-increasing for $a \le x \le x^*$ and non-decreasing for $b \ge x \ge x^*$, and x^* is called the *mode* of f.

Definition 4. Suppose $g:\mathbb{R}\to\mathbb{R}$ is a piece-wise convex function with convex pieces $(-\infty, c_1]$, $(c_1,c_2],\ldots,(c_{L-1},\infty)$. Define the quasi-convex pieces of g as $(-\infty,\widetilde{c}_1]$, $(\widetilde{c}_1,\widetilde{c}_2],\ldots,(\widetilde{c}_{R-1},\infty)$, which should satisfy the following requirements:

(a) $\{\widetilde{c}_1,\widetilde{c}_2,\ldots,\widetilde{c}_{R-1}\}\subseteq\{c_1,c_2,\ldots,c_{L-1}\}$;

(b) g is weakly unimodal in each quasi-convex piece; and

(c) g is no longer weakly unimodal in the union of any two consecutive quasi-convex pieces.

From Definition 4(a), we know that for any piece-wise convex function g, each quasi-convex piece is a union of several consecutive convex segments. Definition 4(b) implies that within each quasi-convex piece, the function g is non-increasing to the left and non-decreasing to the right of some local minimum (mode). Moreover, Definition 4(c) ensures minimality in the construction of quasi-convex pieces: any combination of two consecutive quasi-convex pieces cannot form a larger quasi-convex piece. With these properties, we now present the following lemma.

Lemma 3. *Suppose g is any piece-wise convex function with $L\in\mathbb{Z}_+$ number of convex pieces and $R\in\mathbb{Z}_+$ number of quasi-convex pieces, where $R\le L$. Then, for any $K\in\mathbb{R}_+$, the function $h:\mathbb{R}\to\mathbb{R}$ defined by*

$$h(x) = \min_{x'\ge x}\left\{K\mathbb{1}\{x'>x\} + g(x')\right\}$$

is piece-wise convex with at most $L+R$ number of convex pieces.

Readers can refer to Figure 4 for a visualization of how newly flat pieces of $h(x)$ are introduced from $g(x)$. The number of newly introduced flat pieces is bounded by "peaks" of $g(x)$, which is equal to the number of quasi-convex pieces. With the definitions and lemmas introduced above, we now bound the number of convex pieces of each $V(i,x)$ by poly(n) for several special classes of arborescence structures.

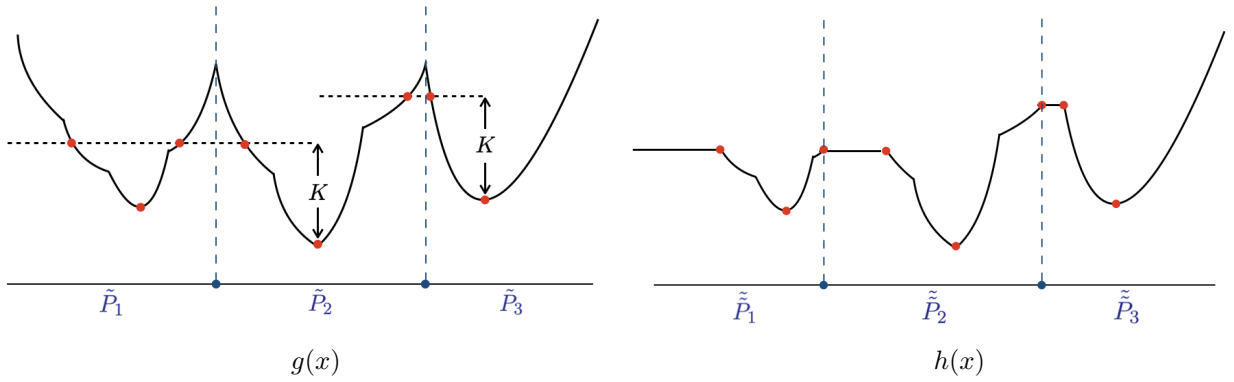

Figure 4 Illustration of Lemma 3, where $\widetilde{P}_1, \widetilde{P}_2, \widetilde{P}_3$ are quasi-convex pieces of $g(x)$ and $\widetilde{\widetilde{P}}_1, \widetilde{\widetilde{P}}_2, \widetilde{\widetilde{P}}_3$ are quasi-convex pieces of $h(x)$

4.2. Balanced Arborescences

We first consider balanced arborescences, i.e., arborescences with height $h = O(\log n)$. To bound the number of convex pieces of each $V(i,x)$ by $\text{poly}(n)$, we introduce a useful lemma applicable to general arborescence structures. For ease of notation, we define the relative height of node $i \in N_{\mathcal{T}}$ in the arborescence $\mathcal{T}$ as $h(i) = \max_{j \in \mathcal{L}(i)} |\mathcal{P}(i,j)|$ (recall that $\mathcal{P}(i,j)$ denotes a path from node i to node j), i.e., the height of the sub-arborescence $\mathcal{T}(i)$ rooted at node i is defined as the longest path from node i to a leaf node.

LEMMA 4. *The value-to-go function $V(i,x)$ is piece-wise convex with at most $2^{h(i)}|\mathcal{L}(i)|$ number of convex pieces, regardless of the arborescence structure.*

When moving one level up in the tree, the number of new convex pieces introduced in $V(i,x)$ relative to $U(i,x)$ is at most the number of convex pieces in $U(i,x)$ as can be seen from Lemma 3 and DP recursion (6), which is itself bounded by the total number of convex pieces from the value-to-go functions $V(j,x)$'s of all the child nodes $j \in \mathcal{C}(i)$. This implies that each level can, at most, double the total number of convex pieces. Starting with at most two convex pieces per leaf node, this leads to an overall bound of $2^{h(i)}|\mathcal{L}(i)|$. The next proposition follows directly from the fact that all balanced arborescences have a height on the order of $O(\log n)$, implying that the Arborescence problem can be solved in polynomial time for balanced arborescences.

PROPOSITION 6. *Given a balanced arborescence $\mathcal{T}$, the number of convex pieces of $V(i,x)$ can be bounded by* $\text{poly}(n)$ *for each $i \in N_{\mathcal{T}}$.*

Proposition 6 may not be directly applicable to unbalanced arborescences since $2^{h(i)}|\mathcal{L}(i)|$ may fail to be polynomially bounded if the height $h(i)$ is of order strictly higher than $\log n$. The main reason behind this fact is the overly conservative use of Lemma 3, where we bound the number of quasi-convex pieces of $U(i,x)$ only by its number of convex pieces. This simplification leads to a

potential doubling of the number of convex pieces of $V(i,x)$ at each node compared to that of $U(i,x)$. However, as illustrated in Section 2, the Arborescence problem remains solvable in polynomial time for any path graph– a special case of an unbalanced arborescence– via a simple DP algorithm. Yet, our current analytical framework fails to capture this unbalanced structure.

4.3. Unbalanced Arborescence with Constant Rounds of Branching

This subsection aims to show a polynomial bound on the number of convex pieces of each $V(i,x)$ for a special class of unbalanced arborescences. To accomplish this goal, we require a more refined analysis. Given a piece-wise convex function $f:\mathbb{R}\to\mathbb{R}$, we call $x\in\mathbb{R}$ a non-smooth point of f if the left and right derivatives of f at x, denoted by $f'_-(x)$ and $f'_+(x)$, are different, i.e., f is not differentiable at x. Note that the existence of the left and right derivatives is ensured by piece-wise convexity. Based on that, we first introduce an assumption that will be used throughout this subsection.

Assumption 4. *The convex function f_i has only a constant number of non-smooth points for each $i\in N_{\mathcal{T}}$.*

Note that each convex function f_i can be regarded as a piece-wise smooth function, with its non-smooth points serving as breakpoints of the smooth pieces. It should be noted that according to our assumption on $O(1)$ revaluation complexity of f_i at any feasible point, within each smooth convex segment, f_i must be able to be evaluated in $O(1)$ time for any point within that segment. However, the functional expressions differ across distinct smooth convex segments. Evaluating f_i at any feasible point requires first identifying the appropriate smooth convex segment. This identification can be performed in constant time if and only if the number of smooth convex segments (equivalently, the number of non-smooth points) is constant. Thus, Assumption 4 aligns with our previous assumption of $O(1)$ evaluation complexity of f_i at any feasible point.

Next, rather than bounding the number of quasi-convex pieces of a piece-wise convex function g simply by the number of its convex pieces, we conduct a more refined analysis to develop a tighter bound. In the analysis, we leverage the fact that g is weakly unimodal within each quasi-convex piece and that bounding the number of quasi-convex pieces of g becomes equivalent to bounding the number of modes of g. We categorize all quasi-convex pieces of g into two distinct types, named *Type I* and *Type II*, and estimate their corresponding numbers of quasi-convex pieces separately. Correspondingly, the modes associated with those two types of quasi-convex pieces are termed *Type I* mode and *Type II* mode.

Definition 5. Suppose $g:\mathbb{R}\to\mathbb{R}$ is a piece-wise convex function with $L\in\mathbb{Z}_+$ convex pieces on $P_1=(-\infty,c_1]$, $P_2=(c_1,c_2],\ldots,$ and $P_L=(c_{L-1},\infty)$. Then, any of its quasi-convex pieces, say $\widetilde{P}_{ij}=\cup_{l=i}^{j}P_l$ with some $1\le i\le j\le L$, can be categorized into:

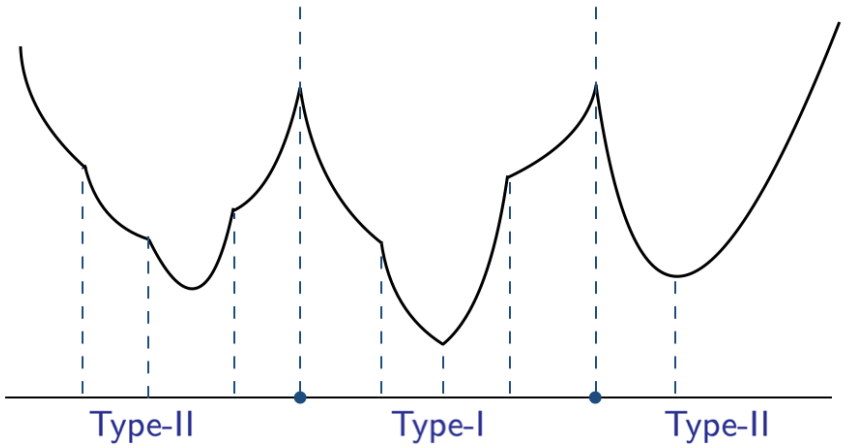

Figure 5 Illustration of Type I and Type II modes. Different convex pieces are separated by dashed lines beneath the curve, while different quasi-convex pieces are separated by dashed lines that intersect the curve. From left to right: Type II (mode in the interior of a convex piece), Type I (mode at a breakpoint with unequal one-sided slopes), and Type II (mode at a breakpoint where both one-sided slopes are zero).

(a) *Type I* if the minimum of g in $\widetilde{P}_{ij}$ is uniquely achieved at c_ℓ for some $\ell \in [L-1]$ with $g'_-(c_\ell) < g'_+(c_\ell)$; and

(b) *Type II* otherwise, i.e., there exists a minimum of g in $\widetilde{P}_{ij}$ achieved at the interior of P_ℓ for some $\ell \in [L]$, or at c_ℓ for some $\ell \in [L-1]$ with $g'_-(c_\ell) = g'_+(c_\ell) = 0$.

Remark 5. We give two remarks on the above classification:

- The classification of Type I and Type II quasi-convex pieces depends on the breakpoints one specifies for those convex pieces of g. Once the breakpoints are fixed, the classification is unique. For instance, the middle Type II quasi-convex piece in Figure 5 could just as well be labeled Type I, provided the breakpoint at its mode is not regarded as the boundary between two distinct convex pieces.
- If g is non-smooth at any breakpoint of its convex pieces, i.e., c_ℓ for any $\ell \in [L-1]$, then the minimum of any Type II quasi-convex piece of g will only take at the interior of P_ℓ for some $\ell \in [L]$. Thus, g possesses no mode corresponding to the rightmost Type II quasi-convex piece.

From Lemma 2, we know that each $V(i,x)$ $(U(i,x))$ is piecewise convex and can be partitioned into several quasi-convex pieces. Using Definition 5, we can classify the mode of each quasi-convex piece of $V(i,x)$ $(U(i,x))$ into one of the types introduced above. We characterize the quasi-convex pieces into different types, as the number of them should be bounded using different techniques. The rationale behind this classification and the corresponding bounding approaches will be formally discussed shortly. We can use the following simple example as a preliminary illustration.

EXAMPLE 1. Consider a path $\mathcal{P}$ with $N_{\mathcal{P}} = \{1,2\}$ and $A_{\mathcal{P}} = \{(1,2)\}$. Let $K_1 = K_2 = 1$; $f_1(x) = 3|x+1|, f_2(x) = x^2$. Since $U(2,x) = f_2(x)$ is convex and smooth, it contains only one quasi-convex piece, which is of Type II and has its mode achieved at the global minimizer $x = 0$ with zero derivative. Hence, we can obtain that

$$V(2,x) = \begin{cases} 1, & x \le -1, \\ x^2, & x > -1, \end{cases}$$

with a newly introduced flat piece on the left and a breakpoint $x = -1$. It should be noted that $V(2,x)$ is non-smooth at $x = -1$ with $V_-'(1,-1) = 0 > -2 = V_+'(1,-1)$. However, after adding $f_1(x)$, we would get

$$U(1,x) = \begin{cases} -3x - 2, & x \le -1, \\ x^2 + 3x + 3, & x > -1, \end{cases}$$

which contains only one quasi-convex piece with mode $x = -1$. Note that $U(1,x)$ is non-smooth at $x = -1$ with $U_-'(1,-1) = -3 < 1 = U_+'(1,-1)$, where the signs of directional derivatives on the two sides of -1 is reversed from that of $V(2,x)$, which is caused by the non-smooth point $x = -1$ of $f_1(x)$. Hence, $U(1,x)$ has only a single quasi-convex piece which is of Type I and is associated with a mode at -1. Finally, we obtain

$$V(1,x) = \begin{cases} 2, & x \le -4/3, \\ -3x - 2, & -4/3 < x \le -1, \\ x^2 + 3x + 3, & x > -1. \end{cases}$$

This example illustrates that Type I modes of $V(i,x)$ are tied to non-smooth points introduced by some f_j and can emerge when directional derivative signs reverse across a breakpoint. If a Type II mode of $V(i,x)$ is located at some smooth convex piece, it must be a stationary point. Those characteristics indicate that to bound Type I and Type II convex pieces of $V(i,x)$, we only need to bound its stationary points and all non-smooth points introduced by f_j's, respectively. We begin our formal analysis with the following lemma.

LEMMA 5. *For any $i \in N_{\mathcal{T}}$, let the convex pieces of $V(i,x)$ be $P_1^i = (-\infty, c_1^i], P_2^i = (c_1^i, c_2^i], \ldots, P_{L_i}^i = (c_{L_i-1}^i, \infty)$, with $L_i \in \mathbb{Z}_{++}$ being the number of convex pieces of $V(i,x)$. Denote $\mathcal{E}_j \subseteq \mathbb{R}_+$ as the set of non-smooth points of $f_j(x)$ for any $j \in N_{\mathcal{T}}$. Then,*

(a) For any $c \in \{c_1^i, c_2^i, \ldots, c_{L_i-1}^i\} \setminus \cup_{j \in N_{\mathcal{T}(i)}} \mathcal{E}_j$, it holds that $V_-'(i,c) \ge V_+'(i,c)$.

(b) For any $\ell \in [L_i]$, there exists a constant $K_\ell^i \in \mathbb{R}$ and a set of nodes $\mathcal{D}_\ell^i \subseteq N_{\mathcal{T}(i)}$, such that $V(i,x) \equiv K_\ell^i + \sum_{j \in \mathcal{N}_\ell^i} f_j(x)$ for x in convex piece P_ℓ^i, where $\mathcal{N}_\ell^i = \cup_{k \in \mathcal{D}_\ell^i} \mathcal{P}(i,k)$.

The first statement of the lemma implies that the mode of any Type I quasi-convex piece of $V(i,x)$ must coincide with some non-smooth point of $f_i(x)$, i.e., some $c \in \{c_1^i, c_2^i, \ldots, c_{L_i-1}^i\}$ where $V_-'(i,c) < V_+'(i,c)$. The second statement of the lemma provides a structural characterization of $V(i,x)$ over each convex piece: the constant captures the height of the most recently introduced flat piece, while

the $f_j(x)$ terms record the cost functions added later in the dynamic-programming recursion. We can illustrate it using $V(1,x)$ in Example 1: when $x \le -4/3$, $V(1,x) = 2$; when $-4/3 < x \le -1$, $V(1,x) = 1 + f_1(x)$; when $x > -1$, $V(1,x) = f_1(x) + f_2(x)$. These insights underpin the key lemma that follows.

LEMMA 6. *For any $i \in N_{\mathcal{T}}$, consider any quasi-convex piece (say, $\widetilde{P}$) of $V(i,x)$, we have that any mode of this quasi-convex piece (say, $m_{\widetilde{P}} \in \arg\min_{x\in\widetilde{P}} V(i,x)$) satisfies:*

(a) $m_{\widetilde{P}} \in \cup_{j\in N_{\mathcal{T}(i)}} \mathcal{E}_j$ where $\mathcal{E}_j$ is the set of non-smooth points of $f_j(x)$, if $\widetilde{P}$ is of Type I; or

(b) $m_{\widetilde{P}} \in \arg\min_x \sum_{j\in\mathcal{N}^i} f_j(x)$ where $\mathcal{N}^i = \cup_{k\in\mathcal{D}^i} \mathcal{P}(i,k)$ for some $\mathcal{D}^i \subseteq N_{\mathcal{T}(i)}$, if $\widetilde{P}$ is of Type II.

It can be seen that Lemma 6 follows directly from Lemma 5, which characterizes the mode based on the type of quasi-convex piece it belongs to. Note that both Lemma 5 and Lemma 6 also hold when we replace $V(i,x)$ by $U(i,x)$. Such a characterization is critical for bounding the number of quasi-convex pieces of $U(i,x)$, and consequently, to bound the number of convex pieces in $V(i,x)$ for each $i \in N_{\mathcal{T}}$. Next, we separately derive upper bounds for the number of *Type I* and *Type II* quasi-convex pieces of $U(i,x)$, as detailed below:

- The modes of *Type I* quasi-convex pieces of $U(i,x)$ can only appear within the set $\cup_{j\in N_{\mathcal{T}(i)}} \mathcal{E}_j$. Under Assumption 4, we have $|\cup_{j\in N_{\mathcal{T}(i)}} \mathcal{E}_j| \le \sum_{j\in N_{\mathcal{T}(i)}} |\mathcal{E}_j| = O(|N_{\mathcal{T}(i)}|)$. Therefore, the number of *Type I* quasi-convex pieces is always bounded by $O(n)$.
- The number of *Type II* quasi-convex pieces of $U(i,x)$ is bounded by the number of all possible sets $\mathcal{N}^i$, i.e., all possible node sets of induced sub-arborescences in $\mathcal{T}$ rooted at node i. In the worst case, this can be exponentially large. Thus, bounding the number of *Type II* quasi-convex pieces by a polynomial function of n is generally difficult, though possible for special arborescences. For instance, when $\mathcal{T}$ is a path, the number of all possible sets $\mathcal{N}^i$ is simply $n - i + 1$, providing an alternative perspective on why the Path problem is polynomial-time solvable, as shown in Section 2. Our goal, however, is to extend beyond paths and address a broader class of unbalanced arborescences. This is achieved by observing that when all non-smooth points $\cup_{j\in N_{\mathcal{T}(i)}} \mathcal{E}_j$ are no longer considered as each of them has been counted for a *Type I* quasi-convex piece, bounding by the number of all possible sets $\mathcal{N}^i$ can be refined by bounding all stationary points of $U(i,x)$.

To bound the number of *Type II* quasi-convex pieces of $U(i,x)$ for a broader class of unbalanced arborescences, we introduce the following key lemma.

LEMMA 7. *Let $\mathcal{P} := \mathcal{P}(1,j)$ with some $j \in \mathcal{L}$, through which we re-index the nodes in $\mathcal{P}$ from the root to the leaf as $1, 2, \ldots, |\mathcal{P}|$. For each $\ell \in \cup_{k\in\mathcal{P}} \mathcal{C}(k)\backslash\mathcal{P}$, denote the number of convex pieces of $V(\ell,x)$ by m_ℓ. Under Assumptions 4, for any $i \in \mathcal{P}$, the number of convex pieces of $V(i,x)$ is upper bounded by $O(n^2) + n^2 \sum_{\ell\in\cup_{k\in\mathcal{P}}\mathcal{C}(k)\backslash\mathcal{P}} m_\ell$.*

The above lemma shows that for any node i on the "spine" of the arborescence (i.e., any selected path within the tree), the number of convex pieces in $V(i,x)$ is bounded by the total number of convex pieces in all $V(j,x)$'s for child nodes j's branching off the spine, multiplied by an n^2 factor. Such a factor arises as follows. Partition $\mathbb{R}$ into at most n disjoint regions, with each being the union of the intervals of newly introduced flat pieces created while evaluating $V(i,x)$ for some $i \in \mathcal{P}$. In any such region, the number of convex (hence quasi-convex) pieces inherited from off-spine children is at most $\sum_{\ell \in (\cup_{k\in\mathcal{P}} \mathcal{C}(k))\setminus\mathcal{P}} m_\ell$. As we sweep along the spine and evaluate $V(i,x)$ for each $i \in \mathcal{P}$ (at most n times), a region can introduce at most one time of convex pieces inherited from off-spine children per level, i.e., an additional factor of n; across all regions, this yields the claimed n^2 multiplier. We illustrate this intuition using the following example. This result is crucial for bounding the number of convex pieces in $V(i,x)$ across a broad class of arborescences with bounded branching depth, which we will define formally in a second.

Before proceeding, we note that Lemma 7 extends to additional useful settings. For each $i \in \mathcal{P}$, one can regard $f_i(x) + \sum_{\ell\in\mathcal{C}(i)\setminus\mathcal{P}} V(\ell,x)$ as a single piecewise-convex function. The quadratic-growth conclusion in Lemma 7 then applies verbatim to the Path problem with piecewise-convex stage costs.

Corollary 2. *Consider the Path problem, where at each node i, the cost f_i is piecewise convex with m_i convex pieces instead of simply convex. Besides, assume each convex piece of $f_i(x)$ is smooth, and at every breakpoint c, the one-sided derivatives satisfy $(f_i)'_-(c) > (f_i)'_+(c)$. Then, for any $i \in [n]$, the value function $V(i,x)$ is piecewise convex in x, and its number of convex pieces is bounded by $O(n^2) + n^2 \sum_{i=1}^n m_i$.*

k-round-branching Arborescences: We consider k-round-branching arborescences as a structured subclass in which the arborescence can be decomposed into k hierarchical levels, formally defined below. Although any arborescence can be represented in this form for some k, focusing on instances with bounded k captures many practically relevant structures and allows us to show polynomial bounds on the number of convex pieces in the corresponding value-to-go functions.

Definition 6. An arborescence $\mathcal{T}$ is called a *k-round-branching arborescence* ($k \geq 1$) if it can be constructed via the following process: In the first round, starting from the root node, we generate a single directed path, called the 1-*round branch*, in which each arc points away from the root; all nodes along this path are termed 1-*round nodes*. If $k \geq 2$, then for each subsequent round ℓ (where $2 \leq \ell \leq k$), we select certain $(\ell-1)$-*round nodes* and generate one or more directed paths (referred to as ℓ-*round branches*), each starting from an $(\ell-1)$-round node and extending outward. Nodes on these ℓ-round branches, excluding the starting $(\ell-1)$-round nodes, are designated as ℓ-*round nodes*. After completing k rounds, the resulting arborescence $\mathcal{T}$ is obtained.

EXAMPLE 2. For an arborescence $\mathcal{T}$, if its underlying undirected graph is

(a) a path, then it is a 1-round-branching arborescence;

(b) a star, then it is a 2-round-branching arborescence; and

(c) a caterpillar tree (i.e., a tree in which all the vertices are within distance 1 of a central path), then it is a 2-round-branching arborescence.

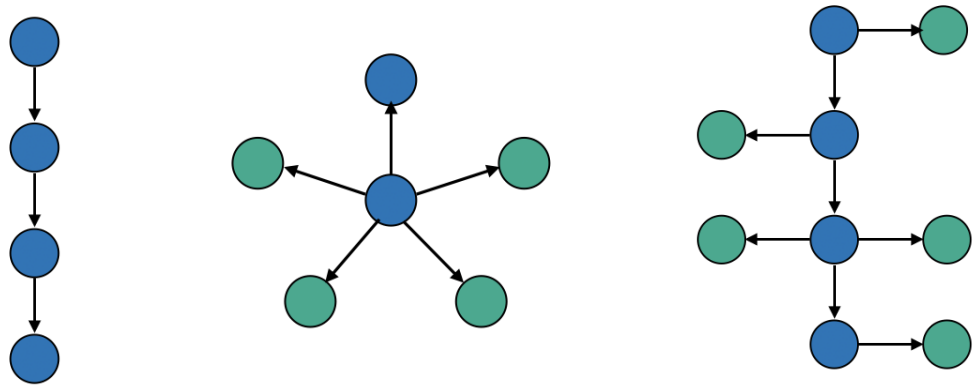

Figure 6 Illustration of three arborescent graphs: path, star, and caterpillar tree (from left to right). The 1-round-nodes are colored blue, while the 2-round-nodes are colored green.

Finally, we arrive at our main result for a k-round-branching arborescence. Particularly, Theorem 4 follows by applying Lemma 7 iteratively to each branching round of the arborescence, and concludes that the total number of convex pieces of each $V(i,x)$ grows by at most $O(n^2)$ after each round of branching.

THEOREM 4. *If the graph $\mathcal{T}$ is a k-round-branching arborescence, then the number of convex pieces of $V(i,x)$ for each $i \in N_{\mathcal{T}}$ is $O(n^{2k})$.*

The following corollary follows directly from Theorem 4.

COROLLARY 3. *If $\mathcal{T}$ is a k-round-branching arborescence with $k = O(1)$, then the number of convex pieces of $V(i,x)$ for each $i \in N_{\mathcal{T}}$ is* poly(n).

5. Numerical Results

Experimental Setup: To evaluate the computational performance of proposed DP algorithms, we perform three numerical experiments involving different types of problem instances: (i) quadratic cost functions with path structures, (ii) quadratic cost functions with general arborescence structures, and (iii) piece-wise linear convex cost functions with general arborescence structures. The baseline for

comparison is the commercial solver Gurobi. Specifically, for each $i \in [n]$, we linearized the bilinear constraint $(x_i - x_{p(i)})(1-y_i)$ using the big-M coefficient M_i as $x_i - x_{p(i)} \le M_i y_i$, where M_i is selected as $\max_{i\in[n]} x^*_{i,i} - \min_{i\in[n]} x^*_{i,i}$, i.e., the largest possible increase in x between any two consecutive indexes. We set the computational time limit for Gurobi as 30 minutes. For each class of instances, we implement the algorithm with the best theoretical time complexity developed in this paper and compare it against Gurobi. All the experiments are conducted on a MacBook Pro with an Apple M2 Chip and 8 GB RAM. The Gurobi solver we use is of version 12.0.0 build v12.0.0rc1. The detailed experimental setups are described below:

- *Quadratic & path*: We compare two algorithms in this paper with the baseline: the DP without acceleration given by (DP-original) (with time complexity $O(n^3)$) here, which we denote by DP_1; and the DP with acceleration given by Algorithm 1 (with time complexity $O(n^2)$, the best algorithm under this setting in the paper), which we denote by DP_2. We test for $n \in \{500, 1000, 1500, 2000, 2500, 3000\}$, and for any particular n, we generate 10 random instances as follows: $K_i \overset{\text{i.i.d.}}{\sim} \text{Unif}(0, 60)$; $f_i(x) = a_i x^2 - 2b_i x$, in which a_i's form a decreasingly ordered sequence of n random numbers generated from $\text{Unif}(0, 20)$, and $b_i = \widehat{b}_i + \epsilon_i$ where $\widehat{b}_i$'s form an increasingly ordered sequence of n random numbers generated from $\text{Unif}(0, 20)$ and $\epsilon_i \overset{\text{i.i.d.}}{\sim} \text{Unif}(-5, 0)$. We generate a_i, b_i in such a way because the minimum of $f_i(x)$ is attained at b_i/a_i, which is isotonic concerning i if $\epsilon_i = 0$. ϵ_i are added to prevent b_i/a_i from being isotonic, but make them "almost" isotonic. For Gurobi, the average running time and the average gap are reported; for DP_1 and DP_2, the average running time is reported, both in Table 1.
- *Quadratic & arborescence*: We compare the DP algorithm given by Section 3.2 and Section 4 with the baseline. It is implemented by tracking the functional forms of piece-wise convex value-to-go functions. Although the number of convex pieces of value-to-go functions can only be bounded polynomially in certain cases, we generate random instances in a general setting. We test for $n \in \{500, 1000, 1500, 2000, 2500, 3000\}$, and for any particular n, we generate 10 random instances as follows: the arborescence is generated by uniformly assigning node i as a child of one of $1, \ldots, i-1$; $K_i \overset{\text{i.i.d.}}{\sim} \text{Unif}(0, 60)$; $f_i(x) = a_i x^2 - 2b_i x$, in which a_i's form a decreasingly ordered sequence of n random numbers generated from $\text{Unif}(0, 20)$, and b_i's form an increasingly ordered sequence of n random numbers generated from $\text{Unif}(0, 20)$ (here we do not need to add noises on b_i to make b_i/a_i "almost" isotonic). For Gurobi, the average running time and the average gap are reported; for DP, the average running time and the average maximum number of convex pieces among all value-to-go functions are reported, both in Table 2.
- *Piece-wise Linear & arborescence*: We compare the efficient implementation of the DP recursion (DP-PL) with the baseline, which is shown to have the complexity of $O(n^2)$ by Proposition 4. It is the algorithm with the best complexity designed under this setting for both the path graph and

the arborescence graph. To conduct a unified experiment, we only test for the arborescence case given the page limit. The problem we choose is the scenario tree-based stochastic lot sizing model given by (SLS-C). We test for $n \in \{500, 1000, 1500, 2000, 2500, 3000\}$, and for any particular n, we generate 10 random instances as follows: $K_i \overset{\text{i.i.d.}}{\sim} \text{Unif}(0, 60)$; $h_i \overset{\text{i.i.d.}}{\sim} \text{Unif}(0, 10)$; $b_i \overset{\text{i.i.d.}}{\sim} \text{Unif}(0, 10)$; $d_i \overset{\text{i.i.d.}}{\sim} \text{Unif}(5, 20)$. For Gurobi, the average running time and the average gap are reported; for the DP algorithm, the average running time is reported, both in Table 3.

Table 1 Quadratic & Path

	Gurobi		DP_1	DP_2
n	time (s)	gap (%)	time (s)	time (s)
500	1269.24	0.02	10.99	0.67
1000	1800.00	0.08	87.53	2.68
1500	1800.00	0.08	301.63	7.17
2000	1800.00	0.08	695.95	10.62
2500	1800.00	0.08	1389.40	16.90
3000	1800.00	0.06	1800.00+	24.13

Table 2 Quadratic & Arborescence

	Gurobi		DP	
n	time (s)	gap (%)	time (s)	# of maximum convex pieces
500	1800.00	5.20	0.19	164.50
1000	1800.00	4.66	0.54	310.90
1500	1800.00	4.93	1.37	465.80
2000	1800.00	5.10	2.02	623.40
2500	1800.00	5.89	3.01	773.10
3000	1800.00	6.04	4.07	921.20

Implications: The following managerial insights can be obtained from the numerical results.

- *Quadratic & path*: Gurobi requires significant running time and consistently fails to terminate within the 30-minute limit for instances where $n \geq 1000$, although the optimality gap is typically small. In contrast, both DP_1 and DP_2 successfully identify optimal solutions within the given time

Table 3 Piece-wise Linear & Arborescence

	Gurobi		DP
n	time (s)	gap (%)	time (s)
500	156.88	0.00	1.04
1000	1800.00	0.53	4.27
1500	1800.00	0.66	9.70
2000	1800.00	1.16	17.15
2500	1800.00	1.22	27.80
3000	1800.00	1.24	40.64

limit in most instances. When comparing the two DP algorithms, DP_2 consistently outperforms DP_1, with the advantage becoming increasingly obvious as the instance size n grows– achieving up to nearly a hundredfold reduction in runtime. This numerical performance aligns with and validates the theoretical complexity results shown in our analysis.

- *Quadratic & arborescence*: Gurobi is unable to terminate within the 30-minute time limit for all test instances, resulting in relatively large optimality gaps. In contrast, the DP algorithm performed exceptionally well, solving all instances within five seconds. Part of this efficiency stems from the quadratic cost structure of our problem setup, which allows for the explicit analytical computation of breakpoints rather than relying on binary search. The remarkable computational advantage is also attributed to the limited growth of convex segments in the value-to-go functions. Although we are unable to theoretically prove a polynomial bound on the number of convex segments in the general setting, our experiments indicate that this number increases nearly linearly with respect to n for randomly generated instances. This empirical finding demonstrates that our proposed DP algorithm performs robustly and efficiently on random instances.
- *Piece-wise linear & arborescence*: Gurobi fails to terminate within the 30-minute time limit when $n \geq 1000$, with the optimality gap growing significantly as the problem size increases. In contrast, the proposed DP algorithm identifies optimal solutions within just a few seconds. Moreover, the observed computational time scales roughly quadratically with n, closely aligning with the theoretical complexity of $O(n^2)$ shown in Proposition 4. This numerical evidence further validates our theoretical findings and demonstrates the efficiency of the proposed DP approach.

6. Conclusions and Future Direction

This paper presented a general framework for isotonic optimization problems incorporating fixed costs. By generalizing conventional isotonic optimization to include general convex cost functions

and enforcing isotonic constraints, we addressed a broader class of practical optimization problems. For path structures, we developed an efficient dynamic programming (DP) algorithm and demonstrated additional computational improvements by leveraging inherent structural properties. For arborescence graph structures, we illustrated that despite increased complexity, the problem remains solvable in polynomial time under specific conditions. These conditions include scenarios with piecewise linear convex cost functions, cases with dominant fixed costs in parent nodes, and particular graph structures such as balanced or constant-round-branching arborescences. Our numerical experiments illustrated that the proposed algorithms significantly outperform the commercial solvers, such as Gurobi, in both accuracy and computational efficiency, thereby empirically substantiating our theoretical results.

Potential avenues for future research include exploring whether isotonic optimization with fixed costs on general arborescences is polynomially solvable. Extending the framework to general graph structures would allow modeling a wider range of applications. Developing convex reformulations for cases that have been shown to be polynomially solvable can be useful for integrating them into larger optimization models, thereby yielding tighter formulations and further enhancing computational performance.

References

Aggarwal A, Klawe M, Moran S, Shor P, Wilber R (1986) Geometric applications of a matrix searching algorithm. *Proceedings of the second annual symposium on Computational geometry*, 285–292.

Aggarwal A, Park JK (1993) Improved algorithms for economic lot size problems. *Operations research* 41(3):549–571.

Ahuja RK, Hochbaum DS, Orlin JB (2003) Solving the convex cost integer dual network flow problem. *Management Science* 49(7):950–964.

Ahuja RK, Orlin JB (2001) A fast scaling algorithm for minimizing separable convex functions subject to chain constraints. *Operations Research* 49(5):784–789.

Bacchetti P (1989) Additive isotonic models. *Journal of the American Statistical Association* 84(405):289–294.

Barlow R, Ubhaya V (1971) Isotonic approximation. *Optimizing Methods in Statistics*, 77–86 (Elsevier).

Barlow RE, Brunk HD (1972) The isotonic regression problem and its dual. *Journal of the American Statistical Association* 67(337):140–147.

Best MJ, Chakravarti N (1990) Active set algorithms for isotonic regression; a unifying framework. *Mathematical Programming* 47(1):425–439.

Best MJ, Chakravarti N, Ubhaya VA (2000) Minimizing separable convex functions subject to simple chain constraints. *SIAM Journal on Optimization* 10(3):658–672.

Chakravarti N (1989) Isotonic median regression: a linear programming approach. *Mathematics of operations research* 14(2):303–308.

Chen X, Hu P (2012) Joint pricing and inventory management with deterministic demand and costly price adjustment. *Operations Research Letters* 40(5):385–389.

Chen X, Li X, Su Y (2023) An active-set based recursive approach for solving convex isotonic regression with generalized order restrictions. *arXiv preprint arXiv:2304.00244* .

Chen X, Simchi-Levi D (2004a) Coordinating inventory control and pricing strategies with random demand and fixed ordering cost: The finite horizon case. *Operations research* 52(6):887–896.

Chen X, Simchi-Levi D (2004b) Coordinating inventory control and pricing strategies with random demand and fixed ordering cost: The infinite horizon case. *Mathematics of operations Research* 29(3):698–723.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2022) *Introduction to algorithms* (MIT press).

Crowston WB, Wagner M, Williams JF (1973) Economic lot size determination in multi-stage assembly systems. *Management Science* 19(5):517–527.

Dykstra RL (1981) An isotonic regression algorithm. *Journal of Statistical Planning and Inference* 5(4):355–363.

Galil Z, Park K (1989) A linear-time algorithm for concave one-dimensional dynamic programming .

Guan Y (2011) Stochastic lot-sizing with backlogging: computational complexity analysis. *Journal of Global Optimization* 49(4):651–678.

Guan Y, Miller AJ (2008) Polynomial-time algorithms for stochastic uncapacitated lot-sizing problems. *Operations Research* 56(5):1172–1183.

Haiminen N, Gionis A, Laasonen K (2008) Algorithms for unimodal segmentation with applications to unimodality detection. *Knowledge and information systems* 14:39–57.

Hardwick J, Stout QF (2014) Optimal reduced isotonic regression. *arXiv preprint arXiv:1412.2844* .

Hochbaum DS (2001) An efficient algorithm for image segmentation, markov random fields and related problems. *Journal of the ACM (JACM)* 48(4):686–701.

Hochbaum DS (2004) 50th anniversary article: Selection, provisioning, shared fixed costs, maximum closure, and implications on algorithmic methods today. *Management Science* 50(6):709–723.

Kaufman Y, Tamir A (1993) Locating service centers with precedence constraints. *Discrete Applied Mathematics* 47(3):251–261.

Love SF (1972) A facilities in series inventory model with nested schedules. *Management Science* 18(5-part-1):327–338.

Maxwell WL, Muckstadt JA (1985) Establishing consistent and realistic reorder intervals in production-distribution systems. *Operations Research* 33(6):1316–1341.

Niculescu-Mizil A, Caruana R (2005) Predicting good probabilities with supervised learning. *Proceedings of the 22nd international conference on Machine learning*, 625–632.

Pardalos PM, Xue G (1999) Algorithms for a class of isotonic regression problems. *Algorithmica* 23:211–222.

Robertson T, Wright F (1973) Multiple isotonic median regression. *The Annals of Statistics* 1(3):422–432.

Robertson T, Wright F (1975) Consistency in generalized isotonic regression. *The Annals of Statistics* 350–362.

Salanti G, Ulm K (2003) A nonparametric changepoint model for stratifying continuous variables under order restrictions and binary outcome. *Statistical methods in medical research* 12(4):351–367.

Scarf H, Arrow K, Karlin S, Suppes P (1960) The optimality of (s, s) policies in the dynamic inventory problem. *Optimal pricing, inflation, and the cost of price adjustment* 49–56.

Schell MJ, Singh B (1997) The reduced monotonic regression method. *Journal of the American Statistical Association* 92(437):128–135.

Schwarz LB, Schrage L (1975) Optimal and system myopic policies for multi-echelon production/inventory assembly systems. *Management Science* 21(11):1285–1294.

Simchi-Levi D, Chen X, Bramel J (2014) The logic of logistics: Theory, algorithms, and applications for logistics management. *Springer Series in Operations Research and Financial Engineering*, 1–447 (Springer).

Stout QF (2008) Unimodal regression via prefix isotonic regression. *Computational Statistics & Data Analysis* 53(2):289–297.

Stout QF (2013) Isotonic regression via partitioning. *Algorithmica* 66:93–112.

Stout QF (2014) An algorithm for L_∞ approximation by step functions. *arXiv preprint arXiv:1412.2379* .

Stout QF (2015) L_∞ isotonic regression for linear, multidimensional, and tree orders. *arXiv preprint arXiv:1507.02226* .

Stout QF (2018) Weighted L_∞ isotonic regression. *Journal of Computer and System Sciences* 91:69–81.

Stout QF (2019) Fastest known isotonic regression algorithms .

Stout QF (2021a) L_0 isotonic regression with secondary objectives. *arXiv preprint arXiv:2106.00279* .

Stout QF (2021b) L_p isotonic regression algorithms using an L_0 approach. *arXiv preprint arXiv:2107.00251* .

Stout QF (2023) Best L_p isotonic regressions, $p \in \{0, 1, \infty\}$. *arXiv preprint arXiv:2306.00269* .

Tamir A (1993) The least element property of center location on tree networks with applications to distance and precedence constrained problems. *Mathematical programming* 62:475–496.

Tempelmeier H (2013) Stochastic lot sizing problems. *Handbook of stochastic models and analysis of manufacturing system operations*, 313–344 (Springer).

Tibshirani RJ, Hoefling H, Tibshirani R (2011) Nearly-isotonic regression. *Technometrics* 53(1):54–61.

Tu CY, Song D, Breidt FJ, Berger TW, Wang H (2012) Functional model selection for sparse binary time series with multiple inputs. *Economic time series: Modeling and seasonality* 477–497.

Vargas V (2009) An optimal solution for the stochastic version of the wagner–whitin dynamic lot-size model. *European Journal of Operational Research* 198(2):447–451.

Wang H, Kai B (2015) Functional sparsity: Global versus local. *Statistica Sinica* 1337–1354.

Wang X, Ying J, Cardoso JVdM, Palomar DP (2022) Efficient algorithms for general isotone optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 8575–8583.

Yu Z, Chen X, Li X (2023) A dynamic programming approach for generalized nearly isotonic optimization. *Mathematical Programming Computation* 15(1):195–225.

Online Appendix: Isotonic Optimization with Fixed Costs

Appendix A: Proofs of Technical Results in Section 2

A.1. Proof of Theorem 1

THEOREM 1. *Suppose* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *is an optimal solution for the Path problem such that* $y_i^* = 1$ *if* $i \in \{i_0, i_1, i_2, \ldots, i_R\} \subseteq [n]$ *where* $1 = i_0 < i_1 < i_2 < \ldots < i_R < i_{R+1} = n+1$ *and* $y_i^* = 0$. *Then*

(a) $x_i^* = \overline{x}^*_{i_{r-1}, i_r - 1}$ *for all* $i \in B(i_{r-1}, i_r - 1)$; *and*

(b) The strict inequality $\overline{x}^*_{i_{r-1}, i_r - 1} < \overline{x}^*_{i_r, i_{r+1}-1}$ *must hold for any* $r \in [R]$.

Proof. Indexes $i_1, \ldots, i_R$ partition $[n]$ into blocks $B(1, i_1 - 1), B(i_1, i_2 - 1) \ldots, B(i_R, n)$ where $B(i, j) = \{i, \ldots, j\}$. Since x_i's in each block share the same value, we let $x_i^* = \widetilde{x}_{i_{r-1}, i_r - 1}$ for any $i \in B(i_{r-1}, i_r - 1)$. Hence, the cost corresponding to block $B(i_{r-1}, i_r - 1)$ equals $K_{i_{r-1}} + \sum_{\ell \in B(i_{r-1}, i_r - 1)} f_\ell(\widetilde{x}_{i_{r-1}, i_r - 1})$. If we can prove that $\widetilde{x}_{i_{r-1}, i_r - 1} = \overline{x}^*_{i_{r-1}, i_r - 1}$, the proof is done. Assume by contradiction that for some $r \in [R+1]$, $\widetilde{x}_{i_{r-1}, i_r - 1} \neq \overline{x}^*_{i_{r-1}, i_r - 1}$. Two cases need to be discussed.

Case 1: $\boldsymbol{\widetilde{x}_{i_{r-1}, i_r - 1} > \overline{x}^*_{i_{r-1}, i_r - 1}}$. We decrease $\widetilde{x}_{i_{r-1}, i_r - 1}$ until one of the following two different cases happens: if $r = 1$ or $r > 1$ and $\widetilde{x}_{i_{r-2}, i_{r-1} - 1} \leq \overline{x}^*_{i_{r-1}, i_r - 1}$, then we decrease $\widetilde{x}_{i_{r-1}, i_r - 1}$ to be $\overline{x}^*_{i_{r-1}, i_r - 1}$ without violating the isotonic constraint between block $B(i_{r-2}, i_{r-1} - 1)$ and $B(i_{r-1}, i_r - 1)$. Since $\overline{x}^*_{i_{r-1}, i_r - 1}$ minimizes $\sum_{\ell \in B(i_{r-1}, i_r - 1)} f_\ell(x)$, we know that the cost corresponding to block $B(i_{r-1}, i_r - 1)$ decreases or at least holds the same, contradicting the optimality of $\boldsymbol{x}^*$; if $r > 1$ and $\widetilde{x}_{i_{r-2}, i_{r-1} - 1} > \overline{x}^*_{i_{r-1}, i_r - 1}$, then we decrease $\widetilde{x}_{i_{r-1}, i_r - 1}$ to be $\widetilde{x}_{i_{r-2}, i_{r-1} - 1}$. Therefore, block $B(i_{r-2}, i_{r-1} - 1)$ and $B(i_{r-1}, i_r - 1)$ are merged together to be a single block $B(i_{r-2}, i_r - 1)$. From the convexity of $\sum_{\ell = i_{r-1}}^{i_r - 1} f_\ell(x)$, we know that it is nondecreasing in $[\overline{x}^*_{i_{r-1}, i_r - 1}, \infty)$, and by moving its evaluated point from $\widetilde{x}_{i_{r-1}, i_r - 1}$ towards its minimizer $\overline{x}^*_{i_{r-1}, i_r - 1}$ (and make it to be $\widetilde{x}_{i_{r-2}, i_{r-1} - 1}$), its value decreases or at least holds the same. Besides, by merging block $B(i_{r-2}, i_{r-1} - 1)$ and $B(i_{r-1}, i_r - 1)$, the fixed cost $K_{i_{r-1}}$ is saved. Hence, the overall cost decreases or holds the same, contradicting the optimality of $(\boldsymbol{x}^*, \boldsymbol{y}^*)$.

Case 2: $\boldsymbol{\widetilde{x}_{i_{r-1}, i_r - 1} < \overline{x}^*_{i_{r-1}, i_r - 1}}$. We increase $\widetilde{x}_{i_{r-1}, i_r - 1}$ until one of the following two different cases happens: if $r = R+1$ or $r > R+1$ and $\widetilde{x}_{i_r, i_{r+1} - 1} \geq \overline{x}^*_{i_{r-1}, i_r - 1}$, then we increase $\widetilde{x}_{i_{r-1}, i_r - 1}$ to be $\overline{x}^*_{i_r, i_{r+1} - 1}$ without violating the isotonic constraint between block $B(i_{r-1}, i_r - 1)$ and $B(i_r, i_{r+1} - 1)$. Since $\overline{x}^*_{i_{r-1}, i_r - 1}$ minimizes $\sum_{\ell = i_{r-1}}^{i_r - 1} f_\ell(x)$, we know that the cost corresponding to block $B(i_{r-1}, i_r - 1)$ decreases or at least holds the same, contradicting the optimality of $\boldsymbol{x}^*$; if $r < R+1$ and $\widetilde{x}_{i_r, i_{r+1} - 1} < \overline{x}^*_{i_{r-1}, i_r - 1}$, then we increase $\widetilde{x}_{i_{r-1}, i_r - 1}$ to be $\widetilde{x}_{i_r, i_{r+1} - 1}$. Therefore, block $B(i_{r-1}, i_r - 1)$ and $B(\widetilde{x}_{i_r, i_{r+1} - 1})$ are merged together to be a single block $B(i_{r-1}, i_{r+1} - 1)$. From the convexity of $\sum_{\ell = i_{r-1}}^{i_r - 1} f_\ell(x)$, we know that it is nonincreasing in $(-\infty, \overline{x}^*_{i_{r-1}, i_r - 1}]$, and by moving its evaluated point from $\widetilde{x}_{i_{r-1}, i_r - 1}$

towards its minimizer $\overline{x}^*_{i_{r-1},i_r-1}$ (and make it to be $\widetilde{x}_{i_r,i_{r+1}-1}$), its value decreases or at least holds the same. Besides, by merging block $B(i_{r-1},i_r-1)$ and $B(\widetilde{x}_{i_r,i_{r+1}-1})$, the fixed cost K_{i_r} is saved. Hence, the overall cost decreases or holds the same, contradicting the optimality of $(\boldsymbol{x}^*,\boldsymbol{y}^*)$. □

A.2. Proof of Proposition 1

PROPOSITION 1. *Suppose that $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s are precomputed. Then, the DP recursion* (DP-original) *can be efficiently implemented using Algorithm 1 in $O(n^2\log n)$ time.*

Proof. We first show the correctness of the algorithm. Line 2 to line 4 computes $V(j,k)$ for the case when $1=j<k\le n+1$, according to the recursion (DP-original). In the for loop from line 5 to line 16, for each $j=2,\ldots,n$, we first sort those $\overline{x}^*_{1,j-1},\ldots,\overline{x}^*_{j-1,j-1},\overline{x}^*_{j,j},\ldots,\overline{x}^*_{j,n}$ and then compute the value-to-go functions $V(j,j+1),\ldots,V(j,n+1)$ progressively according to the sorted list, such that we do not need to go over the list over and over again to determine each of $V(j,j+1),\ldots,V(j,n+1)$ and some computation is saved. The key is that we use $V_{\min}(\ell)$ to record the minimum $V(i,j)$ from the beginning up to the current position in the sorted list, which is kept updated in the loop. Hence, at the ℓ-th position of the sorted list, if $\pi(\ell)<j$, then $V_{\min}(\ell)$ is updated to be $\min\{V(\pi(\ell),j),V_{\min}(\ell-1)\}$; else if $\pi(\ell)\ge j$, from (DP-original), we update $V_{\min}(\ell)=V_{\min}(\ell-1)$ and $V(j,\pi(\ell)+1)=c_{j,\pi(\ell)}+V_{\min}(\ell)$. This computes $V(j,k)$ for the case when $2\le j<k\le n+1$. Hence, we prove the correctness of Algorithm 1.

Next, we derive the time complexity of Algorithm 1. The for loop from line 2 to line 4 requires $O(n)$ time; the for loop from line 5 to line 16 has $O(n)$ iterations, and in each iteration, line 6 can be implemented in $O(n)$ time, the sorting in line 7 takes $O(n\log n)$ time, and the for loop from line 9 to line 15 takes $O(n)$ time. Hence, the overall dynamic programming can be implemented in $O(n^2\log n)$ time. □

A.3. Proof of Theorem 2

We begin with a technical lemma.

LEMMA EC.1. *Suppose that three convex functions $g_1(y),g_2(y),g_3(y)$ satisfy $-\infty<\overline{y}^*_1\le\overline{y}^*_2\le\overline{y}^*_3<\infty$, where $\overline{y}^*_i=\min\{y^*|y^*\in\arg\min g_i(y)\}$ for $i\in[3]$. Then the following inequality must hold:*

$$\min_y\Big\{g_1(y)+g_2(y)+g_3(y)\Big\}+\min_y\Big\{g_2(y)\Big\}\ge\min_y\Big\{g_1(y)+g_2(y)\Big\}+\min_y\Big\{g_2(y)+g_3(y)\Big\}.$$

Proof. Other than $\overline{y}_1^*, \overline{y}_2^*, \overline{y}_3^*$ defined in the statement of the lemma, let us define some additional minima:

$$\overline{y}_{12}^* = \min\left\{ y^* \middle| y^* \in \arg\min\left\{ g_1(y) + g_2(y) \right\} \right\},$$

$$\overline{y}_{23}^* = \min\left\{ y^* \middle| y^* \in \arg\min\left\{ g_2(y) + g_3(y) \right\} \right\},$$

$$\overline{y}_{123}^* = \min\left\{ y^* \middle| y^* \in \arg\min\left\{ g_1(y) + g_2(y) + g_3(y) \right\} \right\}.$$

It can be easily seen that $\overline{y}_1^* \le \overline{y}_{12}^* \le \overline{y}_2^* \le \overline{y}_{23}^* \le \overline{y}_3^*$ and $\overline{y}_{12}^* \le \overline{y}_{123}^* \le \overline{y}_{23}^*$ due to the convexity of $g_1(y), g_2(y), g_3(y)$ and the assumption $-\infty < \overline{y}_1^* \le \overline{y}_2^* \le \overline{y}_3^* < \infty$. Two cases are remaining to be discussed: $\overline{y}_{123}^* \le \overline{y}_2^*$ and $\overline{y}_{123}^* > \overline{y}_2^*$.

Case 1: $\overline{y}_{123}^* \le \overline{y}_2^*$. In this case, we know that

$$\begin{aligned}
&\min_y \left\{ g_1(y) + g_2(y) + g_3(y) \right\} + \min_y \left\{ g_2(y) \right\} - \min_y \left\{ g_1(y) + g_2(y) \right\} - \min_y \left\{ g_2(y) + g_3(y) \right\} \\
=& g_1(\overline{y}_{123}^*) + g_2(\overline{y}_{123}^*) + g_3(\overline{y}_{123}^*) + g_2(\overline{y}_2^*) - g_1(\overline{y}_{12}^*) - g_2(\overline{y}_{12}^*) - g_2(\overline{y}_{23}^*) - g_3(\overline{y}_{23}^*) \\
\ge& g_1(\overline{y}_{123}^*) + g_2(\overline{y}_{123}^*) + g_3(\overline{y}_{123}^*) + g_2(\overline{y}_2^*) - g_1(\overline{y}_{123}^*) - g_2(\overline{y}_{123}^*) - g_2(\overline{y}_2^*) - g_3(\overline{y}_2^*) \\
=& g_3(\overline{y}_{123}^*) - g_3(\overline{y}_2^*) \\
\ge& 0,
\end{aligned}$$

where the first inequality holds because $g_1(\overline{y}_{12}^*) + g_2(\overline{y}_{12}^*) = \min_y\{g_1(y) + g_2(y)\} \le g_1(\overline{y}_{123}^*) + g_2(\overline{y}_{123}^*)$, and $g_2(\overline{y}_{23}^*) + g_3(\overline{y}_{23}^*) = \min_y\{g_2(y) + g_3(y)\} \le g_2(\overline{y}_2^*) + g_3(\overline{y}_2^*)$, and the second inequality follows from the convexity of $g_3(y)$ and $\overline{y}_{123}^* \le \overline{y}_2^* \le \overline{y}_3^*$. Therefore, the inequality in the lemma holds.

Case 2: $\overline{y}_{123}^* > \overline{y}_2^*$. In this case, we have that

$$\begin{aligned}
&\min_y \left\{ g_1(y) + g_2(y) + g_3(y) \right\} + \min_y \left\{ g_2(y) \right\} - \min_y \left\{ g_1(y) + g_2(y) \right\} - \min_y \left\{ g_2(y) + g_3(y) \right\} \\
=& g_1(\overline{y}_{123}^*) + g_2(\overline{y}_{123}^*) + g_3(\overline{y}_{123}^*) + g_2(\overline{y}_2^*) - g_1(\overline{y}_{12}^*) - g_2(\overline{y}_{12}^*) - g_2(\overline{y}_{23}^*) - g_3(\overline{y}_{23}^*) \\
=& \underbrace{\left[g_1(\overline{y}_2^*) + g_2(\overline{y}_2^*) - g_1(\overline{y}_{12}^*) - g_2(\overline{y}_{12}^*) \right]}_{\ge 0} + \underbrace{\left[g_2(\overline{y}_{123}^*) + g_3(\overline{y}_{123}^*) - g_2(\overline{y}_{23}^*) - g_3(\overline{y}_{23}^*) \right]}_{\ge 0} + \underbrace{\left[g_1(\overline{y}_{123}^*) - g_1(\overline{y}_2^*) \right]}_{\ge 0} \\
\ge& 0,
\end{aligned}$$

where the second equality holds by adding $g_1(\overline{y}_2^*)$ and subtracting $g_1(\overline{y}_2^*)$ simultaneously from the last step, and after grouping, we know that the expression in the first curly bracket is nonnegative because $g_1(\overline{y}_{12}^*) + g_2(\overline{y}_{12}^*) = \min_y\{g_1(y) + g_2(y)\} \le g_1(\overline{y}_2^*) + g_2(\overline{y}_2^*)$, the expression in the second curly bracket is nonnegative because $g_2(\overline{y}_{23}^*) + g_3(\overline{y}_{23}^*) = \min_y\{g_2(y) + g_3(y)\} \le g_2(\overline{y}_{123}^*) + g_3(\overline{y}_{123}^*)$, and the expression in the third curly bracket is nonnegative because of the convexity of $g_1(y)$ and the fact that $\overline{y}_{123}^* > \overline{y}_2^* \ge \overline{y}_1^*$. Therefore, the inequality in the lemma holds. □

Based on Lemma EC.1, we can prove Theorem 2.

THEOREM 2. *Under Assumption 1, matrix* $\widetilde{C} = (\widetilde{c}_{i,j})_{i\in[n],j\in[n]}$ *defined by* (4) *satisfies the Monge property, i.e.,*

$$\widetilde{c}_{i,j} + \widetilde{c}_{i+1,j+1} \le \widetilde{c}_{i+1,j} + \widetilde{c}_{i,j+1}$$

for each $i \in [n-1], j \in [n-1]$.

Proof. If $i \ge j$, then $\widetilde{c}_{i+1,j} = \infty$ and the inequality of the Monge condition is automatically satisfied as its RHS is ∞. Hence, we only consider the case when $i \le j-1$. In this case,

$$\begin{aligned}
&\widetilde{c}_{i,j} + \widetilde{c}_{i+1,j+1} \le \widetilde{c}_{i+1,j} + \widetilde{c}_{i,j+1} \\
\iff\quad & K_i + \min_x \left\{\sum_{\ell=i}^{j} f_\ell(x)\right\} + K_{i+1} + \min_x \left\{\sum_{\ell=i+1}^{j+1} f_\ell(x)\right\} \le K_{i+1} + \min_x \left\{\sum_{\ell=i+1}^{j} f_l(x)\right\} + K_i + \min_x \left\{\sum_{\ell=i}^{j+1} f_\ell(x)\right\} \\
\iff\quad & \min_x \left\{\sum_{\ell=i}^{j} f_\ell(x)\right\} + \min_x \left\{\sum_{\ell=i+1}^{j+1} f_\ell(x)\right\} \le \min_x \left\{\sum_{\ell=i+1}^{j} f_\ell(x)\right\} + \min_x \left\{\sum_{\ell=i}^{j+1} f_\ell(x)\right\} \\
\iff\quad & \min_x \left\{f_i(x) + \sum_{\ell=i+1}^{j} f_\ell(x)\right\} + \min_x \left\{\sum_{\ell=i+1}^{j} f_\ell(x) + f_j(x)\right\} \\
& \le \min_x \left\{\sum_{\ell=i+1}^{j} f_\ell(x)\right\} + \min_x \left\{f_i(x) + \sum_{\ell=i+1}^{j} f_\ell(x) + f_j(x)\right\}
\end{aligned}$$

By viewing $g_1(y) = f_i(y), g_2(y) = \sum_{\ell=i+1}^{j} f_\ell(y), g_3(y) = f_{j+1}(y)$ and applying the result in Lemma EC.1 (the conditions for $g_1(y), g_2(y), g_3(y)$ in Lemma EC.1 are satisfied), we know that the inequality in the last line holds. This concludes the proof. □

A.4. Proof of Lemma 1

LEMMA 1. *To compute* $\overline{x}^*_{i,j}$*'s and* $c_{i,j}$*'s defined in* (3) *for all pairs* (i,j) *such that* $1 \le i \le j \le n$*, we have*

(i) If $f_i(x) = p_i x^2 - 2q_i x + r_i$ *with* $p_i > 0$ *for any* $i \in [n]$*, then the computation can be done in* $O(n^2)$ *time; and*

(ii) If $f_i(x) = p_i(r_i - x)^+ + q_i(x - r_i)^+$ *with* $p_i, q_i > 0$ *for any* $i \in [n]$*, then the computation can be done in* $O(n^2 \log n)$ *time.*

Proof. (i) Since $\overline{x}^*_{i,j} = \arg\min \sum_{\ell=i}^{j}(p_l x^2 - 2q_\ell x + r_\ell)$, it can be easily seen that $\overline{x}^*_{i,j} = \frac{\sum_{\ell=i}^{j} q_\ell}{\sum_{\ell=i}^{j} p_\ell}$. Hence, $c_{i,j} = K_i + \sum_{\ell=i}^{j}(p_i(\frac{\sum_{\ell=i}^{j} q_\ell}{\sum_{\ell=i}^{j} p_\ell})^2 - 2q_\ell(\frac{\sum_{\ell=i}^{j} q_\ell}{\sum_{\ell=i}^{j} p_\ell}) + \sum_{\ell=i}^{j} r_\ell) = K_i - \frac{(\sum_{\ell=i}^{j} q_\ell)^2}{\sum_{\ell=i}^{j} p_\ell} + \sum_{\ell=i}^{j} r_\ell$. Let us pre-calculate $\sum_{\ell=1}^{i} p_\ell$, $\sum_{\ell=1}^{i} q_\ell$ and $\sum_{\ell=1}^{i} r_\ell$ for all $i \in [n]$ using $O(n)$ time and store them. Hence, each $\overline{x}^*_{i,j} = \frac{\sum_{\ell=1}^{j} q_\ell - \sum_{\ell=1}^{i-1} q_\ell}{\sum_{\ell=1}^{j} p_\ell - \sum_{\ell=1}^{i-1} p_\ell}$ and $c_{i,j} = K_i + \frac{(\sum_{\ell=1}^{j} q_\ell - \sum_{\ell=1}^{i-1} q_\ell)^2}{\sum_{\ell=1}^{j} p_\ell - \sum_{\ell=1}^{i-1} p_\ell} + \sum_{\ell=1}^{j} r_\ell - \sum_{\ell=1}^{i-1} r_\ell$ can be calculated from those stored prefix sums in $O(1)$ time. Then calculating all $\overline{x}^*_{i,j}$'s and $c_{i,j}$'s requires $O(n^2)$ time.

(ii) Since $\sum_{\ell=i}^{j-1}(p_\ell(r_\ell - x)^+ + q_\ell(x - r_\ell)^+)$ is a piece-wise linear function with kinking points $r_i, \ldots, r_j$, then its minimum minimizer $\overline{x}^*_{i,j}$ should also be one of those kinking points. Let ℓ^*_{ij} be the minimum

index in $\{i,\dots,j\}$ such that $\overline{x}^*_{i,j}=r_{\ell^*_{ij}}$. We require that the left derivative at $r_{\ell^*_{ij}}$ is non-positive and the right derivative is non-negative, i.e., the standard first-order conditions for minimizing a convex function. This ensures that ℓ^*_{ij} is the first index at which the cumulative marginal cost shifts from decreasing to increasing. Hence, we know that ℓ^*_{ij} should satisfy

$$\begin{aligned}&\sum_{\ell=i:r_\ell>r_{\ell^*_{ij}}}^{j} p_\ell \le \sum_{\ell=i:r_\ell\le r_{\ell^*_{ij}}}^{j} q_\ell \quad\text{and}\quad \sum_{\ell=i:r_\ell\ge r_{\ell^*_{ij}}}^{j} p_\ell > \sum_{\ell=i:r_\ell<r_{\ell^*_{ij}}}^{j} q_\ell,\\ \iff &\sum_{\ell=i:r_\ell>r_{\ell^*_{ij}}}^{j} (p_\ell+q_\ell) \le \sum_{\ell=i}^{j} q_\ell \quad\text{and}\quad \sum_{\ell=i:r_\ell\ge r_{\ell^*_{ij}}}^{j} (p_\ell+q_\ell) > \sum_{\ell=i}^{j} q_\ell.\end{aligned} \tag{EC.1}$$

Let us compute $\sum_{\ell\in[i]} q_\ell$ for all $i\in[n]$ in $O(n)$ time and store them, such that we can obtain $\sum_{\ell=i}^{j} q_\ell = \sum_{\ell\in[j]} q_\ell - \sum_{\ell\in[i-1]} q_\ell$ in $O(1)$ time from those stored prefix sums. After that, let us maintain an appropriate data structure such that we can calculate $\overline{x}^*_{i,i},\overline{x}^*_{i,i+1},\dots,\overline{x}^*_{i,n}$ in a single round efficiently. This requires us to maintain a special data structure, such as an augmented balanced binary search tree (e.g., an augmented red-black tree), which is used to compute dynamic order statistics efficiently, as mentioned by Stout (2008). Each node represents an $i\in[n]$ with its key r_i, and the augmented position is the sum of $p_\ell+q_\ell$ over all ℓ's in the subtree rooted at the current node i. To compute $\overline{x}^*_{i,i},\overline{x}^*_{i,i+1},\dots,\overline{x}^*_{i,n}$ in round $i\in[n]$, we insert node i till node n one by one, and maintain an augmented balanced binary search tree along the way. Inserting a new node into the tree, rebalancing it, and updating the augmented positions takes $O(\log n)$ time (Stout 2008, Cormen et al. 2022). After doing so for each node j where $j\in\{i,\dots,n\}$, we find ℓ^*_{ij} satisfying the condition (EC.1) in the following way:

(a) Initializing $S=0$ and starting from the root node.

(b) For the current node k, we check the augmented position of its right child (i.e., the sum of $p_\ell+q_\ell$ in its right subtree), and compare its value R plus S with $\sum_{\ell=i}^{j} q_\ell$; if $S+R>\sum_{\ell=i}^{j} q_\ell$, then we move to the right child; else if $S+R\le\sum_{\ell=i}^{j} q_\ell$ and $S+R+(p_k+q_k)>\sum_{\ell=i}^{j} q_\ell$, we output $\ell^*_{ij}=k$ and stop; else if $S+R+(p_k+q_k)\le\sum_{\ell=i}^{j} q_\ell$, we update $S=S+R+(p_k+q_k)$ and move to the left child.

(c) Recursively doing step (b) until we output ℓ^*_{ij}.

The validity follows directly from the logic of the above procedure. It requires $O(\log n)$ steps, each implemented using $O(1)$ time. Hence, calculating all $\overline{x}^*_{i,i},\overline{x}^*_{i,i+1},\dots,\overline{x}^*_{i,n}$ in the same loop requires $O(n\log n)$ time, and computing all $\overline{x}^*_{i,j}$'s requires us to implement the above loop for all $i\in[n]$, which takes $O(n^2\log n)$ time.

To compute $c_{i,j}$'s based on those $\overline{x}^*_{i,j}$'s, note that each $\overline{x}^*_{i,j}$ is just $r_{\ell^*_{ij}}$ for some $\ell^*_{ij}\in\{i,\dots,j\}$. Hence, we calculate $\sum_{\ell\in[i]}(p_\ell(r_\ell-r_k)^+ + q_\ell(r_k-r_\ell)^+)$ for each $i\in[n]$ and $k\in[n]$ in $O(n^2)$ time and store them, and compute $c_{i,j}=K_i+\sum_{\ell=i}^{j}(p_\ell(r_\ell-r_{\ell^*_{ij}})^+ + q_\ell(r_{\ell^*_{ij}}-r_\ell)^+) = K_i + (\sum_{\ell\in[j]}(p_\ell(r_\ell-$

$r_{\ell^*_{ij}})^+ + q_\ell(r_{\ell^*_{ij}} - r_\ell)^+)) - (\sum_{\ell\in[i-1]}(p_\ell(r_\ell - r_{\ell^*_{ij}})^+ + q_\ell(r_{\ell^*_{ij}} - r_\ell)^+))$ from those prefixed sums in $O(1)$ time. Hence, the computation of all $c_{i,j}$'s needs additional $O(n^2)$ time.

Consequently, the overall computation complexity is $O(n^2 \log n)$. □

Appendix B: Proofs of Technical Results in Section 3

B.1. Proof of Proposition 3

PROPOSITION 3. *Suppose* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *is an optimal solution to the Arborescence problem and let* $M \subseteq [n]$ *denote the support of the vector* $\boldsymbol{y}^*$. *Then the optimal solution* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *partitions* $\mathcal{T}$ *into several induced arborescences* $\mathcal{T}^i = (N_{\mathcal{T}^i}, A_{\mathcal{T}^i})$ *for all* $i \in M$, *where each* $\mathcal{T}^i$ *is rooted at node* i *and* $N_{\mathcal{T}^i} = N_{\mathcal{T}(i)} \backslash (\cup_{j\in N_{\mathcal{T}(i)}\cap M, j\neq i} N_{\mathcal{T}(j)})$, *and* $A_{\mathcal{T}^i}$ *are arcs of* $A_{\mathcal{T}}$ *restricted on* $N_{\mathcal{T}^i}$. *Let* $\overline{x}^*_{\mathcal{T}^i} = \min\{x^* | x^* \in \arg\min \sum_{\ell\in N_{\mathcal{T}^i}} f_\ell(x)\}$ *and* $c_{\mathcal{T}^i} = \sum_{\ell\in N_{\mathcal{T}^i}} f_\ell(\overline{x}^*_{\mathcal{T}^i})$. *We have that:*

(a) $x^*_\ell = \overline{x}^*_{\mathcal{T}^i}$ *for all* $\ell \in N_{\mathcal{T}^i}$, *and*

(b) $\overline{x}^*_{\mathcal{T}^i} < \overline{x}^*_{\mathcal{T}^j}$ *for all* $i \in M$ *and* $j \in N_{\mathcal{T}(i)} \cap M$.

Proof. The proof follows a similar idea to the proof for Theorem 1. Hence, we will only prove it by sketch. Assume by contradiction that $x^*_\ell \equiv \widetilde{x}_{\mathcal{T}^i}$ for all $\ell \in N_{\mathcal{T}^i}$ and $\widetilde{x}_{\mathcal{T}^i} \neq \overline{x}^*_{\mathcal{T}^i}$. Two cases need to be discussed. One is that $\widetilde{x}_{\mathcal{T}^i} > \overline{x}^*_{\mathcal{T}^i}$, for which case we can decrease $\widetilde{x}_{\mathcal{T}^i}$ until $\widetilde{x}_{\mathcal{T}^i} = \overline{x}^*_{\mathcal{T}^i}$ or arborescence $\mathcal{T}^i$ is merged with its precedent arborescence $\mathcal{T}^j$ where $i \in N_{\mathcal{T}(j)} \cap M$, contradicting the optimality of $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. The other is that $\widetilde{x}_{\mathcal{T}^i} < \overline{x}^*_{\mathcal{T}^i}$, in which case we can either increase $\widetilde{x}_{\mathcal{T}^i}$ until $\widetilde{x}_{\mathcal{T}^i} = \overline{x}^*_{\mathcal{T}^i}$ or the arborescence $\mathcal{T}^i$ is merged with one of its subsequent arborescences $\mathcal{T}^j$ for all $j \in N_{\mathcal{T}(i)} \cap M$, which contradicts the optimality of $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. This concludes the proof. □

B.2. Proof of Proposition 4

PROPOSITION 4. *Under Assumption 2, the Arborescence problem can be solved in* $O((n + \log(\sum_{i\in[n]} m_i)) \sum_{i\in[n]} m_i)$ *time by efficiently implementing* (DP-PL), *which is of order* $O(n^2)$ *if we assume* $m_i \equiv O(1)$ *for all* $i \in [n]$.

Proof. We first sort all the kink points in $\mathcal{K}$ using $O(\sum_{i\in[n]} m_i \log(\sum_{i\in[n]} m_i))$ time. Then, for each $i \in [n]$, we do backpropagation following the same idea of Algorithm 1 to calculate $V(i,k)$'s for all $k \in \mathcal{K}$ in the same round. With the presorted kink points in $\mathcal{K}$, for each $i \in [n]$, the backpropagation step to compute $V(i,k)$'s for all $k \in \mathcal{K}$ only needs $O(\sum_{i\in[n]} m_i)$ time, and for all $i \in [n]$ it takes $O(n \sum_{i\in[n]} m_i)$. Besides, recall that the evaluation part also takes $O(n \sum_{i\in[n]} m_i)$ time. Putting them together, we conclude that (DP-PL) can be solved in $O((n + \log(\sum_{i\in[n]} m_i)) \sum_{i\in[n]} m_i)$ total time. It is of the order $O(n^2)$ if we assume that $m_i \equiv O(1)$ for all $i \in [n]$. □

B.3. Proof of Theorem 3

Below, we summarize several key properties of K-convex functions from the literature, which are useful to prove Theorem 3.

LEMMA EC.2 **(Lemma 9.3.2 and Proposition 9.3.3 of Simchi-Levi et al. (2014))**. *The K-convex functions have the following properties:*

(a) A real-valued convex function is also 0-convex and hence K convex for all $K \geq 0$. A K_1-convex function is also K_2-convex function for $K_1 \leq K_2$.

(b) If $f_1(x)$ and $f_2(x)$ are K_1-convex and K_2-convex, then for $\alpha, \beta \geq 0$, $\alpha f_1(x) + \beta f_2(x)$ is $(\alpha K_1 + \beta K_2)$-convex.

(c) If $f(x)$ is a K-convex function, then function

$$\phi(x) = \min_{x' \geq x} \left\{ Q\mathbb{1}\{x' > x\} + f(x') \right\}$$

is $\max\{K, Q\}$-convex.

(d) Assume that f is a continuous K-convex function and $f(x) \to \infty$ as $|x| \to \infty$. Let S be a minimum point of f and s be any element of the set

$$\left\{ x | x \leq S, f(x) = f(S) + K \right\}.$$

Then the following results hold:

(i) $f(S) + K = f(s) \leq f(x)$, for all $x \leq s$.

(ii) $f(x)$ is a non-increasing function on $(-\infty, s)$.

(iii) $f(x) \leq f(x') + K$ for all x, x' with $s \leq x \leq x'$.

THEOREM 3. *Under Assumption 3, for each $i \in [n]$, the value-to-go function $V(i, x)$ defined in (5) satisfies the following three properties:*

(a) $V(i, x)$ is K_i-convex;

(b) there exists a pair of real numbers (s_i, S_i) satisfying

$$S_i \in \arg\min U(i, x), \tag{7a}$$

$$s_i \in \left\{ x | x \leq S_i, U(i, x) = U(i, S_i) + K_i \right\}, \tag{7b}$$

such that an optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ to the Arborescence problem is

$$x_i^* = \begin{cases} S_i, & x_{p(i)}^* \leq s_i, \\ x_{p(i)}^*, & o.w. \end{cases}, \quad y_i^* = \begin{cases} 1, & x_{p(i)}^* \leq s_i, \\ 0, & o.w; \end{cases}$$

(c) $V(i,x)$ can be expressed as

$$V(i,x)=\begin{cases}K_i+U(i,S_i), & x\le s_i,\\ U(i,x), & o.w.;\end{cases}$$

(d) $V(i,x)$ is piece-wise convex with at most $|N_{\mathcal{T}(i)}|+|\mathcal{L}(i)|$ number of convex pieces.

Proof. We prove this by induction. First consider all $i\in\mathcal{L}$, and in this case $U(i,x)=f_i(x)$. For each $i\in\mathcal{L}$, $V(i,x)=\min_{x'\ge x}\{K_i\mathbb{1}\{x'>x\}+f_i(x')\}$. Because $f_i(x)$ is convex whence 0-convex by Lemma EC.2(a), from Lemma EC.2(c) we know that $V(i,x)$ is K_i-convex and (a) holds. Besides, since $f_i(x)\to\infty$ as $|x|\to\infty$, with the definition of (s_i,S_i) we know that when $x^*_{p(i)}\le s_i$, we should increase x_i to let $x^*_i=S_i$ and incur a cost $K_i+f_i(S_i)=f_i(s_i)\le f_i(x^*_{p(i)})$, otherwise we holds x_i still and make $x^*_i=x^*_{p(i)}$, incurring a cost $f_i(S_i)$. Hence, we can express $V(i,x)$ as

$$V(i,x)=\begin{cases}K_i+f_i(S_i), & x\le s_i,\\ f_i(x), & \text{o.w.}\end{cases}$$

It verifies (b). It can be easily seen that $V(i,x)$ is a piece-wise convex function with two pieces $(-\infty,s_i]$ and (s_i,∞). Since i is a leaf node, $|N_{\mathcal{T}(i)}|=|\mathcal{L}(i)|=1$, we know that $V(i,x)$ is a piece-wise convex function with $|N_{\mathcal{T}(i)}|+|\mathcal{L}(i)|$ pieces. Hence, (c) is justified, and the statement in the theorem holds for all leaf nodes.

For $i\in[n]\backslash\mathcal{L}$, suppose the statement in the theorem holds for $V(j,x)$ for all $j\in\mathcal{C}(i)$. Again, recall that $V(i,x)=\min_{x'\ge x}\{K_i\mathbb{1}\{x'>x\}+U(i,x')\}$. Since each $V(j,x)$ is K_j-convex for each $j\in\mathcal{C}(i)$ and $f_i(x)$ is convex whence 0-convex, by Lemma EC.2(b) we know that $U(i,x)=f_i(x)+\sum_{j\in\mathcal{C}(i)}V(j,x)$ is $\sum_{j\in\mathcal{C}(i)}K_j$-convex. Since $K_i\ge\sum_{j\in\mathcal{C}(i)}K_j$ according to our assumption, by Lemma EC.2(c) we know that $V(i,x)$ is K_i-convex, which means (a) holds. Because $U(i,x)$ is $\sum_{j\in\mathcal{C}(i)}K_j$-convex whence K_i-convex by our assumption $K_i\ge\sum_{j\in\mathcal{C}(i)}K_j$ and Lemma EC.2(a), and goes to ∞ as $|x|\to\infty$, from the definition of (s_i,S_i) and Lemma EC.2(d) we know that

(i) $U(i,S_i)+K_i=U(i,s_i)\le U(i,x)$, for all $x\le s_i$.

(ii) $U(i,x)$ is a non-increasing function on $(-\infty,s_i)$.

(iii) $U(i,x)\le U(i,x')+K_i$ for all x,x' with $s_i\le x\le x'$.

Hence, we will make an increase and let $x^*_i=S_i$ if $x^*_{p(i)}\le s_i$, and hold it still to be $x^*_{p(i)}$ otherwise. Therefore, $V(i,x)$ can be expressed as

$$V(i,x)=\begin{cases}K_i+U(i,S_i), & x\le s_i,\\ U(i,x), & \text{o.w.}\end{cases}$$

It verifies (b) and (c). Besides, for each $j\in\mathcal{C}(i)$, $V(j,x)$ is a piece-wise convex function with $|N_{\mathcal{T}(j)}|+|\mathcal{L}(j)|$ pieces by induction hypothesis, which means that $\sum_{j\in\mathcal{C}(i)}V(j,x)$ is a piece-wise convex function with at most $\sum_{j\in\mathcal{C}(i)}|N_{\mathcal{T}(j)}|+\sum_{j\in\mathcal{C}(i)}|\mathcal{L}(j)|=|N_{\mathcal{T}(i)}\backslash\{i\}|+|\mathcal{L}(i)|=|N_{\mathcal{T}(i)}|+|\mathcal{L}(i)|-1$ number of

pieces, the same for $U(i,x) = f_i(x) + \sum_{j\in\mathcal{C}(i)} V(j,x)$ by simply adding another convex function $f_i(x)$. From the expression of $V(i,x)$ we know that it introduces at most one convex piece $(-\infty, s_i]$ compared with $U(i,x)$. Hence, we conclude that $V(i,x)$ is a piece-wise convex function with at most $|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|$ pieces and the proof of part (d) is completed. □

B.4. Proof of Proposition 5

PROPOSITION 5. *Suppose $K_i \geq \sum_{j\in\mathcal{C}(i)} K_j$ for any $i \in [n]$. Assume S_i defined by (7a) and s_i defined by (7b) are located on a one-dimensional ϵ-grid of the domain (i.e., a discretized grid over the domain with resolution $\epsilon > 0$) for any $i \in [n]$, then*

(a) There exists a list representation (8) for each $V(i,x)$, whose functional form can be recovered from the list representation according to (9).

(b) The optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ for the Arborescence problem can be obtained in $O(n^3 \log(U/\epsilon))$ time.

Proof. We provide a step-by-step algorithm to show the validity of the list representation, and the algorithm's computational complexity will be derived.

Step A. Computation for Leaf Nodes: We first show that the list representation (9) is valid for each $i \in \mathcal{L}$ and show how to obtain the list as well as (s_i, S_i) and compute its complexity. Note that in this case, $U(i,x) = f_i(x)$. Hence, $S_i \in \arg\min\{f_i(x)\}$, which in general can be computed in $O(\log(U/\epsilon))$ time using binary search in $[\ell, u]$. Besides, $s_i \in \{x | x \leq S_i, f(x) = f(S_i) + K_i\}$ can also be computed in $O(\log(U/\epsilon))$ time using binary search in $[\ell, u]$. Note that there can be a case where $s_i \leq \ell$. If so, s_i does not need to be computed since it never helps us compute the optimal solution, as $x^*_{p(i)} \in [\ell, u]$ indicated by Theorem 3 and $x^*_{p(i)} \geq s_i$ always holds, which means we do not need to make the increase at this node according to Theorem 3(b). Hence, by Theorem 3(b), we know that

$$V(i,x) = \begin{cases} K_i + f_i(S_i), & x \leq s_i, \\ f_i(x), & \text{o.w.} \end{cases}$$

It can be easily seen that the representation list for $V(i,x)$ is

$$\text{List}_i = [(-\infty, s_i, K_i + f_i(S_i), \emptyset), (s_i, \infty, 0, \{i\})],$$

and the overall complexity of obtaining this list as well as (s_i, S_i) is $O(\log(U/\epsilon))$. Hence, for all $i \in \mathcal{L}$, this step takes $O(|\mathcal{L}| \log(U/\epsilon)) = O(n \log(U/\epsilon))$ time.

Step B. Computation for Non-leaf Nodes: For each $i \in [n]\backslash\mathcal{L}$, now assume that List_j and (s_j, S_j) have already been computed for all $j \in \mathcal{C}(i)$. We show that the list representation (9) also exists for $V(i,x)$, discuss how to compute List_i as well as (s_i, S_i), and how to compute its complexity. We further divide the process into several sub-procedures:

Step B.1. Get List Representation of $U(i,x)$: We first obtain the list representation of $U(i,x) = f_i(x) + \sum_{j\in\mathcal{C}(i)} V(j,x)$ from List_j for each $j \in \mathcal{C}(i)$. Since $f_i(x)$ is convex and $V(j,x)$ is piece-wise convex for each $j \in \mathcal{C}(i)$, $U(i,x)$ is also piece-wise convex, whose breakpoints can be obtained by taking the distinct union of breakpoints of $V(j,x)$'s for all $j \in \mathcal{C}(i)$ and sort them increasingly, i.e.,

$$[\widetilde{b}^i_1, \widetilde{b}^i_2, \ldots, \widetilde{b}^i_{\widetilde{R}_i}] = \text{sort}\left(\bigcup_{j\in\mathcal{C}(i)} [b^j_1, b^j_2, \ldots, b^j_{R_j-1}]\right),$$

where $\text{sort}(\cdot)$ is the operator of returning an increasingly sorted input list, and $\widetilde{b}^i_1, \widetilde{b}^i_2, \ldots, \widetilde{b}^i_{\widetilde{R}_i-1}$ are $\widetilde{R}_i - 1$ many increasingly ordered distinct breakpoints of $U(i,x)$. Since each $[b^j_1, b^j_2, \ldots, b^j_{R_j}]$ is an increasingly sorted list, to get the merged increasingly sorted list $[\widetilde{b}^i_1, \widetilde{b}^i_2, \ldots, \widetilde{b}^i_{\widetilde{R}_i-1}]$, the best-known complexity is $O(\log(|\mathcal{C}(i)|)\sum_{i\in\mathcal{C}(i)} R_j)$ achieved by using heap (see, for example, Cormen et al. (2022)). By Theorem 3(c) we know that $\sum_{i\in\mathcal{C}(i)} R_j \le \sum_{i\in\mathcal{C}(i)}(|N_{\mathcal{T}(j)}| + |\mathcal{L}(j)|) \le |N_{\mathcal{T}(i)}| + |\mathcal{L}(i)| - 1$, hence we know that the complexity can be represented as $O(\log(|\mathcal{C}(i)|)(|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|))$.

Besides $\widetilde{b}^i_1, \widetilde{b}^i_2, \ldots, \widetilde{b}^i_{\widetilde{R}_i-1}$, we should also get $\widetilde{L}^i_1, \widetilde{L}^i_2, \ldots, \widetilde{L}^i_{\widetilde{R}_i}$ and $\widetilde{\mathcal{J}}^i_1, \widetilde{\mathcal{J}}^i_2, \ldots, \widetilde{\mathcal{J}}^i_{\widetilde{R}_i}$ in order to get the full list representation of $U(i,x)$ as

$$\widetilde{\text{List}}_i = \left[(-\infty, \widetilde{b}^i_1, \widetilde{L}^i_1, \widetilde{\mathcal{J}}^i_1), (\widetilde{b}^i_1, \widetilde{b}^i_2, \widetilde{L}^i_2, \widetilde{\mathcal{J}}^i_2), \ldots, (\widetilde{b}^i_{\widetilde{R}_i-1}, \infty, \widetilde{L}^i_{\widetilde{R}_i}, \widetilde{\mathcal{J}}^i_{\widetilde{R}_i})\right].$$

Denote $\widetilde{b}^i_0 = -\infty$ and $\widetilde{b}^i_{\widetilde{R}_i} = \infty$. For each $r \in [\widetilde{R}_i]$ and $j \in \mathcal{C}(i)$, let $r_j(r) \in [R_j]$ be the index such that $(\widetilde{b}^i_{r-1}, \widetilde{b}^i_r) \in (b^j_{r_j-1}, b^j_{r_j})$. Hence, we can calculate $\widetilde{L}^i_r$ and $\mathcal{J}^i_r$ through

$$\widetilde{L}^i_r = \sum_{j\in\mathcal{C}(i)} L^j_{r_j(r)}, \quad \widetilde{\mathcal{J}}^i_r = \{i\} \cup \left(\bigcup_{j\in\mathcal{C}(i)} \mathcal{J}^j_{r_j(r)}\right), \quad \text{for each } r \in [\widetilde{R}_i].$$

Those summations and unions operations for all $r \in [\widetilde{R}_i]$ can be done in $O(\widetilde{R}_i \sum_{j\in\mathcal{C}(i)} |\mathcal{J}^j_{r_j(r)}|)$ time (note that by maintaining a pointer of each list $[\widetilde{b}^j_1, \widetilde{b}^j_2, \ldots, \widetilde{b}^j_{\widetilde{R}_j-1}]$ for $j \in \mathcal{C}(i)$, we can decide $r_j(1), r_j(2), \ldots, r_j(\widetilde{R}_i)$ sequentially by moving the pointer gradually to the right in the list and never returning back, which requires time $O(\widetilde{R}_i|\mathcal{C}(i)|)$ for all $j \in \mathcal{C}(i)$ and does not harm this complexity). Since $\widetilde{R}_i \le \sum_{j\in\mathcal{C}(i)} R_j \le |N_{\mathcal{T}(i)}| + |\mathcal{L}(i)| - 1$, and $\sum_{j\in\mathcal{C}(i)} |\mathcal{J}^j_{r_j(r)}| \le \sum_{j\in\mathcal{C}(i)} |N_{\mathcal{T}(j)}| = |N_{\mathcal{T}(i)}| - 1$, we can re-express this complexity as $O(|N_{\mathcal{T}(i)}|(|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|))$.

In conclusion, the overall complexity of getting the list representation $\widetilde{\text{List}}_i$ of $U(i,x)$ is $O(|N_{\mathcal{T}(i)}|(|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|))$. For all $i \in [n]\backslash\mathcal{L}$, this step takes $O(\sum_{i\in[n]\backslash\mathcal{L}} |N_{\mathcal{T}(i)}|(|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|)) = O(n^3)$ time.

Step B.2. Obtain (s_i, S_i)**:** Now given the list representation $\widetilde{\text{List}}_i$ obtained from the last step for $U(i,x)$, we want to compute (s_i, S_i). According to the definition, $S_i \in \arg\min U(i,x)$. Then, to compute S_i, we only need to compute a minimizer in each convex piece and choose the one with the minimum function value. On r-th piece, $U(i,x) = \widetilde{L}^i_r + \sum_{j\in\widetilde{\mathcal{J}}^i_r} f_j(x)$, whose evaluation for any

particular x in the piece takes $O(|\widetilde{\mathcal{J}}_r^i|)$ time. Note that the binary search only needs to be implemented in r-th piece intersecting with $[\ell, u]$ (the range that any possible x_i^* must locate in), which is denoted as $\mathcal{B}_r^i$ with interval length B_r^i. If $\mathcal{B}_r^i = \emptyset$ and $B_r^i = 0$, interval r can be removed or never focused; otherwise, the binary search algorithm needs to be applied on $\mathcal{B}_r^i$ using $O(|\widetilde{\mathcal{J}}_r^i| \log(B_r^i/\epsilon))$ time. Hence, the overall complexity to find S_i is $O(\sum_{r\in[\widetilde{R}_i]} |\widetilde{\mathcal{J}}_r^i| \log(B_r^i/\epsilon))$ time. Using the fact that $|\widetilde{\mathcal{J}}_r^i| \le |N_{\mathcal{T}(i)}|$ and $B_r^i \le U$ for any $r \in \widetilde{R}_i$, the complexity can be further simplified as $O(\widetilde{R}_i |N_{\mathcal{T}(i)}| \log(U/\epsilon)) = O((|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|)|N_{\mathcal{T}(i)}| \log(U/\epsilon))$.

To compute $s_i \in \{x | x \le S_i, U(i,x) = U(i,S_i) + K_i\}$, we first need to find the convex piece in $\widetilde{\text{List}}_i$ (say, r_{s_i}-th piece) whose left endpoint $\widetilde{b}^i_{r_{s_i}-1} \le S_i$ and satisfies

$$U(i, \widetilde{b}^i_{r_{s_i}-1}) \ge U(i,S_i) + K_i,$$
$$\text{and} \qquad U(i, \widetilde{b}^i_{r_{s_i}}) \le U(i,S_i) + K_i.$$

Hence, from Lemma EC.2(d) and Theorem 3(b), we know that r_{s_i}-th piece is where s_i is located. Hence, we only need to conduct a binary search in this interval to obtain s_i. Finding r_{s_i}-th convex piece requires $O(\widetilde{R}_i)$ time and the binary search in r_{s_i}-th interval takes $O(\log(B^i_{r_{s_i}}/\epsilon))$ time. Hence, the overall complexity is $O(\widetilde{R}_i + \log(B^i_{r_{s_i}}/\epsilon)) = O(|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)| + \log(U/\epsilon))$ time.

In conclusion, the time for obtaining (s_i, S_i) from $\widetilde{\text{List}}_i$ is $O((|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|)|N_{\mathcal{T}(i)}| \log(U/\epsilon))$. Therefore, for all $i \in [n]\backslash\mathcal{L}$, this step takes $O(\sum_{i\in[n]\backslash\mathcal{L}}(|N_{\mathcal{T}(i)}| + |\mathcal{L}(i)|)|N_{\mathcal{T}(i)}| \log(U/\epsilon)) = O(n^3 \log(U/\epsilon))$ time.

Step B.3. Get List Representation of $V(i,x)$: With $\widetilde{\text{List}}_i$ and (s_i, S_i) pre-computed, we are now ready to get the list representation List_i of $V(i,x)$. Recall that $V(i,x)$ can be expressed as

$$V(i,x) = \begin{cases} K_i + U(i,S_i), & x \le s_i, \\ U(i,x), & \text{o.w.} \end{cases}$$

From this, we can see that List_i can be obtained from $\widetilde{\text{List}}_i$ by executing the following operations: we first delete all tuples in $\widetilde{\text{List}}_i$ whose left endpoint is less than or equal to s_i, i.e., delete the first until the r_{s_i}-th tuple. After that, we append two tuples $(-\infty, s_i, K_i + U(i,S_i), \emptyset)$ and $(s_i, \widetilde{b}_{r_{s_i}}, \widetilde{L}_{r_{s_i}}, \widetilde{\mathcal{I}}_{r_{s_i}})$ on the left side of the list. It only consumes $O(1)$ time. Hence, for all $i \in [n]\backslash\mathcal{L}$, this step consumes $O(n)$ time.

Step C. Recover the Optimal Solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ Using Theorem 3(b), we can easily recover the optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ in $O(n)$ time given (s_i, S_i)'s for any $i \in [n]$ have already been obtained.

Putting all the analysis above together, we can conclude that there exists a list representation (8) for each $V(i,x)$, whose functional form can be recovered from the list representation according to (9), and the optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ of Arborescence problem can be obtained in $O(n^3 \log(U/\epsilon))$ time. □

B.5. Proof of Lemma 2

LEMMA 2. *The value-to-go function $V(i,x)$ is piece-wise convex for any $i \in N_{\mathcal{T}}$, regardless of the arborescence structure.*

Proof. The proof is conducted by induction. We first check the statement for each $i \in \mathcal{L}$. From Theorem 3 and the fact that $U(i,x) = f_i(x)$, we know that

$$V(i,x) = \begin{cases} K_i + f_i(S_i), & x \le s_i, \\ f_i(x), & \text{o.w.} \end{cases}$$

where S_i and s_i are defined by (7a) and (7b). Therefore, $V(i,x)$ is piece-wise convex with two convex pieces.

Next, for any $i \in N_{\mathcal{T}} \backslash \mathcal{L}$, assume $V(j,x)$ is piece-wise convex for each $j \in \mathcal{C}(i)$. Recall that

$$V(i,x) = \min_{x' \ge x} \left\{ K_i \mathbb{1}\{x' > x\} + U(i,x') \right\}.$$

Since the summation of piece-wise convex functions is still piece-wise convex, we know that $U(i,x) = f_i(x) + \sum_{j \in \mathcal{C}(i)} V(j,x)$ is also a piece-wise convex function. For convenience, let us express its convex pieces to be $P_1 = (-\infty, c_1], P_2 = (c_1, c_2], \ldots, P_L = (c_{L-1}, \infty)$, and the minimum at each convex piece to be

$$\widehat{x}_\ell = \min \left\{ x \,|\, x \in \arg\min_{y \in P_\ell} U(i,y) \right\}, \quad \forall \ell \in [L].$$

Now define a map $S_i(x) : (-\infty, \widehat{x}_L) \to \{\widehat{x}_1, \ldots \widehat{x}_L\}$ as

$$S_i(x) = \arg\min_{y \in \{\widehat{x}_{\ell(x)}, \ldots, \widehat{x}_L\}} U(i,y),$$

where $\ell(x) = \min\{\ell \in [L] \,|\, \widehat{x}_\ell > x\}$. $S_i(x)$ represents the optimal minimum point of $U(i,x)$ in all convex pieces with its value greater than x. Hence, $V(i,x)$ can be re-expressed as

$$V(i,x) = \begin{cases} \min\{U(i,x),\, K_i + U(i,S_i(x))\} & x \in (-\infty, \widehat{x}_L) \\ U(i,x) & x \in [\widehat{x}_L, \infty) \end{cases}.$$

It is because when $x \in (-\infty, \widehat{x}_L)$, due to the piece-wise convexity of $K_i + U(i,x)$, as long as making an increase is a better decision compared with holding still at i, the increased level should be the best minimum point of $U(i,x)$ in all convex pieces with its value greater than x, i.e., $S_i(x)$; when $x \in [\widehat{x}_L, \infty)$, $U(i,x)$ is non-decreasing, which means that the optimal decision is to make no increase. Besides, observe that $S_i(x)$ is a non-decreasing right continuous step function with at most L steps, and we can define its discontinuous points as $q_1, q_2, \ldots, q_{\tau-1}$ (note that $\{q_1, \ldots, q_{\tau-1}\} \subseteq \{\widehat{x}_1, \ldots, \widehat{x}_L\}$), such that $S_i(x)$ holds to be the same value in $Q_1 = (-\infty, q_1), Q_2 = [q_1, q_2), \ldots, Q_\tau = [q_{\tau-1}, \infty)$, with

$\tau \le L$. Let the value of $S_i(x)$ evaluated in $Q_1, \dots, Q_\tau$ to be $S_i^1, \dots, S_i^\tau$, then $V(i,x)$ can be further evaluated as

$$V(i,x) = \begin{cases} \min\{U(i,x),\, K_i + U(i,S_i^1) & x \in Q_1, \\ \min\{U(i,x),\, K_i + U(i,S_i^2) & x \in Q_2, \\ \dots & \\ \min\{U(i,x),\, K_i + U(i,S_i^\tau) & x \in Q_\tau, \\ U(i,x) & x \in [\widehat{x}_L, \infty) \end{cases}.$$

It just means that in each Q_t for $t \in [\tau]$, the functional form of $V(i,x)$ is obtained from $U(i,x)$ by changing the evaluation at the region where $U(i,x) \ge K_i + U(i,S_i^t)$ to be $K_i + U(i,S_i^t)$ and holding the evaluation the same as $U(i,x)$ in the remaining region of Q_t. Geometrically speaking, it just means that several flat pieces evaluated as $K_i + U(i,S_i^t)$ might be introduced. Hence, $V(i,x)$ is still piece-wise convex in Q_t. In conclusion, $V(i,x)$ is a piece-wise convex function. □

B.6. Proof of Lemma 3

LEMMA 3. *Suppose g is any piece-wise convex function with $L \in \mathbb{Z}_+$ number of convex pieces and $R \in \mathbb{Z}_+$ number of quasi-convex pieces, where $R \le L$. Then, for any $K \in \mathbb{R}_+$, the function $h : \mathbb{R} \to \mathbb{R}$ defined by*

$$h(x) = \min_{x' \ge x} \left\{ K \mathbb{1}\{x' > x\} + g(x') \right\}$$

is piece-wise convex with at most $L + R$ number of convex pieces.

Proof. For g, let its convex pieces be $P_1 = (-\infty, c_1], P_2 = (c_1, c_2], \dots, P_L = (c_{L-1}, \infty)$, and quasi-convex pieces be $\widetilde{P}_1 = (-\infty, \widetilde{c}_1], \widetilde{P}_2 = (\widetilde{c}_1, \widetilde{c}_2], \dots, \widetilde{P}_R = (\widetilde{c}_{R-1}, \infty)$. Define the minimum of g in each quasi-convex piece to be:

$$\widetilde{x}_r = \min \left\{ x | x \in \arg\min_{y \in \widetilde{P}_r} g(y) \right\}, \quad \forall r \in [R].$$

Following the similar idea as in the proof of Lemma 2, but now for quasi-convex pieces instead of convex pieces, define a map $\widetilde{S}(x) : (-\infty, \widetilde{x}_R) \to \{\widetilde{x}_1, \dots \widetilde{x}_R\}$ as

$$\widetilde{S}(x) = \arg\min_{y \in \{\widetilde{x}_{r(x)}, \dots, \widehat{x}_R\}} g(y),$$

where $r(x) = \min\{r \in [R] \,|\, \widetilde{x}_r > x\}$. $\widetilde{S}(x)$ represents the optimal minimum point in all quasi-convex pieces with its value greater than x. By nature of the weak unimodality of $g(y)$ in each quasi-convex piece, with the same reason as mentioned in the proof of Lemma 2, $h(x)$ can be re-expressed as

$$h(x) = \begin{cases} \min\{g(x)\,,\, K + g(\widetilde{S}(x))\} & x \in (-\infty, \widetilde{x}_R) \\ g(x) & x \in [\widetilde{x}_R, \infty) \end{cases}.$$

Now let us partition $(-\infty, \widetilde{x}_R)$ by pieces $\widetilde{Q}_1 = (-\infty, \widetilde{x}_1), \widetilde{Q}_2 = [\widetilde{x}_1, \widetilde{x}_2), \dots, \widetilde{Q}_R = [\widetilde{x}_{R-1}, \widetilde{x}_R)$. Note that $\widetilde{S}(x)$ is constantly evaluated in each $\widetilde{Q}_r$ for $r \in [R]$ (actually, $\widetilde{S}(x)$ may even be constantly evaluated

in several consecutive $\widetilde{Q}_r$). Define the evaluation of $\widetilde{S}(x)$ in $\widetilde{Q}_1,\ldots,\widetilde{Q}_R$ to be $\widetilde{S}^1,\ldots,\widetilde{S}^R$. Then, $h(x)$ can be further expressed as:

$$h(x)=\begin{cases}\min\{g(x),K+g(\widetilde{S}^1)\}, & x\in\widetilde{Q}_1,\\ \min\{g(x),K+g(\widetilde{S}^2)\}, & x\in\widetilde{Q}_2,\\ \ldots & \\ \min\{g(x),K+g(\widetilde{S}^R)\}, & x\in\widetilde{Q}_R,\\ g(x) & x\in[\widetilde{x}_R,\infty)\end{cases}.$$

It can be observed that g is non-increasing in $\widetilde{Q}_1$, and weakly unimodal in $\widetilde{Q}_r$ (more specifically, non-decreasing in $[\widetilde{x}_{r-1},\widetilde{c}_{r-1})$ and non-increasing in $[\widetilde{c}_{r-1},\widetilde{x}_r)$) whence quasi-concave in $\widetilde{Q}_r$ since g is univariate, for any $r\in[R]\backslash\{1\}$. Hence, compared with $g(x)$, $\min\{g(x),K+g(\widetilde{S}^r)\}$ can only introduce at most one more convex piece in the $\widetilde{Q}_r$, whose range is $\{x\in\widetilde{Q}_r\,|\,g(x)\geq K+g(\widetilde{S}^r)\}$ and the evaluation of $h(x)$ on it is constantly $K+g(\widetilde{S}^r)$. In conclusion, compared with $g(x)$, $h(x)$ has at most R additional convex pieces (actually flat pieces), which means that $h(x)$ is piece-wise convex with at most $L+R$ number of convex pieces. □

B.7. Proof of Lemma 4

LEMMA 4. *The value-to-go function $V(i,x)$ is piece-wise convex with at most $2^{h(i)}|\mathcal{L}(i)|$ number of convex pieces, regardless of the arborescence structure.*

Proof. The proof is conducted by induction. First, we check the statement for any $i\in\mathcal{L}$, for which $h(i)=|\mathcal{L}(i)|=1$. Then, from the statement of the lemma, $V(i,x)$ should be piece-wise convex with at most 2 convex pieces, which is justified by Theorem 3.

Next, we check the statement for any $i\in N_{\mathcal{T}}\backslash\mathcal{L}$. Suppose the statement in the lemma holds for all $j\in\mathcal{C}(i)$. Recall that the dynamic programming recursion to compute $V(i,x)$ is

$$V(i,x)=\min_{x'\geq x}\left\{K_i\mathbb{1}\{x'>x\}+U(i,x')\right\}.$$

By Lemma 2, we know that $V(i,x)$ is piece-wise convex. From the induction hypothesis, we know that $U(i,x)=f_i(x)+\sum_{j\in\mathcal{C}(i)}V(j,x)$ has at most $\sum_{j\in\mathcal{C}(i)}2^{h(j)}|\mathcal{L}(j)|\leq 2^{h(i)-1}\sum_{j\in\mathcal{C}(i)}|\mathcal{L}(j)|=2^{h(i)-1}|\mathcal{L}(i)|$ number of convex pieces; besides, the number of quasi-convex pieces of $U(i,x)$ can be bounded by the number of its convex pieces. Hence, by Lemma 3, we know that the number of convex pieces of $V(i,x)$ is upper bounded by $2^{h(i)-1}|\mathcal{L}(i)|+2^{h(i)-1}|\mathcal{L}(i)|=2^{h(i)}|\mathcal{L}(i)|$. This completes the proof. □

B.8. Proof of Proposition 6

PROPOSITION 6. *Given a balanced arborescence $\mathcal{T}$, the number of convex pieces of $V(i,x)$ can be bounded by* poly(n) *for each $i\in N_{\mathcal{T}}$.*

Proof. For a balanced arborescence $\mathcal{T}$, using Lemma 4, we know that the number of convex pieces of each $V(i,x)$ can be bounded by

$$2^{h(i)}|\mathcal{L}(i)| \overset{(i)}{\leq} 2^h n \overset{(ii)}{\leq} 2^{O(\log n)} n \leq n^{O(1)},$$

where (i) follows from the fact that $h \leq h(i)$ and $|\mathcal{L}(i)| \leq n$, and (ii) follows from the fact that $\mathcal{T}$ is a balanced arborescence and $h = O(\log n)$. This completes the proof. □

B.9. Proof of Lemma 5

LEMMA 5. *For any $i \in N_{\mathcal{T}}$, let the convex pieces of $V(i,x)$ be $P_1^i = (-\infty, c_1^i], P_2^i = (c_1^i, c_2^i], \ldots, P_{L_i}^i = (c_{L_i-1}^i, \infty)$, with $L_i \in \mathbb{Z}_{++}$ being the number of convex pieces of $V(i,x)$. Denote $\mathcal{E}_j \subseteq \mathbb{R}_+$ as the set of non-smooth points of $f_j(x)$ for any $j \in N_{\mathcal{T}}$. Then,*

(a) For any $c \in \{c_1^i, c_2^i, \ldots, c_{L_i-1}^i\} \backslash \cup_{j \in N_{\mathcal{T}(i)}} \mathcal{E}_j$, it holds that $V'_-(i,c) \geq V'_+(i,c)$.

(b) For any $\ell \in [L_i]$, there exists a constant $K_\ell^i \in \mathbb{R}$ and a set of nodes $\mathcal{D}_\ell^i \subseteq N_{\mathcal{T}(i)}$, such that $V(i,x) \equiv K_\ell^i + \sum_{j \in \mathcal{N}_\ell^i} f_j(x)$ for x in convex piece P_ℓ^i, where $\mathcal{N}_\ell^i = \cup_{k \in \mathcal{D}_\ell^i} \mathcal{P}(i,k)$.

Proof. The proof is conducted by induction. We first check for the base case when $i \in \mathcal{L}$. From Theorem 3 and the fact that $U(i,x) = f_i(x)$, we know that

$$V(i,x) = \begin{cases} K_i + f_i(S_i), & x \leq s_i, \\ f_i(x), & \text{o.w.} \end{cases}$$

where S_i and s_i are defined by (7a) and (7b), and we know that $c_1^i = s_i$. If $K_i = 0$, we have $f'(S_i) = 0$ since $S_i = s_i = c_1^i \notin \mathcal{E}_i$ is a smooth point of $f_i(x)$, which implies $V'_-(i,c_1^i) = 0 = (f_i)'_+(c_1^i) = V'_+(i,c_1^i)$; if $K_i > 0$, then $(f_i)'_+(c_1^i) < 0$, and $V'_-(i,c_1^i) = 0 > (f_i)'_+(c_1^i) = V'_+(i,c_1^i)$, which means that (a) holds to be true. By setting $K_1^i = K_i + f_i(S_i), \mathcal{D}_1^i = \emptyset$ and $K_2^i = 0, \mathcal{D}_2^i = \{i\}$, we can check that (b) also holds to be true.

For any $i \in N_{\mathcal{T}} \backslash \mathcal{L}$, suppose (a) and (b) hold to be true for any $j \in \mathcal{C}(i)$. Note that $U(i,x) = f_i(x) + \sum_{j \in \mathcal{C}(i)} V(j,x)$ is piece-wise convex with breakpoints $\bigcup_{j \in \mathcal{C}(i)} \{c_1^j, c_2^j, \ldots, c_{L_j-1}^j\}$. For any $c \in \bigcup_{j \in \mathcal{C}(i)} \{c_1^j, c_2^j, \ldots, c_{L_j-1}^j\} \backslash \cup_{j \in N_{\mathcal{T}(i)}} \mathcal{E}_j$, we know that:

(i) $(f_i)'_-(c) = (f_i)'_+(c)$ since $c \notin \mathcal{E}_i$ and is a smooth point of f_i.

(ii) For any $j \in \mathcal{C}(i)$, $V'_-(j,c) = V'_+(j,c)$ when $c \notin \{c_1^j, c_2^j, \ldots, c_{L_j-1}^j\}$ due to the equivalent expression of $V(j,x)$ in every convex piece according to the induction hypothesis (b), together with the requirement that $c \notin \cup_{k \in N_{\mathcal{T}(j)}} \mathcal{E}_k$, which means that $V(j,x)$ is smooth at c; $V'_-(j,c) \geq V'_+(j,c)$ when $c \in \{c_1^j, c_2^j, \ldots, c_{L_j-1}^j\}$ due to the induction hypothesis that (a).

Putting them together, we have that for any $c \in \bigcup_{j \in \mathcal{C}(i)} \{c_1^j, c_2^j, \ldots, c_{L_j-1}^j\} \backslash \cup_{j \in \mathcal{T}(i)} \mathcal{E}_j$:

$$U'_-(i,x) = (f_i)'_-(c) + \sum_{j \in \mathcal{C}(i)} V'_-(j,c) \geq (f_i)'_+(c) + \sum_{j \in \mathcal{C}(i)} V'_+(j,c) = U'_+(i,x).$$

Recall the recursion that $V(i,x)=\min_{x'\geq x}\{K_i\mathbb{1}\{x'>x\}+U(i,x')\}$, and those newly cut breakpoints $c\in\{c_1^i,c_2^i,\ldots,c_{L_i-1}^i\}\backslash\bigcup_{j\in\mathcal{C}(i)}\{c_1^j,c_2^j,\ldots,c_{L_j-1}^j\}$ can be created through the way discussed in the proof of Lemma 3. Then, we can apply a similar discussion as in the base case to every newly cut breakpoint that is not in $\cup_{j\in N_{\mathcal{T}(i)}}\mathcal{E}_j$ and show that (a) holds for it. Finally, we show that for any $c\in\{c_1^i,c_2^i,\ldots,c_{L_i-1}^i\}\backslash\cup_{j\in N_{\mathcal{T}(i)}}\mathcal{E}_j$, it holds that $V'_-(i,c)\geq V'_+(i,c)$ and (a) is verified.

To check (b) for $V(i,x)$, first consider any convex piece of $U(i,x)$, say, $\widehat{P}$. From the induction hypothesis that (b) holds for $V(j,x)$ for any $j\in\mathcal{C}(i)$, we have that in $\widehat{P}$, there exists a constant $K^j_{\widehat{P}}\in\mathbb{R}$ and a set of nodes $\mathcal{D}^j_{\widehat{P}}\subseteq N_{\mathcal{T}(j)}$, such that $V(j,x)\equiv K^j_{\widehat{P}}+\sum_{k\in\mathcal{N}^j_{\widehat{P}}}f_k(x)$ where $\mathcal{N}^i_{\widehat{P}}=\cup_{k\in\mathcal{D}^j_{\widehat{P}}}\mathcal{P}(j,k)$. Therefore,

$$\begin{aligned}U(i,x)&=f_i(x)+\sum_{j\in\mathcal{C}(i)}V(j,x)\\&=\sum_{j\in\mathcal{C}(i)}K^j_{\widehat{P}}+f_i(x)+\sum_{j\in\mathcal{C}(i)}\sum_{k\in\mathcal{N}^j_{\widehat{P}}}f_k(x)\\&=\sum_{j\in\mathcal{C}(i)}K^j_{\widehat{P}}+\sum_{k\in\{i\}\cup(\cup_{j\in\mathcal{C}(i)}\mathcal{N}^j_{\widehat{P}})}f_k(x)\\&=\sum_{j\in\mathcal{C}(i)}K^j_{\widehat{P}}+\sum_{k\in\mathcal{N}^i_{\widehat{P}}}f_k(x).\quad\left(\because\mathcal{N}^i_{\widehat{P}}:=\{i\}\cup\left(\cup_{j\in\mathcal{C}(i)}\mathcal{N}^j_{\widehat{P}}\right)\right)\end{aligned}$$

It can be easily observed that $\mathcal{N}^i_{\widehat{P}}=\cup_{k\in\mathcal{D}^i_{\widehat{P}}}\mathcal{P}(i,k)$ where $\mathcal{D}^i_{\widehat{P}}:=\cup_{j\in\mathcal{C}(i)}\mathcal{D}^j_{\widehat{P}}$. Hence, we know that (b) holds to be true for $U(i,x)$. Besides, the function value of $V(i,x)$ on each newly cut flat piece from $U(i,x)$ according to its recursion can be represented using a constant, which means that (b) holds to be true for $V(i,x)$. It concludes the induction and also our proof. □

B.10. Proof of Lemma 6

LEMMA 6. *For any $i\in N_{\mathcal{T}}$, consider any quasi-convex piece (say, $\widetilde{P}$) of $V(i,x)$, we have that any mode of this quasi-convex piece (say, $m_{\widetilde{P}}\in\arg\min_{x\in\widetilde{P}}V(i,x)$) satisfies:*

(a) $m_{\widetilde{P}}\in\cup_{j\in N_{\mathcal{T}(i)}}\mathcal{E}_j$ where $\mathcal{E}_j$ is the set of non-smooth points of $f_j(x)$, if $\widetilde{P}$ is of Type I; or

(b) $m_{\widetilde{P}}\in\arg\min_x\sum_{j\in\mathcal{N}^i}f_j(x)$ where $\mathcal{N}^i=\cup_{k\in\mathcal{D}^i}\mathcal{P}(i,k)$ for some $\mathcal{D}^i\subseteq N_{\mathcal{T}(i)}$, if $\widetilde{P}$ is of Type II.

Proof. (a) By Definition 5(a), when $\widetilde{P}$ is of *Type I*, $m_{\widetilde{P}}$ must be located on some breakpoint c of the convex pieces of $V(i,c)$ such that $V'_-(i,c)<V'_+(i,c)$. However, by Lemma 5(a), as long as the breakpoint $c\notin\cup_{j\in N_{\mathcal{T}(i)}}\mathcal{E}_j$, we have that $V'_-(i,c)\geq V'_+(i,c)$. Hence, we have that $m_{\widetilde{P}}=c\in\cup_{j\in N_{\mathcal{T}(i)}}\mathcal{E}_j$ and statement (a) is proved.

(b) We separately discuss two cases: (i) If $m_{\widetilde{P}}$ is located on some breakpoint c of the convex pieces of $V(i,x)$, then we have $V'_-(i,x)\leq 0\leq V'_+(i,x)$. Since $\widetilde{P}$ is of *Type II*, then by Definition 5(b), we know that $V'_-(i,x)\geq V'_+(i,x)$. Hence, the only possible case is $V'_-(i,x)=V'_+(i,x)=0$. It means that $m_{\widetilde{P}}=c$ is a minimum point of $V(i,x)$ on both the convex pieces on the left and the right of c. Combined with Lemma 5(b), we can prove statement (b). (ii) If $m_{\widetilde{P}}$ is not located on any breakpoint of the

convex pieces in $\widetilde{P}$, then $m_{\widetilde{P}}$ is an unconstrained minimum point of the functional form of $V(i,x)$ in the convex piece containing $m_{\widetilde{P}}$. Combined with Lemma 5(b), we can also prove the statement (b). □

B.11. Proof of Lemma 7

LEMMA 7. *Let $\mathcal{P} := \mathcal{P}(1,j)$ with some $j \in \mathcal{L}$, through which we re-index the nodes in $\mathcal{P}$ from the root to the leaf as $1,2,\ldots,|\mathcal{P}|$. For each $\ell \in \cup_{k\in\mathcal{P}}\mathcal{C}(k)\backslash\mathcal{P}$, denote the number of convex pieces of $V(\ell,x)$ by m_ℓ. Under Assumptions 4, for any $i \in \mathcal{P}$, the number of convex pieces of $V(i,x)$ is upper bounded by $O(n^2) + n^2 \sum_{\ell\in\cup_{k\in\mathcal{P}}\mathcal{C}(k)\backslash\mathcal{P}} m_\ell$.*

Proof. To prove the lemma, let us first re-express the dynamic programming recursion for $V(i,x)$ for each $i \in \mathcal{P}$ by grouping the directly-incurred cost at i (i.e., $f_i(x)$) and all value-to-go functions of the child nodes of i except $i+1$ (i.e., $\sum_{j\in\mathcal{C}(i)\backslash\{i+1\}} V(j,x)$) together:

$$\begin{aligned} V(i,x) &= \min_{x'\geq x}\left\{K_i\mathbb{1}\{x'>x\} + U(i,x')\right\} \\ &= \min_{x'\geq x}\left\{K_i\mathbb{1}\{x'>x\} + f_i(x') + \sum_{j\in\mathcal{C}(i)} V(j,x')\right\} \\ &= \min_{x'\geq x}\left\{K_i\mathbb{1}\{x'>x\} + \big(f_i(x') + \sum_{j\in\mathcal{C}(i)\backslash\{i+1\}} V(j,x')\big) + V(i+1,x')\right\} \\ &= \min_{x'\geq x}\left\{K_i\mathbb{1}\{x'>x\} + \widetilde{f}_i(x') + V(i+1,x')\right\} \qquad \left(\because \widetilde{f}_i(x') := f_i(x') + \sum_{j\in\mathcal{C}(i)\backslash\{i+1\}} V(j,x')\right) \end{aligned} \tag{EC.2}$$

where we view node $|\mathcal{P}|+1$ as the only child of node $|\mathcal{P}|$ and let $V(|\mathcal{P}|+1,x) \equiv 0$ for all x. It can be observed from (EC.2) that getting the functional representation of $V(i,x)$ for all $i \in \mathcal{P}$ is the same as doing isotonic optimization on only a path with nodes of $\mathcal{P}$, except that each convex cost function $f_i(x)$ should be substituted by a piece-wise convex cost function $\widetilde{f}_i(x)$ which has at most $\max\{1, \sum_{j\in\mathcal{C}(i)\backslash\{i+1\}} m_j\}$ number of convex pieces. Based on such a transformation, for each $i \in \mathcal{P}$, let us separately bound the number of *Type I* and *Type II* quasi-convex pieces of $\widetilde{f}_i(x) + V(i+1,x)$ to bound the newly introduced convex pieces of $V(i,x)$ compared with $\widetilde{f}_i(x) + V(i+1,x)$, and finally bound the number of overall convex pieces of each $V(i,x)$.

Step A: bounding the number of *Type I* quasi-convex pieces of $\widetilde{f}_i(x) + V(i+1,x)$: since $\widetilde{f}_i(x) + V(i+1,x)$ is exactly $U(i,x)$, by Lemma 6(a) and our discussion before, the number of its *Type I* quasi-convex pieces is at most the number of all non-smooth points of $f_j(x)$'s for all $j \in N_{\mathcal{T}(i)}$, which is bounded by $O(|N_{\mathcal{T}(i)}|)$ when Assumption 4 is made.

Step B: bounding the number of *Type II* quasi-convex pieces of $\widetilde{f}_i(x) + V(i+1,x)$: the quasi-convex pieces with modes at non-smooth points of $f_j(x)$'s for all $j \in \mathcal{T}(i)$ are not considered

anymore in this part since they are already counted in the last part (even the corresponding quasi-convex pieces might be of *Type II*). We partition the domain into several regions, with the partition changing dynamically during the dynamic programming recursion. More specifically, for a given $i \in \mathcal{P}$, we let $\cup_{j=1}^{|\mathcal{P}|+1} \mathcal{F}_j^i = \mathbb{R}_+$ with $\mathcal{F}_j^i \cap \mathcal{F}_k^i = \emptyset$ for any pair (j,k) and $j \neq k$ be the partition of the domain after the functional form of $V(i,x)$ is obtained based on the DP recursion. Those $\mathcal{F}_j^i$'s can be interpreted as follows: $\mathcal{F}_{|\mathcal{P}|+1}^i$ represents the region where no new flat piece is introduced until the DP recursion for $V(i,x)$ is executed; $\mathcal{F}_j^i$ for $i \leq j \leq |\mathcal{P}|$ represents the region that is: (i) a part of the union of intervals of newly introduced flat pieces upon $V(j,x)$ is evaluated (ii) no new flat pieces are introduced in it after that till $V(i,x)$ is evaluated. Let us characterize how those partitions are obtained along the DP recursion, through which the meaning of the partition will be clearer:

- Initialize $\mathcal{F}_i^{|\mathcal{P}|+1} \leftarrow \emptyset$ for all $i \in \mathcal{P}$, and $\mathcal{F}_{|\mathcal{P}|+1}^{|\mathcal{P}|+1} \leftarrow \mathbb{R}_+$.
- Each time when the functional form of $V(i,x)$ is obtained for some $i \in \mathcal{P}$ by solving the DP recursion, we obtain $\mathcal{F}_j^i$ for all $i \leq j \leq |\mathcal{P}|+1$ from $\mathcal{F}_j^{i+1}$ for all $i+1 \leq j \leq |\mathcal{P}|+1$ as follows:

— $\mathcal{F}_i^i$ is assigned to be the union of the regions where $V(i,x)$ has different function values compared with $\widetilde{f}_i(x) + V(i+1,x)$ (any such region is left open and right closed).

— $\mathcal{F}_j^i \leftarrow \mathcal{F}_j^{i+1} \setminus \mathcal{F}_i^i$ for all $i \leq j \leq |\mathcal{P}|+1$, meaning that $\mathcal{F}_j^i$ is derived from $\mathcal{F}_j^{i+1}$ by excluding the region corresponding to $\mathcal{F}_i^i$.

Now, let us consider when the functional form of $V(i+1,x)$ is obtained for some $i \in \mathcal{P}$. Note that $\mathcal{F}_j^{i+1}$ can only be nonempty for $i+1 \leq j \leq |\mathcal{P}|+1$. For each $\mathcal{F}_j^{i+1}$ with $i+1 \leq j \leq |\mathcal{P}|+1$, we know that if it is non-empty, it must be composed of several pieces, which we denote as $P_{j,1}^{i+1}, P_{j,2}^{i+1}, \ldots, P_{j,L_j}^{i+1} \subseteq \mathbb{R}_+$ for some $L_j \in \mathbb{Z}_+$ with $\bigcup_{\ell \in [L_j]} P_{j,\ell}^{i+1} = \mathcal{F}_j^{i+1}$ and $P_{j,\ell_1}^{i+1} \cap P_{j,\ell_2}^{i+1} = \emptyset$ if $\ell_1 \neq \ell_2$, where each of those pieces is a part of some piece introduced when obtaining $V(j,x)$, and in each of them no new flat pieces are introduced after that till the functional form of $V(i,x)$ is obtained. From the interpretation of $\mathcal{F}_j^{i+1}$, we know that the functional form of $V(i+1,x)$ on each $P_{j,\ell}^{i+1}$ for any $\ell \in [L_j]$ is the summation of a constant (i.e., the evaluation of $V(j,x)$ on $P_{j,\ell}^{i+1}$ which is a flat piece right after $V(j,x)$ is obtained) and $\widetilde{f}_k(x)$'s with k from $j-1$ back to $i+1$ (piecewise functions we add from round $j-1$ back to i through dynamic programming recursion). Hence, there exists $K_{j,1}^{i+1}, K_{j,2}^{i+1}, \ldots K_{j,L_j}^{i+1} \in \mathbb{R}$ such that

$$\begin{aligned} & V(i+1,x) \equiv K_{j,\ell}^{i+1} + \sum_{k=i+1}^{j-1} \widetilde{f}_k(x), \qquad \forall x \in P_{j,\ell}^{i+1},\ \forall \ell \in [L_j], \\ \Longrightarrow \quad & \widetilde{f}_i(x) + V(i+1,x) \equiv K_{j,\ell}^{i+1} + \sum_{k=i}^{j-1} \widetilde{f}_k(x), \qquad \forall x \in P_{j,\ell}^{i+1},\ \forall \ell \in [L_j]. \end{aligned} \tag{EC.3}$$

Combining Lemma 6(b) and the requirement that the mode of any *Type II* quasi-convex piece of $V(i,x)$ we consider in this part never locates on any non-smooth point of $f_j(x)$ for any $j \in \mathcal{T}(i)$, it can be easily verified that if $\widehat{x} \in \mathbb{R}$ is a mode of some *Type II* quasi-convex piece of $\widetilde{f}_i(x) + V(i+1,x)$, then

$\widetilde{f}_i'(\widehat{x})+V'(i+1,\widehat{x})=0$. From (EC.3) we know that on any $\mathcal{F}_j^{i+1}$ with $i+1\le j\le|\mathcal{P}|+1$, the functional form of $\widetilde{f}_i(x)+V(i+1,x)$ on each piece $P_{j,\ell}^{i+1}$ for any $\ell\in[L_j]$ only differs in its constant term and has no difference when taking the derivative. Therefore, any $\widehat{x}\in\mathcal{F}_j^{i+1}$ with $\widetilde{f}_i'(\widehat{x})+V'(i+1,\widehat{x})=0$ must satisfy that $(\sum_{k=i}^{j-1}\widetilde{f}_k)'(\widehat{x})=0$. However, $\sum_{k=i}^{j-1}\widetilde{f}_k(x)=\sum_{k=i}^{j-1}f_k(x)+\sum_{k=i}^{j-1}\sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}}V(\ell,x)$ has at most $\max\{1,\sum_{k=i}^{j-1}\sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}}m_\ell\}$ number of convex pieces. We claim that each convex piece of $\sum_{k=i}^{j-1}\widetilde{f}_k(x)$ gives rise to at most one *Type II* quasi-convex piece whose mode lies in $\mathcal{F}_j^{i+1}$. To see this, consider a convex piece of $\sum_{k=i}^{j-1}\widetilde{f}_k(x)$ with minimizers in $[a,b]$ and analyze two cases:

- If a and b fall within the same segment of $\mathcal{F}_j^{i+1}$ or at most one of a and b falls within any segment of $\mathcal{F}_j^{i+1}$, then the entire minimizer interval contributes at most one *Type II* mode.
- If a and b lie in different segments of $\mathcal{F}_j^{i+1}$, suppose a belongs to segment P_- with adjacent right segment P_+, and let p denote their breakpoint. By Lemma 5, $V_-'(i+1,p)\ge V_+'(i+1,p)$. If $V_-'(i+1,p)>V_+'(i+1,p)$, since $(\widetilde{f}_i)_-'(p)=(\widetilde{f}_i)_+'(p)=0$, it follows that $0=(\sum_{k=i}^{j-1}\widetilde{f}_k)_-'(p)=(\widetilde{f}_i(x)+V(i+1,x))_-'(p)>(\widetilde{f}_i(x)+V(i+1,x))_+'(p)$, so no mode arises in P_-. If $V_-'(i+1,p)=V_+'(i+1,p)=0$, then a mode exists at p but is shared by not only P_- but also P_+ (where the mode on P_+ corresponds to some convex piece of $\sum_{k=i}^{j'-1}\widetilde{f}_k(x)$ given $P_+\in\mathcal{F}_{j'}^{i+1}$), so it should only be counted once. A similar argument applies to b. Consequently, at most one mode will be counted in total.

Therefore, each convex piece contributes at most one *Type II* quasi-convex piece with mode in $\mathcal{F}_j^{i+1}$. Hence, we know that the number of *Type II* quasi-convex pieces of $\widetilde{f}_i(x)+V(i+1,x)$ on $\mathcal{F}_j^{i+1}$ is at most $\max\{1,\sum_{k=i}^{j-1}\sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}}m_\ell\}$. Consequently, the number of overall *Type II* quasi-convex pieces of $\widetilde{f}_i(x)+V(i+1,x)$ on $\mathbb{R}=\cup_{j=i+1}^{|\mathcal{P}|+1}\mathcal{F}_j^{i+1}$ is at most $\sum_{j=i+1}^{|\mathcal{P}|+1}\max\{1,\sum_{k=i}^{j-1}\sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}}m_\ell\}$.

Step C: bounding the number of convex pieces of $V(i,x)$: combining the analysis above and using the fact that the number of convex pieces of $V(i,x)$ is bounded by the summation of the number of convex pieces of $\widetilde{f}_{|\mathcal{P}|}(x)$ (the base number of convex pieces before doing dynamic programming recursion and introducing any new flat piece) and the number of quasi-convex pieces of $\widetilde{f}_{|\mathcal{P}|}(x)+V(|\mathcal{P}|+1,x),\widetilde{f}_{|\mathcal{P}|-1}(x)+V(|\mathcal{P}|,x),\dots,\widetilde{f}_i(x)+V(i+1,x)$ (the number of newly introduced convex pieces when the functional forms of $V(|\mathcal{P}|,x),V(|\mathcal{P}|-1,x),\dots,V(i,x)$ are obtained). Hence, the number of convex pieces of $V(i,x)$ is bounded by:

$$\underbrace{\max\left\{1,\sum_{\ell\in\mathcal{C}(|\mathcal{P}|)\setminus\{|\mathcal{P}|+1\}}m_\ell\right\}}_{\text{upper bound on \# of convex pieces of }\widetilde{f}_{|\mathcal{P}|}(x)}+\sum_{i'=i}^{|\mathcal{P}|}\Bigg(\underbrace{O\left(|N_{\mathcal{T}(i')}|\right)+\sum_{j=i'+1}^{|\mathcal{P}|+1}\max\left\{1,\sum_{k=i'}^{j-1}\sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}}m_\ell\right\}}_{\text{upper bound on \# of quasi-convex piece of }\widetilde{f}_{i'}(x)+V(i'+1,x)}\Bigg)$$

$$=\underbrace{\max\left\{1,\sum_{\ell\in\mathcal{C}(|\mathcal{P}|)\setminus\{|\mathcal{P}|+1\}}m_\ell\right\}}_{(a)}+\underbrace{\sum_{i'=i}^{|\mathcal{P}|}O\left(|N_{\mathcal{T}(i')}|\right)}_{(b)}+\underbrace{\sum_{i'=i}^{|\mathcal{P}|}\sum_{j=i'+1}^{|\mathcal{P}|+1}\max\left\{1,\sum_{k=i'}^{j-1}\sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}}m_\ell\right\}}_{(c)}.$$

Let us further bound (a),(b), and (c) separately. (a) is dominated in the order by (c) and does not need to be considered. (b) can be bounded as follows:

$$(b) \overset{(i)}{\leq} \sum_{i'=i}^{|\mathcal{P}|} O(n) \overset{(ii)}{\leq} O(n^2),$$

where (i) holds because $|N_{\mathcal{T}(i)}| \leq n$, and (ii) holds because $|\mathcal{P}| - i \leq n$. (c) can be bounded as follows:

$$\begin{aligned}(c) &\overset{(i)}{\leq} \sum_{i'=i}^{|\mathcal{P}|} \sum_{j=i'+1}^{|\mathcal{P}|+1} \left(1 + \sum_{k=i'}^{j-1} \sum_{\ell \in \mathcal{C}(k)\setminus\{k+1\}} m_\ell\right) \\ &\overset{(ii)}{\leq} \sum_{i'=i}^{|\mathcal{P}|} \sum_{j=i'+1}^{|\mathcal{P}|+1} \left(1 + \sum_{\ell \in \cup_{k\in\mathcal{P}}\mathcal{C}(k)\setminus\mathcal{P}} m_\ell\right) \\ &\overset{(iii)}{\leq} n^2 + n^2 \sum_{\ell \in \cup_{k\in\mathcal{P}}\mathcal{C}(k)\setminus\mathcal{P}} m_\ell,\end{aligned}$$

where (i) holds since $\max\{1, \sum_{k=i'}^{j-1} \sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}} m_\ell\} \leq 1 + \sum_{k=i'}^{j-1} \sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}} m_\ell$ for any $k \in \mathcal{P}$, (ii) holds since $\sum_{k=i'}^{j-1} \sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}} m_\ell \leq \sum_{k=1}^{|\mathcal{P}|} \sum_{\ell\in\mathcal{C}(k)\setminus\{k+1\}} m_\ell = \sum_{\ell\in\cup_{k\in\mathcal{P}}\mathcal{C}(k)\setminus\mathcal{P}} m_\ell$, and (iii) holds since $\sum_{i'=i}^{|\mathcal{P}|} \sum_{j=i'+1}^{|\mathcal{P}|+1} 1 \leq n^2$.

Putting them all together, we conclude that the number of convex pieces of $V(i,x)$ is upper bounded by $O(n^2) + n^2 \sum_{\ell\in\cup_{k\in\mathcal{P}}\mathcal{C}(k)\setminus\mathcal{P}} m_\ell$. □

B.12. Proof of Theorem 4

Proof. We check the statement by induction. First, let us check the base case when $k = 1$, i.e., $\mathcal{T}$ is a 1-round-branching arborescence. From Definition 6, we know that any 1-round-branching arborescence is a path. Applying Lemma 7, we know that the number of convex pieces of the value-to-go function of $V(i,x)$ for any $i \in N_{\mathcal{T}}$ is $O(n^2)$. Hence, the statement for the base case when $k = 1$ is verified.

Suppose the statement holds for all ℓ-round-branching arborescences with $\ell \in [k-1]$ with $k \geq 2$. Let us check the statement for any k-round-branching arborescence $\mathcal{T}$. Let $\mathcal{P}$ be the set of all its 1-round nodes, i.e., all nodes on the 1-round branch of $\mathcal{T}$. It can be easily observed that $\mathcal{T}(j)$ for any $j \in (\cup_{i\in\mathcal{P}}\mathcal{C}(i))\setminus\mathcal{P}$ is a $(k-1)$-round-branching arborescence, and by our induction hypothesis, $V(j,x)$ has at most $O(|N_{\mathcal{T}(j)}|^{2(k-1)})$ number of convex pieces. According to Lemma 7, we have that for any $i \in \mathcal{P}$, the number of convex pieces of $V(i,x)$ is upper bounded by

$$\begin{aligned}&O(n^2) + n^2 \sum_{j\in(\cup_{i\in\mathcal{P}}\mathcal{C}(i))\setminus\mathcal{P}} O\left(|N_{\mathcal{T}(j)}|^{2(k-1)}\right) \\ \overset{(a)}{\leq}& O(n^2) + n^2 O\left(\left(\sum_{j\in(\cup_{i\in\mathcal{P}}\mathcal{C}(i))\setminus\mathcal{P}} |N_{\mathcal{T}(j)}|\right)^{2(k-1)}\right) \\ \overset{(b)}{\leq}& O(n^2) + n^2 O(n^{2(k-1)}) \\ =& O(n^{2k}),\end{aligned}$$

where (a) holds because $\sum_{i\in[n]} a_i^m \le (\sum_{i\in[n]} a_i)^m$ for any nonnegative a_i's and $m \ge 1$, and (b) holds because $\sum_{j\in(\cup_{i\in\mathcal{P}}\mathcal{C}(i))\setminus\mathcal{P}} |N_{\mathcal{T}(j)}| \le |N_{\mathcal{T}(i)}| = n$. Hence, the theorem statement is verified for any k-round-branching arborescence. □

Appendix C: Efficiently Implemented DP Recursion in Section 3.1

Algorithm 2 Efficient Implementation of DP Recursion (DP-PL)

Input: f_i's and their corresponding kink points set $(\{k_1^i, \dots, k_{m_i}^i\})$'s, for $i \in [n]$
$\overline{\mathcal{K}} \leftarrow$ the decreasingly sorted list of $\{-\infty\} \cup (\bigcup_{i\in[n]} \{k_1^i, \dots, k_{m_i}^i\})$
procedure DFS(i)
    for $j \in \mathcal{C}(i)$ **do**
        DFS(j)
    end for
    $V_{\rightarrow}(i, \overline{\mathcal{K}}[0]) \leftarrow f_i(\overline{\mathcal{K}}[0]) + \sum_{j\in\mathcal{C}(i)} V(j, \overline{\mathcal{K}}[0])$ ▷ $\overline{\mathcal{K}}[\ell]$ is the ℓ-th element in $\overline{\mathcal{K}}$
    $V_{\uparrow}(i, \overline{\mathcal{K}}[0]) \leftarrow \infty$
    $V(i, \overline{\mathcal{K}}[0]) \leftarrow \min\{V_{\uparrow}(i, \overline{\mathcal{K}}[0]), V_{\rightarrow}(i, \overline{\mathcal{K}}[0])\}$
    for $\ell \in [|\overline{\mathcal{K}}|]$ **do**
        $V_{\rightarrow}(i, \overline{\mathcal{K}}[\ell])) \leftarrow f_i(\overline{\mathcal{K}}[\ell]) + \sum_{j\in\mathcal{C}(i)} V(j, \overline{\mathcal{K}}[\ell])$
        $V_{\uparrow}(i, \overline{\mathcal{K}}[\ell]) \leftarrow \min\{V_{\uparrow}(i, \overline{\mathcal{K}}[\ell-1]), K_i + f_i(\overline{\mathcal{K}}[\ell-1]) + \sum_{j\in\mathcal{C}(i)} V(j, \overline{\mathcal{K}}[\ell-1])\}$
        $V(i, \overline{\mathcal{K}}[\ell]) \leftarrow \min\{V_{\uparrow}(i, \overline{\mathcal{K}}[\ell]), V_{\rightarrow}(i, \overline{\mathcal{K}}[\ell])\}$
    end for
end procedure
Execute DFS(1)
Output: the optimal decisions towards $V(1, -\infty)$