The Subinterval Cover Problem

Anonymous author
Anonymous affiliation

Abstract

In this paper, we introduce the subinterval cover problem. Given a set of interval candidates, each associated with a specific cost factor, we need to select a subinterval from each candidate to cover the entire range. The cost of selecting a subinterval in a candidate interval is determined by multiplying the selected length by the cost factor (the weight). Our goal is to minimize the total cost. This problem has broad applications, such as drone delivery and robot motion planning.

Our first finding is that a natural LP formulation of the problem has an unbounded integrality gap, which shows the challenge of studying this problem. Second, we find polynomial-time dynamic programming algorithms for two special cases: 1) two distinct cost factors; 2) a constant number of distinct cost factors, and the interval candidates form a laminar set family: each pair of intervals is either disjoint, or one interval contains the other. As a corollary of our dynamic programming algorithm, we design a polynomial-time $\sqrt{w_{\max}/w_{\min}}$-approximation algorithm for the general cases, where $w_{\max}$ and $w_{\min}$ are the maximum and minimum weights of the intervals, respectively.

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

Keywords and phrases Subinterval Cover, Approximation Algorithm, Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The Set Cover problem is a classic and fundamental combinatorial optimization problem. Given a set of elements and a collection of subsets, we aim to select several subsets to cover all the elements. The objective can be minimizing the number of subsets or the total weights. This problem is recognized as NP-hard by Karp [24], and various approximation algorithms have been proposed to find approximation solutions efficiently, such as the standard greedy algorithm, which admits an approximation ratio of $O(\log n)$ [15, 27]. Fruitful studies also consider the Set Cover problem in geometric settings, as they already capture a broad range of real-world applications. We are given a universe of points in R^d, and a set of geometric shapes like disks. The objective is to cover the universe with a minimized number of shapes. The problem is NP-hard even for very simple shapes such as unit disks and squares [7, 20, 23]. However, in the one-dimensional special case, the shapes and the universe become intervals, and the problem can be solved by a simple polynomial-time greedy algorithm even in the weighted version (each candidate interval has an associated weight).

Our paper focuses on the one-dimensional scenario and points out the following perspective. In the set-cover style definition, when we discuss a candidate interval, the only two options are to select or not to select. However, in many cases, we do not need to pay the complete cost to select the entire interval. The question is, can we provide an option to select a part of the interval with a lower price? It is motivated by the following real-world scenario related to Drone Delivery.

Example (Drone Delivery). We aim to deliver a package from location A to B (on a line) by drone. Each drone can serve a given area (an interval on the line). We need to select some drones and arrange for them to deliver the package and minimize the total usage of drones. If we use the Geometric Interval Set Cover problem to model this scenario, we can only consider the selection of the drone but omit the arrangement. However, in some cases, it is enough to use a drone in a small subinterval of its serving area but not the whole interval, with a lower cost. To optimize the arrangement, we need to allow a partial selection of intervals in the theoretical model.

To this end, we formalize the following *Subinterval Cover Problem (SIC)*, which allows algorithms to select an interval partially. Given n candidate intervals $\{1, 2, \ldots, n\}$ (denoted by $[n]$), each is defined within the range $[0, 1)$ as $T_i = [L_i, R_i)$. Each interval can either cover a specific subinterval $[l_i, r_i)$ such that $[l_i, r_i) \subseteq T_i$, or remain unused. The cost of this operation is defined to be $W_i(l_i, r_i)$. The goal is to cover the range $[0, 1)$ while minimizing the total cost. (To ensure feasibility, we further require that $\bigcup_{i=1}^{n} T_i = [0, 1)$.) As a fairly natural starting point, we consider a linear cost function of each candidate interval. That is, we assume that each interval T_i has a nonnegative weight w_i, and the cost of an interval is calculated based on the length of the used subinterval $[l_i, r_i)$, i.e., $W_i(l_i, r_i) = w_i \cdot (r_i - l_i)$. The classic idea for solving interval set cover fails to solve the SIC problem, since they have not considered partial selection. Then, the main question we should ask is, can we also solve SIC in polynomial time?

1.1 Our results

We first demonstrate a natural Integral Linear Programming (ILP) and its LP relaxation. The LP rounding method is a standard attempt to solve the problem. However, we show an unbounded integrality gap under our natural ILP formulation, even in the case of only three distinct weights in Section 3, which highlights the challenge of studying this problem.

Note that if every candidate interval has identical weights, the problem becomes trivial, and the optimal cost is always w_i. Therefore, to understand the challenge of the problem, we start with a minimal non-trivial special case of two weights, where only two distinct weights exist. Is it already tough to solve? We answer this question on the positive end with a polynomial-time dynamic programming algorithm.

▶ **Theorem 1.** *There is an $O(n^7)$ algorithm for the SIC problem with 2 distinct weights.*

However, we find that our approach is hard to generalize even if the number of distinct weights only grows to three, unless we further introduce some assumptions. Our second positive result is a polynomial-time DP algorithm when there is a constant number of distinct cost factors and the interval candidates form a laminar family.

▶ **Theorem 2.** *There is an $O(kn^{2k+3})$ algorithm for the SIC problem with k distinct weights, under the constraint that the intervals form a laminar set family.*

Without the laminar assumption, we still do not know whether the problem is NP-hard or polynomial solvable, even assuming the number of distinct weights is only three. A remarkable negative signal is that the integrality gap becomes infinity when the number of distinct weights becomes three. We leave it as an open question.

Furthermore, we extend our algorithms to get some approximation results in the general setting. The first is as follows.

▶ **Theorem 3.** *There is a $\min\left(\sqrt{w_{\max}/w_{\min}}, w_{\max}/w_{2^{nd}\min}\right)$-approximation algorithm for the general SIC problem with time complexity $O(n^7)$.*

In addition, our approach to the subinterval cover problem can be extended to tackle the collaborative drone delivery problem. This problem, first introduced by Erlebach et al. [18], conceptualizes each candidate interval as a drone with a limited (and connected) travel range and a specified speed. The objective is to deliver a package from the starting point to the ending point with the minimum total time, with each drone being utilized once within its moving range. The problem aligns with our model if we restrict the problem to a path, except that each drone is associated with an initial position in its serving range (aligned with the candidate interval in our model). The difference is that the cost of scheduling a drone also depends on the moving distance starting from the initial position but not the selected subinterval. The only known solution for this scenario is an $O(n)$-approximation algorithm, which remains the best-known result in general graph settings. In Appendix A, we show that our solutions can be applied effectively in scenarios where each drone (or robot) has a predefined initial placement on the path, providing a 2-approximation reduction for these cases. By tolerating an extra approximation ratio of 2, any algorithmic results for the SIC problem can be applied to the drone delivery problem on paths.

1.2 More related work

The Subinterval Cover Problem (SIC) has a strong connection to the classical Set Cover Problem, particularly in scenarios involving predetermined ranges [14] or routes [1, 16] in variants such as the Geometric Set Cover Problem. Various cases of the geometric set cover problem have been studied in the literature, including both unweighted settings, where the goal is to minimize the number of selected subsets [8, 15, 17, 19, 24, 27], and weighted settings, where the objective is to minimize the total weight [2, 3, 13, 21, 26]. For the unweighted geometric set cover problem in the 1D case, both the geometric set cover and hitting set problems (focus on the dual perspective) can be solved in polynomial time using a simple greedy algorithm. However, in higher dimensions, these problems are known to be NP-complete [24], even for simple shapes, such as when the subset collection is induced by unit disks or squares. Aside from the known near-optimal solutions for the general set cover problem, such as the greedy algorithm [15, 27], which iteratively selects the set that covers the largest number of uncovered elements until the entire universe is covered and achieves an approximation ratio of $O(\log n)$, better approximations can often be achieved in low dimensional geometric settings using linear programming (LP)-based approaches [8, 19]. The weighted version of the geometric set cover problem, where each object is assigned a weight, has also received considerable attention. In this variant, the goal is to select a subset of objects with the minimum total weight that covers the entire range. However, only a few instances are known to offer an approximation guarantee better than $O(\log n)$. For example, selecting unit disks in the plane admits a $O(1)$-approximation algorithm using dynamic programming [2] and the LP approach [26]. However, the 1D weighted geometric set cover problem, also known as a special case of weighted interval cover problem, still can be solved in polynomial time [3].

Our SIC problem extends the 1D weighted geometric set cover by incorporating the concept of subinterval selection and accounting for the associated subinterval costs, which has practical applications, such as in the collaborative delivery problem. The collaborative delivery problem, which uses multiple agents, such as drones, to deliver a package with predesigned route, has gained significant attention in recent years. Initial research on collaborative delivery under energy constraints was pioneered by Chalopin et al. [11]. They considered a scenario where a set of energy-constrained drones such as battery budget, positioned on a straight line, must collectively transport an item from a source point to a target point. A polynomial-time algorithm was provided for determining whether drones can successfully deliver an item along a line when they have a uniform budget. However, the problem becomes (weakly) NP-complete when the drones have different budgets [11]. They also provided a quasi-pseudo-polynomial time algorithm [11], along with resource-augmented algorithms [10]. Subsequent studies have built upon this foundation, exploring various aspects of collaborative delivery under energy constraints [6, 12].

In addition to energy-constrained collaboration, research has also explored heterogeneous collaboration, where drones with diverse energy consumption rates [4] and speeds [5]. Delivering one package with minimum total time and minimum energy consumption can both be solved in polynomial time [4, 5, 9]. However, delivering two packages has been proven to be NP-hard [9]. In this scenario, each drone's unlimited budget implies an unlimited movement range, with the only difference being their efficiency ratios. Additionally, 'natural' flying constraints have been investigated, where drones have designated serving ranges, necessitating collaboration due to geographical, regulatory, or other practical constraints [18, 25]. Erlebach et al. [18] demonstrated that with drones having designated serving ranges, and varying speeds, delivering a package from a source to a destination with minimum completion time is a (weakly) NP-hard problem, even when the drones move along a line [18]. We observe that a 2-approximation algorithm for this problem can be provided by applying the solution for the SIC problem.

2 Preliminary

In this preliminary section, we introduce some basic structural properties for the optimal solution of the problem.

▶ **Definition 4** (Solution). *A solution is defined as a tuple (I, l, r): $I \subseteq [n]$ is the set of used intervals, and $[l_i, r_i) \subseteq T_i$ is the subinterval chosen for interval $i \in I$.*

▶ **Definition 5** (Disjoint). *A solution is considered as disjoint if each point $x \in [0, 1)$ is covered by exactly one selected subinterval.*

▶ **Lemma 6.** *A disjointed optimal solution exists for the SIC problem.*

Proof. Assume there is an optimal solution $\mathsf{OPT} = (I, l, r)$ where the selected subintervals of intervals $i, j \in I$ intersect. In this case, we can adjust the subintervals while maintaining the feasibility of the solution. Without loss of generality, let us assume $l_i \le l_j$. We then consider two cases:

- If $l_i \le l_j \le r_i \le r_j$: adjust l_j to start at r_i by setting $l_j \leftarrow r_i$;
- If $l_i \le l_j < r_j \le r_i$: remove interval j from the solution set.

The cost is non-increasing after either subinterval adjustment because the interval j always involves a shorter length. Since there is only a finite number of interval pairs, we can eliminate all intersections, ensuring that each point is covered by exactly one selected subinterval. This results in a disjoint solution. ◀

▶ **Definition 7** (Discrete). *A solution is considered as discrete if, for every $i \in I$, the selected subinterval $[l_i, r_i)$ satisfies the condition that both l_i and r_i belong to $\{L_1, \dots, L_n, R_1, \dots, R_n\}$.*

▶ **Lemma 8.** *There exists a discrete and disjoint optimal solution for the SIC problem.*

Proof. By Lemma 6, a disjoint optimal solution exists. Suppose $\mathsf{OPT} = (I, l, r)$ is such a solution for the SIC problem. If all its non-empty subintervals of OPT satisfy the condition that every l_i and r_i are in the set $A = \{L_1, \dots, L_n, R_1, \dots, R_n\}$, then it is also a discrete solution. However, if some non-empty subintervals, for example, have r_i not aligned with any points in A, and given that all subintervals cover the entire range, there must be another non-empty selected subinterval of interval j adjacent to the selected subinterval $[l_i, r_i)$, such that $l_j = r_i$. Note that since both subintervals are non-empty and their ending points are not in set A, it holds that $L_i < r_i = l_j < R_i$ and $L_j < r_i = l_j < r_j \le R_j$. Adjustments can be made to the subintervals while maintaining the solution's feasibility. Consider two cases:

- If $w_i \le w_j$: Adjust r_i (and correspondingly l_j) to $\min\{r_j, R_i\}$. If $\min\{r_j, R_i\} = r_j$, then remove interval j from the solution set. If $\min\{r_j, R_i\} = R_i$, then the point r_i (and correspondingly l_j) falls within set A. In this case, the number of non-aligned endpoints of the subintervals is reduced by one, and the cost remains non-increasing.
- If $w_i > w_j$: Adjust r_i (and correspondingly l_j) to $\max\{l_i, L_j\}$. If $\max\{l_i, L_j\} = l_i$, then remove interval i from the solution set. If $\max\{l_i, L_j\} = L_j$, then the point r_i (and correspondingly l_j) falls within set A. In this case, the number of non-aligned endpoints of the subintervals is reduced by one, and the cost remains non-increasing.

▶ **Corollary 9.** *Without loss of generality, We can assume a restriction in the problem that the selection of every l_i, r_i is discrete and disjoint.*

Given that there is only a finite number of non-aligned l and r values, and since each adjustment reduces this number by at least one, we can achieve an aligned *discrete* optimal solution OPT' through a finite number of steps. ◀

▶ **Lemma 10.** *The solution OPT of the SIC problem is equivalent before and after merging any two intersecting least weight intervals.*

Proof. Consider two least weight intervals T_i and T_j with weights $w_i = w_j = w_{\min}$ and $T_i \cap T_j \neq \varnothing$. Suppose these intervals are merged into $T_k = T_i \cup T_j$. Let $\mathsf{OPT}, \mathsf{OPT}'$ denote the optimal solution before and after merging, respectively.

- $\mathsf{OPT} \leq \mathsf{OPT}'$: If k is used in an OPT', by the fact that $T_k = T_i \cup T_j$ we can construct corresponding disjoint $[l_i, r_i), [l_j, r_j)$ such that their union is $[l_k, r_k)$.
- $\mathsf{OPT}' \geq \mathsf{OPT}$: If i, j are used in two non-adjacent intervals, from the adjustment arguments in the proof of Lemma 8, we could always expand them so that $[l_i, r_i) \cup [l_j, r_j)$ is an interval while keeping the optimality.

◀

▶ **Corollary 11.** *Without loss of generality, we can assume that the least weight intervals in any instance are disjoint and* $\forall i \in I, w_i = w_{\min}$*, in* OPT*, the selected subinterval* $[l_i, r_i)$ *is equal to the full interval* $[L_i, R_i)$*.*

3 ILP and LP formalization

To address the subinterval cover problem, we can leverage Integer Linear Programming (ILP) and its Linear Programming (LP) relaxation for modeling and solving this combinatorial optimization problem. The objective is to minimize the total cost of selected subintervals while ensuring complete coverage of the universe interval.

We begin by defining the decision variables for our ILP model. Let $x_{i,l,r}$ be a binary variable indicating that $x_{i,l,r} = 1$ if the interval i is used to cover the interval $[l, r)$, and $x_{i,l,r} = 0$ otherwise. These variables enable us to succinctly formulate the problem requirements: to efficiently cover the entire interval using selected subintervals from the intervals, while minimizing the total cost of the subintervals used. As w_i denotes the unit cost of the interval i, and $(r - l)$ represents the length of the subinterval selected within the interval i, the cost is calculated as $(r - l)w_i x_{i,l,r}$. Although the original model allows for the free selection of the subinterval $[l, r)$ from the intervals, with potentially infinite choices, Lemma 8 asserts that the number of feasible choices for a subinterval $[l, r)$ is $O(n^2)$. Specifically, define

$$A_i = \{[l, r) \mid L_i \leq l \leq r \leq R_i, l, r \in \{L_1, \ldots, L_n, R_1, \ldots, R_n\}\},$$

as the sets of feasible choices for $[l, r)$ with respect to interval i. We can then formulate an ILP as follows.

$$\begin{aligned}
&\text{minimize} && \sum_{i \in [n]} \sum_{[l,r) \in A_i} (r - l) w_i x_{i,l,r} && && (1)\\
&\text{subject to} && \sum_{[l,r) \in A_i} x_{i,l,r} \leq 1 && \forall i \in [n] && (2)\\
& && \sum_{i \in [n], [l,r) \in A_i} x_{i,l,r} \geq 1 && && (3)\\
& && x_{i,l,r} \in \{0, 1\} && \forall i \in [n], [l, r) \in A_i && (4)
\end{aligned}$$

Constraints (2) prevent using interval i in two non-continuous subintervals. Constraints (3) ensure that every point within the interval is covered by some subintervals. We can relax the integrality Constraints (4) from $x_{i,l,r} \in \{0, 1\}$ to $x_{i,l,r} \in [0, 1]$ to have the relaxed LP formulation.

LP rounding is usually an effective way to design an approximation algorithm. However, we demonstrate in Section 3.1 with a simple example that there is an unbounded integrality gap for this natural LP formulation.

Before delving into the details, let us first provide an intuitive understanding of the difference between LP and ILP formulations: LP allows for the allocation of a total quantity of no more than 1 across multiple, say k ($k \geq 1$), non-continuous subintervals within an interval T_i, while in ILP, k is 1. Figure 2 gives an example of the case where $k = 2$.

3.1 Existence of the arbitrarily large LP integrality gap

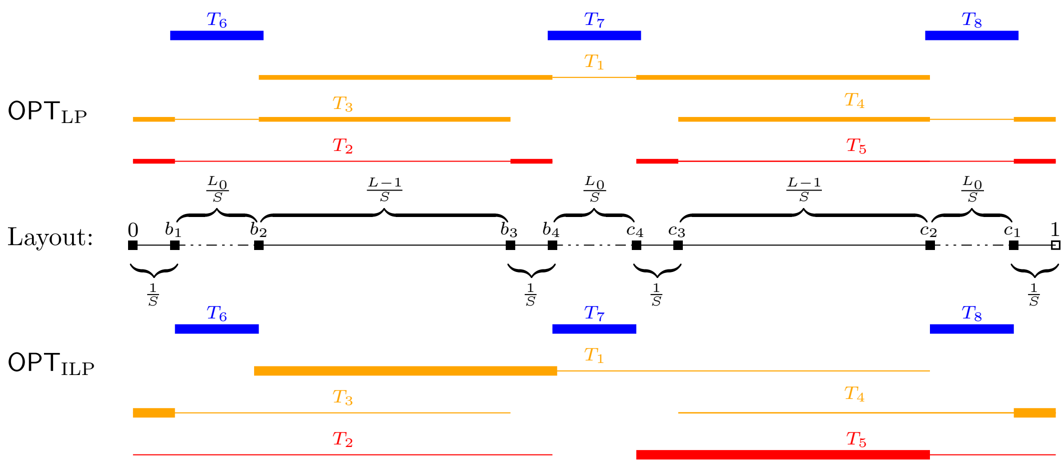

Figure 1 $\mathsf{OPT}_{\mathrm{LP}}$ and $\mathsf{OPT}_{\mathrm{ILP}}$ for the same instance of 8 intervals with 3 distinct weights (marked as 0-Blue, $w_{\min}$-Orange, $w_{\max}$-Red). The thickest, second-thickest, and thinnest line segments represent the allocation of 1, $\frac{1}{2}$ (i.e., $T_{1\ldots5}$ in $\mathsf{OPT}_{\mathrm{LP}}$) and 0, respectively.

For any maximum weight $w_{\max}$ and minimum non-zero weight $w_{\min} < w_{\max}$, we can construct an instance in Figure 2, with 5 non-zero (weight) intervals $T_{1\ldots5}$ and 3 zero (weight) intervals $T_{6\ldots8}$, to demonstrate that the integrality gap is $\Omega(w_{\max}/w_{\min})$. Consequently, the gap can be arbitrarily large for appropriate values of $w_{\max}$ and $w_{\min}$.

Let $a = w_{\min}$, $c = w_{\max}$. Define $L = \left(\frac{c}{a}\right)^2$ and $L_0 = 1 + \frac{c(2L+2)}{a}$. The endpoints $\{0, b_1, b_2, b_3, b_4, c_4, c_3, c_2, c_1, 1\}$ are arranged sequentially from left to right. Before normalization, the total length is $S = 2(1 + L_0 + L) + L_0$, and the normalization factor is $\frac{1}{S}$. The relationships between points are defined as follows: $b_1 - 0 = 1 - c_1 = \frac{1}{S}$, $b_2 - b_1 = c_1 - c_2 = \frac{L_0}{S}$, $b_3 - b_2 = c_2 - c_3 = \frac{L-1}{S}$, $b_4 - b_3 = c_3 - c_4 = \frac{1}{S}$.

The non-zero intervals are placed as $T_1 = [b_2, c_2)$, $T_2 = [0, b_4)$, $T_3 = [0, b_3)$, $T_4 = [c_3, 1)$, $T_5 = [c_4, 1)$ and their weights are assigned as $w_{1,3,4} = a$, $w_{2,5} = c$. The corresponding ranges of the zero intervals $T_6 = [b_1, b_2)$, $T_7 = [b_4, c_4)$, $T_8 = [c_2, c_1)$, are called zero ranges and are represented by dashed segments. Each of $T_{1\ldots5}$ is separated by a zero range, and the structure is symmetric around the center.

The length L_0 guarantees that, in any optimal solution, the subintervals $[l_1, r_1), \ldots, [l_5, r_5)$ do not cross any zero range: Since $a \cdot L_0 > c(2L + 2)$, if a subinterval with the minimum weight crosses a zero range, the cost will be higher than in the case where zero ranges are covered by $T_{6\ldots8}$ and all the rest (with total length $2L+2$) are covered by c-weight intervals. That is, in any optimal solution, zero ranges are covered by zero intervals.

To obtain $\mathsf{OPT}_{\mathrm{LP}}$, the strategy involves minimizing the use of T_2 and T_5 while maximizing the use of T_1, T_3, and T_4. The upper part of Figure 2 shows one possible case.

$$\mathsf{OPT}_{\mathrm{LP}} = \frac{1}{S} \cdot 2 \cdot \frac{1}{2} \left(1 \cdot (a+c) + (La + (L-1)\,a + 1 \cdot c)\right) + 0 \cdot 3 \cdot \frac{L_0}{S} = \frac{2La+2c}{S}.$$

For the ILP optimal solution, Each of $T_{1...5}$ can only be used on at most one side of the zero range that separates it. Note $L > 1 > \frac{1}{S}$, so the range $[0, b_1)$ will be covered by T_3 but not T_2, and the range $[c_1, 1)$ will be covered by T_4 but not T_5, as shown in the lower part of Figure 2,

$$\mathsf{OPT}_{\mathrm{ILP}} = \frac{1}{S}(La + Lc + 2 \cdot a + 0 \cdot 3L_0) = \frac{L(a+c)+2a}{S}.$$

In this instance, the integrality gap

$$\frac{\mathsf{OPT}_{\mathrm{ILP}}}{\mathsf{OPT}_{\mathrm{LP}}} = \frac{L(a+c)+2a}{2La+2c} > \frac{Lc}{2La+2c} = \frac{c}{2a+\frac{2c}{L}} = \frac{c}{a\left(2+2\left(\frac{a}{c}\right)\right)} \geq \frac{1}{4} \cdot \frac{c}{a} = \Omega(w_{\max}/w_{\min}).$$

4 Algorithm under binary weights

In this section, we introduce a dynamic programming algorithm designed to solve the subinterval cover problem, which involves two distinct weights.

The core of any dynamic programming strategy lies in the definition of well-structured subproblems. We define subproblems to address the optimal solution to serve a specific range by a specific subset of intervals. Roughly speaking, the specific subset of intervals can be the first i-th intervals, and the specific range can be $[0, 0.5)$. Now consider the case we are solving i. We can first decide how to use i by enumerating the subinterval of i, say $[l_i, r_i) \subseteq [L_i, R_i)$. For example, it can be $[0.1, 0.2)$. The problem is, how do we cover $[0, 0.1)$ and $[0, 2, 0.5)$ next? In general, we need to enumerate the subset of intervals, which serves $[0, 0.1)$, and which serves $[0, 2, 0.5)$, which takes exponential time. To circumvent this problem, we need a clever DP order and a fine-grained subproblem definition, which reduces the size of the subproblem back to polynomials while keeping the correctness. First, we introduce the DP order.

▶ **Lemma 12** (DP order). *There is a total order $\preceq$ on $[n]$ satisfying the following left-to-right rule and inside-to-outside rule.*

- *Left to right: $u \preceq v$ if $L_u < L_v$ and $R_u < R_v$;*
- *Inside to outside: $u \preceq v$ if $L_v \leq L_u$ and $R_u \leq R_v$.*

Proof. Without loss of generality, we assume that there are no two identical intervals for convenience, as the reflexivity is already given by inside-to-outside-type for identical intervals.

Construct a graph $G = (V, E)$ where $V = [n]$ and

$$E = \{(u, v) \mid u \neq v \wedge ((L_u \geq L_v \wedge R_u \leq R_v) \vee (L_u < L_v \wedge R_u < R_v))\}.$$

We are to prove that this graph is an acyclic tournament graph, which admits a feasible total order. A tournament graph is a graph with exactly one directed edge between each pair of distinct vertices, and the absence of 3-cycles in a tournament graph G ensures that G is acyclic [22].

For any pair of distinct vertices u, v, the condition $(L_u \geq L_v \wedge R_u \leq R_v)$ and $(L_u < L_v \wedge R_u < R_v))$ are mutually exclusive, hence there is exactly one edge between u, v, and by definition, G is a tournament graph. We denote each edge as either inside-to-outside-type (e_1) for the condition $(L_u \geq L_v \wedge R_u \leq R_v)$ or left-to-right-type (e_2) for $(L_u < L_v \wedge R_u < R_v)$. Due to the transitivity of e_1 and e_2, respectively, a 3-cycle cannot consist of edges of the same type. For the two remaining cases:

- $u \xrightarrow{e_1} v \xrightarrow{e_1} w \xrightarrow{e_2} u$, from the two e_1 edges, we infer $R_w \geq R_u \geq R_u$, which contradicts the e_2 edge condition that $R_w < R_u$.
- $u \xrightarrow{e_1} v \xrightarrow{e_2} w \xrightarrow{e_2} u$, from the two e_2 edges, we deduce $L_v < L_w < L_u$, which contradicts the e_1 edge condition that $L_u \geq L_v$.

Therefore, there are no 3-cycles in G, and there exists a valid total order of $[n]$. ◀

Then, we are ready to define the subproblem in our DP algorithm. In general, the subproblem describes the optimal solution for using a subset of candidate intervals to cover a specific range. The trivial way is to define it for every subset, which is exponential. However, we restrict the subset to a family corresponding to the defined DP order, which can be captured by two parameters: 1) i: it should be in the first i-th candidate interval following the DP order. 2) $[C_l, C_r)$, its left endpoints should falls in the interval. Combining with the discrete property in Corollary 9, this restricted family of subsets makes the number of subproblems polynomial.

▶ **Definition 13.** *Let $f_i(S, C)$ denote the minimum cost to cover the range $S = [S_l, S_r)$ using the first i intervals in the ordered sequence of intervals, where only intervals with left endpoints falling within $C = [C_l, C_r)$, that is, $C_l \leq L_i < C_r$. By Lemma 8, without loss of generality, we can assume that all C_l, C_r, S_l, S_r are the endpoints of the intervals.*

Algorithm 1 Dynamic programming under binary weights restriction.

```
1: Sort and renumber all intervals according to Lemma 12 such that $1 \preceq 2 \preceq \cdots \preceq n$.
2: Basic case:
   $f_0(S,C) = \begin{cases} 0, & \text{if } S = \varnothing \\ \infty, & \text{otherwise} \end{cases}$
3: For $i \geq 1$, recursively define $f_i(S,C)$ as follows:
   $f_i(S,C) = \min \begin{cases} f_{i-1}(S,C), \\ f_{i-1}([S_l,u),[C_l,u)) + w_i(v-u) + f_{i-1}([v,S_r),[u,C_r)), \\ \qquad \forall u \in [C_l,C_r), [u,v) \subseteq S \cap [L_i,R_i) \end{cases}$
```

The next question is why we only need to focus on the special family, not every subset. The reason is the existence of a good monotonicity behind the optimal solution, corresponding to the left endpoints. Recall that L_i is the left endpoint of i, and l_i is the left endpoint of the selected subinterval for i. The monotonicity says for every $j \preceq i$, we will use j to cover some range on the left side of l_i, if and only if $L_j < l_i$. We conclude the lemma as follows. We remark that it is the most important property we observe in the two-weight special case. However, this property is hard to generalize even to the three-weight case.

▶ **Lemma 14** (Monotonicity of subinterval left endpoints). *For every subproblem $f_i(S, C)$, there exists an optimal solution* OPT $= (I, l, r)$ *such that for every $i, j \in I$, $j \preceq i$,*

- $L_j < l_i \iff l_j < l_i$.

Proof. Since $L_j \leq l_j$ by definition, $l_j < l_i \Rightarrow L_j < l_i$ always holds. And we only need to prove that $L_j < l_i \Rightarrow l_j < l_i$.

For convenience, we remap the weights to 0 and 1, representing the lower and higher weights, respectively. Let $\mathsf{OPT} = (I, l, r)$ be the optimal solution of the subproblem, by the assumption in Corollary 9, it should be discrete and disjoint. Define $T_j^1 = (T_j \setminus T_i)$ and $T_j^2 = (T_j \cap T_i)$. Given the specified order, $j \preceq i$ implies $R_j \leq R_i$. When $l_j \in T_j^1$, the lemma is upheld. Therefore, we exclusively consider T_j^2, which belongs to T_i.

Corollary 11 necessitates that $w_i = w_j = 1$. With $w_i = 0$, since $[l_j, r_j) \subseteq T_i$, we obtain $[l_i, r_i) \neq [L_i, R_i)$. This contradicts Corollary 11. In the case of $w_j = 0$, $[l_j, r_j) = [L_j, R_j)$. If the implication is false, i.e., $L_j < l_i$ but $l_j \geq l_i$, we also get a contradiction because $L_j = l_j$. The remaining scenario considers $w_i = w_j = 1$. Under this circumstance, when $l_j \geq l_i$, $l_j \geq r_i$ follows from the disjointness, then $[l_i, r_i) \subseteq T_j^2$. Since $T_j^2 \subseteq T_i$, swapping subintervals $[l_i, r_i)$ and $[l_j, r_j)$ is valid, facilitating an optimal solution that conforms to the lemma. The lemma then holds for every i by induction.

◀

▶ **Lemma 15.** *Algorithm 1 correctly solves every subproblem $f_i(S, C)$.*

Proof. The proof follows induction and Lemma 14. Assuming we have an optimal solution for every f_{i-1} subproblem, the algorithm considers every option that satisfies the monotonicity and takes the optimal one based on the optimal solutions for the subproblems. Consider the optimal solution of $f_i(S, C)$, assuming i is not selected, straightforwardly, it equals to $f_{i-1}(S, C)$. Assuming i is selected at $[l_i, r_i)$, we need to further know who should take care $[S_l, l_i)$ and $[r_i, S_r)$. By the nice monotonicity in Lemma 14, we know that the subset that serves the range $[S_l, l_i)$ must have the property $L_j < l_i$, and the subset that serves the range $[r_i, S_r)$ must have the property $L_j \geq l_i$. Therefore, if we know $[l_i, r_i)$, the optimal solution of $f_i(S, C)$ can be recovered by $f_{i-1}([S_l, l_i), [C_l, l_i))$ and $f_{i-1}([l_i, r_i), [l_i, C_r))$. Finally, by enumerating $[l_i, r_i)$, the DP approach can correctly solve $f_i(S, C)$. ◀

Proof of Theorem 1. By Lemma 15, Algorithm 1 gives an optimal solution for the binary weights subinterval cover problem by $f_n([0,1), [0,1))$. The time complexity $O(n^7)$ is determined by the number of states, which is $O(n^5)$, and the number of transitions for each state, which is $O(n^2)$. ◀

As a reduction to Algorithm 1, we have a $\min\left(\sqrt{w_{\max}/w_{\min}}, w_{\max}/w_{2^{\text{nd}}\min}\right)$-approximation algorithm mentioned in Theorem 3.

Proof of Theorem 3. Let OPT be the optimal solution for the original problem. We now modify the weights of the original problem by setting weights $w_i \leq \theta$ to $w_i' = \theta$, where $\theta = \sqrt{w_{\max}/w_{\min}}$, and weights $w_i > \theta$ to $w_i' = w_{\max}$. Let OPT' be the optimal solution for the modified problem. Every weight is scaled up to $\sqrt{w_{\max}/w_{\min}}$ times, so we have $\frac{\mathsf{OPT}'}{\mathsf{OPT}} \leq \sqrt{w_{\max}/w_{\min}}$. The modification leads to a binary-weight condition, applying Algorithm 1, we get OPT'.

Next, we apply the solution OPT' to the original problem by reverting the weights back to their original values, resulting in a solution ALG for the original problem. It is evident that $\mathsf{ALG} \leq \mathsf{OPT}'$, as the weights in the original problem are no greater than those in the modified problem.

Combining the above, $\frac{\mathsf{ALG}}{\mathsf{OPT}} \leq \frac{\mathsf{OPT}'}{\mathsf{OPT}} \leq \sqrt{w_{\max}/w_{\min}}$.

If $w_{\min} = 0$, we can only set $\theta = 0$ to avoid the $+\infty$ approximation ratio(when OPT only consists of 0-weight intervals). By similar procedure, the approximation ratio is $w_{\max}/w_{2^{\text{nd}}\min}$.

◀

5 Algorithm under laminar set family

In this section, we consider the case where all intervals form a laminar set family: each pair of intervals is either disjoint or one interval contains the other. All intervals form a tree structure, where we let i be j's parent if $[L_j, R_j) \subseteq [L_i, R_i)$. Remark that if two intervals are the same, we can select the parent arbitrarily. As a result, we have an auxiliary root of range $[0, 1)$, and the root's children are intervals that form a partition of the whole range due to feasibility; each interval may contain some non-intersecting intervals as children, *sorted from left to right*. We use terminology related to tree structures (e.g., descendant/ancestor, parent/child, root/leaf, subtree) to describe the relationships within the laminar family in the following discussion. Since we are presenting an XP algorithm towards the number of weights k, we require $w_i \in \{\alpha_1, \alpha_2, \dots, \alpha_k\}$ where $\alpha_1 < \alpha_2 < \cdots < \alpha_k$ (except $w_0 = +\infty$ since it's auxiliary).

The key difference between the laminar set family and the general case is the independence provided by the laminar (tree) structure, stated as Lemma 16. This structure allows us to utilize the hierarchical order of the tree to ensure that each interval is used only once while still having a good representation of the solution.

▶ **Lemma 16** (Independence of laminar structure)**.** *Consider a fixed interval i. Suppose we select $[l_i, r_i)$ from $[L_i, R_i)$, then the parts $[L_i, l_i)$ and $[r_i, R_i)$ must be solved by its descendants or ancestors if they are non-empty.*

Proof. Only ancestors and descendants of i may intersect with $[L_i, l_i), [r_i, R_i)$, so for a correct solution, the lemma is true by definition of problem. ◀

We now begin to describe our approach to obtain the optimal solution. Recall that we assume the disjointness and discreteness properties by Corollary 9. We also assume a *non-redundant* property defined as follows.

▶ **Definition 17** (Redundant)**.** *Under the laminar set family constraint, a solution is considered redundant if there exist intervals $i, j \in I$, such that:*

- *j is an ancestor of i: $L_j \le L_i \le R_i \le R_j$;*
- *j covers a range inside $[L_i, R_i)$, i.e., $[l_j, r_j) \subseteq [L_i, R_i)$.*
- *$l_j = r_i$ or $l_i = r_j$.*

If such a pair does not exist, the solution is non-redundant.

▶ **Lemma 18.** *Under the laminar set family constraint, there is a discrete, disjoint, and non-redundant optimal solution (I, l, r).*

Proof. Starting from a discrete and disjoint optimal solution, we can adjust it to a non-redundant solution as described in Lemma 8. If the optimal solution (I, l, r) is redundant with i, j as the interval pairs, the condition $w_i = w_j$ must hold or we can achieve a better solution:

- If $w_i < w_j$, assign $l_i \leftarrow \min(l_i, l_j)$ and $r_i \leftarrow \max(r_i, r_j)$, then remove j from I;
- If $w_j < w_i$, assign $l_j \leftarrow \min(l_i, l_j)$ and $r_j \leftarrow \max(r_i, r_j)$, then remove i from I;

Therefore, we can always safely remove one interval and allow the other to expand to maintain the optimality while decreasing the number of redundant pairs by at least one. Thus, a non-redundant solution can be obtained in finite steps. ◀

▶ **Corollary 19.** *We can assume that we restrict the solution to be disjoint, discrete, and non-redundant without loss of generality.*

Since the laminar family assumption forms a tree structure for candidate intervals, we aim to design a bottom-up Dynamic Programming algorithm to solve the problem. The subproblem is naturally defined on every subtree. Consider a subtree rooted at interval i, and a range $[S_l, S_r)$. What is the best choice to cover $[S_l, S_r)$ by using the intervals in the subtree? On the one hand, considering the cost, we want to use cheaper intervals to cover some part of $[S_l, S_r)$, such as those intervals with a price of 0. However, consider the following case. There is a candidate interval of $[0.5, 0.6)$ with price 0 and another interval of $[0, 1)$ with price 1. When we solve subproblems bottom-up, how do we consider using $[0.5, 0.6)$ to serve the entire range $[0, 1)$? If we select $[0.5, 0.6)$ to cover $[0.5, 0.6)$, we need to use its ancestors to cover both $[0, 0.5)$ and $[0.6, 1)$, which is impossible in this example. Therefore, two factors should be considered: the cost and the future pressure. Generally speaking, we want to formalize a subproblem that captures the minimized cost under every possible future pressure. Consider an arbitrary subinterval selection inside the subtree. After we use these selected subintervals to cover $[S_l, S_r)$, there are several separate and uncovered subranges left, which need to be covered by i's ancestors. Assuming we know each of their future cover price, we will know the total covering cost. However, different cover costs on different separate and uncovered intervals will lead to different future pressures. Fortunately, because of the tree structure, the future pressure only depends on the number of subranges on each price, which can be captured by the vector $\mathbf{p} = (p_1, \ldots, p_k)$, which is polynomial size, where each p_x means we still have p_x uncovered subrange in S, which are reserved by an ancestor of i (whose weight is α_x) to solve. Following this intuition, we define the following subproblems.

▶ **Definition 20** (Subproblems, refer to Figure 2 for a toy example). *Let $f_i(S, \mathbf{p})$ be the minimum cost of solving the following subproblem with a disjoint, discrete, and non-redundant solution. Cover the range $S = [S_l, S_r)$ using i and its descendants, with $\mathbf{p} = (p_1, \ldots, p_k)$ representing the future pressure, i.e., the pending intervals for i's ancestors whose costs are already charged. Each p_x represents ancestors with weight α_x who must cover a subrange inside $[L_i, R_i)$. In addition, we only define f_i when $S \subseteq [L_i, R_i)$, otherwise $f_i(S, \mathbf{p}) = \infty$.*

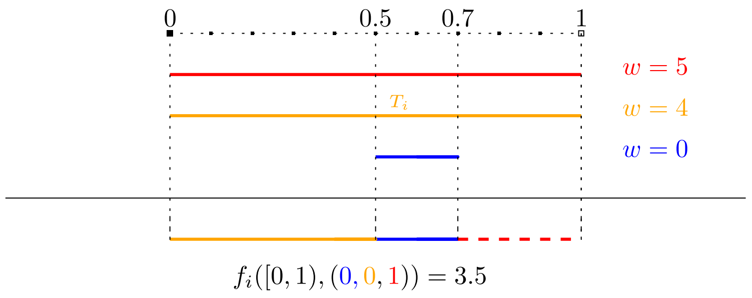

Figure 2 An example of $f_i(S, \mathbf{p})$, where $S = [0, 1)$, $\mathbf{p} = (0, 0, 1)$ and the red dashed segment corresponds to the third element of $\mathbf{p}$. It means that we use i (the orange interval) and i's descendants (the blue interval) to cover the range $[0, 1)$, with a future pressure $(0, 0, 1)$, which means we still have an uncovered subrange $[0.7, 1)$ (the dashed red one), needing to be covered by i's ancestor with weight 2. The reason why $f_i([0, 1), (0, 0, 1)) = 3.5$ is: we pay $4 \cdot 0.5$ on the orange subrange; we pay $0 \cdot 0.2$ on the blue subrange, and we plan to pay $5 \cdot 0.3$ on the dashed red subrange.

Next, we introduce our DP algorithm to calculate every $f_i(S, \mathbf{p})$. First, we use a binary tree special case to illustrate our high-level idea. Consider we aim to calculate $f_i(S, \mathbf{p})$, where it has two children u and v. We have two steps: the children-merging step and the children-parent step. In the children-merging step, we aim to merge the subproblems of the two children f_u and f_v into g_i, where g_i is an intermediate step to calculate f_i. Specifically, $g_i(S, \mathbf{p})$ means the minimized cost to solve S by all i's children, under the future pressure $\mathbf{p}$ for i and i's ancestors. In the children merging step, for an arbitrary target $g_i(S, p)$, the optimal solution will partition S into several segments. Let S^u_l be the left-most point that the optimal solution uses an interval in the subtree of u to serve, and S^u_r be the right-most point; we call $S^u = [S^u_l, S^u_r)$ the serving area of u. Similarly, we define $S^v = [S^v_l, S^v_r)$ to be the serving area of v. The remaining part of S is promised to be solved by i or i's ancestors, which are called ancestor areas. For example, the ancestor area may consist of three segments: $S^1 = [S_l, S^u_l)$, $S^2 = [S^u_r, S^v_l)$, and $S^3 = [S^v_r, S_r)$. Remark that S_u and S_v can be empty, and the ancestor area may not be exactly the three segments. For instance, if $S^u = \emptyset$, the first ancestor area becomes $[S_l, S^v_l)$. The next task is to assign $\mathbf{p}$ to different segments. Assume we know the optimal solution of $g_i(S, \mathbf{p})$; we define $\mathbf{p}_1$ to be the future pressure in S_1, $\mathbf{p}_2$ to be the future pressure in S_2, $\mathbf{p}_3$ to be the future pressure in S_3, $\mathbf{p}_u$ be the future pressure in S^u, and $\mathbf{p}_v$ be the future pressure in S^v. Then, we want to know the cost of the optimal solution using this information. One key fact is $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$ must be three standard basis because of the non-redundant assumption. See the lemma below.

▶ **Lemma 21.** *For any non-redundant feasible solution of $g_i(S, \mathbf{p})$, let $S' \subseteq S$ be an arbitrary segment of the ancestor area. S' must be served by only one interval.*

Proof. Because S' is inside the ancestor area, the intervals serving it must be i or i's ancestors by definition. Assume it is served by more than two intervals, x and y. Without loss of generality, we assume $[l_x, r_x)$ and $[l_y, r_y)$ are adjacent. Because we either have x as y's ancestor or y as x's ancestor, without loss of generality, we assume x is y's ancestor. Then, we know S' is a subset of $[L_i, r_i)$, so $[l_y, r_y)$ is a subset of $[L_x, R_x)$. Combined with the fact that $[l_x, r_x)$ and $[l_y, r_y)$ are adjacent, the solution is redundant, which is a contradiction. ◀

Let β_1, β_2, β_3 be the corresponding weight of the three standard basis $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$, respectively; the cost of the optimal solution should be $\beta_1|S^1| + \beta_2|S^2| + \beta_3|S^3| + f_u(S^u, \mathbf{p}_u) + f_v(S^v, \mathbf{p}_v)$. Although the optimal solution is not known to us, we can enumerate every choice of these parameters and take the minimum cost.

The idea can be straightforwardly generalized to the case i has more than two children, where we should have more parameters, such as another p_x for another child x, and also more segments of the ancestor area. To reduce the running time, we introduce an idea to proceed with the children one by one to calculate a new subproblem $g^j_i(S, \mathbf{p})$, which means the optimal solution to cover S by the first j children of i, under the future pressure of $\mathbf{p}$ to i and i's ancestors. When j becomes d, which is the number of children, g^d_i is equal to g_i. We initialize $g^0_i(S, \mathbf{p})$, which does not consider any children of i. In this case, S is all ancestor area, so we only have a feasible (non-redundant) solution when $\mathbf{p}$ is a standard basis, corresponding to β. The cost is equal to $\beta|S|$. Then, when we proceed with a new child j and aim to solve $g^j_i(S, \mathbf{p})$, we need also to partition S. We need to know $S^j = [S^j_l, S^j_r)$ as the serving area of j in the optimal solution and its corresponding future pressure p_j. Then, we know that the left segment $S^0 = [S_l, S^j_l)$ must be solved by the subtrees of children 1 to $j-1$, under a specific future pressure $\mathbf{p}_0$. The right segment $S^1 = [S^j_r, S_r)$ is an ancestor area; it must be solved by i or i's ancestors, by only one special interval with weight β_1, which can be described by a standard basis $\mathbf{p}_1$. We should have $\mathbf{p}_0 + \mathbf{p}_j + \mathbf{p}_1 = \mathbf{p}$. Again, because we do not know the optimal solution, we need to enumerate every choice of these parameters, and $g^j_i(S, \mathbf{p})$ is equal to the minimum value of $g^{j-1}_i(S^0, \mathbf{p}_0 - \mathbf{p}_j - \mathbf{p}_1) + f_j(S^j, \mathbf{p}_j) + \beta_1|S^1|$.

See Algorithm 2 for a pseudo code of the complete procedure to solve f_i, including more corner cases. In the complete algorithm, we will solve every i in the bottom-up order.

▶ **Lemma 22.** *Every $f_i(S, \mathbf{p})$ is calculated correctly.*

Proof. We prove it by induction. For the base case on leaves, if $\mathbf{p}$ is a standard basis corresponding to weight β, we have $f_i(S, \mathbf{p}) = \beta|S|$, which is trivially correct. If $\mathbf{p}$ is not a standard basis, the solution will become redundant, so we do not have the value, or we can view the value as ∞, which is also correct. Then, for the induction step, we assume every child of i is calculated correctly. $g_i^j(S, \mathbf{p})$ means the optimal solution to serve S by the first j children of i, under the future pressure $\mathbf{p}$. Assume every g_i^j is correct; $g_i = g_i^d$ is correct, where d is the number of i's children. If i is used in the optimal solution, it must be used for a future pressure in g_i, so $f_i(S, \mathbf{p})$ should equal $g_i(S, \mathbf{p} + \mathbf{p}_i)$ where p_i is the standard basis corresponding to w_i; otherwise, if i is not used, $f_i(S, \mathbf{p}) = g_i(S, \mathbf{p})$. The DP algorithm considers the two cases and takes the minimum.

Therefore, the final step is to show g_i^j is correctly calculated again by induction. In the base case, g_i^0 is straightforwardly correct for the same reason as the calculation for leaves. Next, when we introduce a new child j and calculate $g_i^j(S, \mathbf{p})$, we need to consider the partition of the optimal solution. Let S_l^j be the left-most point served by j or j's descendant, and S_r^j be the right-most point. We define $S^j = [S_l^j, S_r^j]$. Some of the segment inside S^j is served by i or i's ancestors, which can be captured by a specific future pressure $\mathbf{p}_j$. The cost inside S^j is exactly $f_j(S^j, \mathbf{p}_j)$. Then, the left part, $S^0 = [S_l, S_l^j)$, is served by children 1 to $j-1$, under a future pressure $\mathbf{p}_0$, whose cost is exactly $g_i^j(S^0, \mathbf{p}_0)$. The right part, $S^1 = [S_r^j, S_r)$, is the ancestor area, which can only be served by one ancestor interval by the non-redundant property, whose cost is $\beta_1|S^1|$. Therefore, the cost of the optimal solution must appear in the enumeration of these parameters so we can solve every g_i^j correctly. ◀

Proof of Theorem 2. By Lemma 22, $f_0([0, 1), \emptyset)$ is calculated correctly, which should be equal to OPT. The only remaining part is the running time. The critical part is the interaction to calculate every g_i^j, which is a total of n times. We need to calculate every S, which needs $O(n^2)$, every S_j, which needs $O(n^2)$, every $\mathbf{p}_0$, which needs $O(n^k)$, $\mathbf{p}_j$, which needs $O(n^k)$, and the standard basis $\mathbf{p}_1$, which needs $O(k)$. Totally, it is $O(kn^{2k+5})$.

Finally, by the fact that used least weight intervals are always maximal (Corollary 11), we can omit the least weight and define $\mathbf{p}$ as a $(k-1)$-dimensional vector. Then, when we calculate i whose weight is the least weight, we only need to discuss whether i is used for the whole range $[L_i, R_i)$, or not used. As a result, the running time of enumerating $\mathbf{p}_0$ and $\mathbf{p}_1$ becomes $O(n^{k-1})$, so that the time complexity can be reduced to $O(kn^{2k+3})$.
◀

6 Conclusion and Future Work

In conclusion, we introduce a clean combinatorial optimization problem: the subinterval cover problem. We find this problem to be inherently challenging, even to approximate. To shed some light for future research, we delve into two specific scenarios: cases with two distinct weights and those with constant weights with laminar families. For these cases, we devise efficient polynomial-time exact algorithms.

However, extending our 2-weight algorithm presents a challenge. The key monotonic structure identified in Lemma 14 does not hold true when confronted with three distinct

Algorithm 2 Dynamic programming under laminar set family.

```
1: procedure SolveLaminar($i$)                                    ▷ The function to calculate $f_i$
2:     Initialize $g_i^0(S,\mathbf{p}) = \begin{cases} 0, & S=\varnothing, \mathbf{p}=\mathbf{0} \\ \beta_x(S_r - S_l), & S \subseteq [L_i, R_i), x \in [k], \mathbf{p} \text{ is a standard basis} \\ +\infty, & \text{otherwise} \end{cases}$
3:     for each child $j$ as the $j$-th child of $i$ do
4:         Initialize $g_i^j(*) = g_i^{j-1}(*)$
5:         for each $S \subseteq [L_i, R_i)$, $\mathbf{p}_0$, $\mathbf{p}_j$, $\mathbf{p}_1$, and $S^j$ where $S^j \subseteq S$ and $S^j \subseteq [L_j, R_j)$ do
6:             ▷ $\mathbf{p}_0$ and $\mathbf{p}_j$ is selected from $\{0 \cdots n\}^k$, $\mathbf{p}_1$ is an arbitrary standard basis.
7:             $S^0 = [S_l, S_l^j)$                                  ▷ The left area for $g_i^{j-1}$
8:             $S^1 = [S_r^j, S_r)$                                  ▷ The right ancestor area
9:             if $S^1 = \emptyset$ then
10:                Update $g_i^j(S, \mathbf{p}_0 + \mathbf{p}_j)$ with $g_i^{j-1}(S^0, \mathbf{p}_0) + f_j(S^j, \mathbf{p}_j)$.
11:                                       ▷ "Update $a$ with $b$" means $a \leftarrow \min\{a, b\}$ for clarity
12:            else
13:                Let $\beta_1$ be the corresponding weight of the standard basis $\mathbf{p}_1$.
14:                Update $g_i^j(S, \mathbf{p}_0 + \mathbf{p}_j + \mathbf{p}_1)$ with $g_i^{j-1}(S^0, \mathbf{p}_0) + f_j(S^j, \mathbf{p}_j) + \beta_1|S^1|$.
15:            end if
16:        end for
17:    end for
18:    Let $d$ be the number of children of $i$.
19:    $g_i(*) = g_i^d(*)$.
20:    if $i$ is the root then
21:        $f_i(*) = g_i(*)$.                ▷ We cannot use the root interval since it is virtual.
22:    else
23:        Let $\mathbf{p}_i$ be the standard basis corresponding to $w_i$.
24:        $f_i(S,\mathbf{p}) = \min\{g_i(S,\mathbf{p}), g_i(S,\mathbf{p}+\mathbf{p}_i)\}$ for every possible $S$ and $\mathbf{p}$
25:    end if
26: end procedure
```

weights, as exemplified in Figure 3. Note that we have tried weaker monotonicity to maintain a polynomial-size DP definition, but all attempts fail when the number of distinct weights increases to three. Intriguingly, the integrality gap becomes unbounded with the introduction of three distinct weights. Consequently, we pose an intriguing open question:

- *Is the problem NP-Hard, even only considering three distinct weights?*

Furthermore, we explore the potential for generalizing the cost function for a wider range of applications. That is, the cost of selecting a subinterval within candidate interval i can be expressed as a function $W_i(l_i, r_i)$. As a simple example, the cost function can be a constant (it becomes the geometric interval set cover problem), and the problem is also solvable in polynomial time [3]. Discovering more well-motivated cost functions continues to be interesting for future studies.

References

1 Junade Ali and Vladimir Dyo. Coverage and mobile sensor placement for vehicles on predetermined routes: A greedy heuristic approach. 2017.

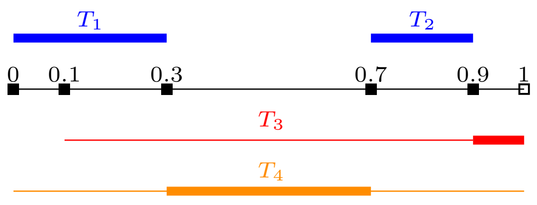

Figure 3 An example where the monotonicity in Lemma 14 fails under three distinct weights. $w_{1,2} = 0 < w_4 = 1 < w_3 = 2$. In this example, $L_3 < l_4$ but $l_4 > l_3$ in the optimal solution.

2 Christoph Ambühl, Thomas Erlebach, Matúš Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 3–14. Springer, 2006.

3 Mikhail J. Atallah, Danny Z. Chen, and DT Lee. An optimal algorithm for shortest paths on weighted interval and circular-arc graphs, with applications. *Algorithmica*, 14 (5):429–441, 1995.

4 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, and Paolo Penna. Energy-efficient delivery by heterogeneous mobile agents. *arXiv preprint arXiv:1610.02361*, 2016.

5 Andreas Bärtschi, Daniel Graf, and Matús Mihalák. Collective fast delivery by energy-efficient agents. *arXiv preprint arXiv:1809.00077*, 2018.

6 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Barbara Geissmann, Daniel Graf, Arnaud Labourel, and Matúš Mihalák. Collaborative delivery with energy-constrained mobile robots. *Theoretical Computer Science*, 810:2–14, 2020.

7 Francine Berman, Frank Thomson Leighton, and Lawrence Snyder. Optimal tile salvage. Technical Report 81-396, Purdue University, Department of Computer Sciences, 1981.

8 H Bronnimann. Almost optimal set covers in finite vc-dimension. *Discrete Comput. Geom.*, 14:263–279, 1995.

9 Iago A Carvalho, Thomas Erlebach, and Kleitos Papadopoulos. On the fast delivery problem with one or two packages. *Journal of Computer and System Sciences*, 115: 246–263, 2021.

10 Jérémie Chalopin, Shantanu Das, Matúš Mihal' ák, Paolo Penna, and Peter Widmayer. Data delivery by energy-constrained mobile agents. In *Algorithms for Sensor Systems: 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS 2013, Sophia Antipolis, France, September 5-6, 2013, Revised Selected Papers 9*, pages 111–122. Springer, 2014.

11 Jérémie Chalopin, Riko Jacob, Matúš Mihalák, and Peter Widmayer. Data delivery by energy-constrained mobile agents on a line. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II 41*, pages 423–434. Springer, 2014.

12 Jérémie Chalopin, Shantanu Das, Yann Disser, Arnaud Labourel, and Matúš Mihalák. Collaborative delivery on a fixed path with homogeneous energy-constrained agents. *Theoretical Computer Science*, 868:87–96, 2021.

13 Timothy M Chan and Qizheng He. Faster approximation algorithms for geometric set cover. *arXiv preprint arXiv:2003.13420*, 2020.

14 Richard Church and Charles R Velle. The maximal covering location problem. *Papers in regional science*, 32(1):101–118, 1974.

15 Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

16 Pedro Henrique Cruz Caminha, Rodrigo de Souza Couto, Luis Henrique Maciel Kosmalski Costa, Anne Fladenmuller, and Marcelo Dias de Amorim. On the coverage of bus-based mobile sensing. *Sensors*, 18(6):1976, 2018.

17 Gautam K Das, Robert Fraser, Alejandro López-Ortiz, and Bradford G Nickerson. On the discrete unit disk cover problem. In *International Workshop on Algorithms and Computation*, pages 146–157. Springer, 2011.

18 Thomas Erlebach, Kelin Luo, and Frits C. R. Spieksma. Package Delivery Using Drones with Restricted Movement Areas, September 2022. URL `http://arxiv.org/abs/2209.12314`. arXiv:2209.12314 [cs].

19 Guy Even, Dror Rawitz, and Shimon Moni Shahar. Hitting sets when the vc-dimension is small. *Information Processing Letters*, 95(2):358–362, 2005.

20 Robert J. Fowler, Michael S. PaLerson, and Steven L. Tanimoto. Optimal packing and covering the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981.

21 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3(1):65–85, 2012.

22 Frank Harary and Leo Moser. The Theory of Round Robin Tournaments. *The American Mathematical Monthly*, 73(3):231–246, 1966. ISSN 0002-9890. doi: 10.2307/2315334. URL `https://www.jstor.org/stable/2315334`. Publisher: Mathematical Association of America.

23 D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 3(2):182–195, 1982.

24 Richard M Karp. *Reducibility among combinatorial problems.* Springer, 2010.

25 Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, and Denis Pankratov. The en route truck-drone delivery problem. *arXiv preprint arXiv:2402.00829*, 2024.

26 Saurav Pandit, Sriram V Pemmaraju, and Kasturi Varadarajan. Approximation algorithms for domatic partitions of unit disk graphs. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 312–325. Springer, 2009.

27 Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the twenty-eighth annual ACM Symposium on Theory of Computing*, pages 435–441, 1996.

A 2-Approximation algorithm for drone delivery on a path

In this section, we propose a 2-approximation algorithm for the drone delivery problem on a path with initial positions. The problem involves a set of n mobile agents (represented as drones), each with a designated traverse interval $[a_i, b_i)$, an initial position s_i, and a velocity v_i. The goal is to deliver a package from the starting point 0 to the ending point 1, minimizing the total time. Note that if a drone carries the package from l_i to r_i, it must first move from L_i to l_i. All drones can move at the same time.

▶ **Theorem 23.** *For any α-approximation algorithm for the SIC problem, there is a corresponding 2α-approximation algorithm for the drone delivery problem on a path with initial positions.*

Proof. Assume there is a feasible solution that takes t time for the drone delivery problem. Our general idea is to spend t extra time for each drone i to select its l_i and reserve t time for carrying the package. Henceforth, we no longer need to consider the time spent when a drone is waiting for the package at l_i or the package is waiting for drone i to reach l_i, i.e., we can "cover" the whole range $[0, 1)$ "at the same time".

Specifically, we transform a drone delivery problem instance into the corresponding SIC problem instance by setting $w_i = \frac{1}{v_i}$ and $[L_i, R_i) = [\max(a_i, s_i - v_i \cdot t), \min(b_i, s_i + v_i \cdot t))$, then apply any algorithm for the SIC problem. Since there is a solution where the whole delivery process takes no more than t time, it costs at most t to cover the entire range $[0, 1)$ with the subintervals $[l_i, r_i)$. Correspondingly, the package is carried from 0 to 1 in at most t time. And note a drone cannot leave $[L_i, R_i)$ within t time, so the feasibility is maintained after adjusting each traverse range $[a_i, b_i)$ to interval $[L_i, R_i)$. Thus, we have constructed a feasible solution for the drone delivery problem, taking at most $t + t = 2t$ time.

We claim that determining whether $\mathsf{OPT} = 0$ can be achieved in polynomial time by checking if the union of zero-cost intervals covers $[0, 1)$. If $\mathsf{OPT} \neq 0$, then $\mathsf{OPT} \geq w_{\min} \cdot d_{\min}$, where $d_{\min}$ is the smallest non-zero difference between any two endpoints in the input. Note that $\log(1/w_{\min})$ and $\log(1/d_{\min})$ serve as lower bounds on the input size.

Now we are to approach OPT by performing a binary search with initial bounds $t_{\min} = 0$ and $t_{\max} = w_{\max}$. Specifically, for the current midpoint t in the binary search process, we adjust each $[L_i, R_i)$ according to the description above. Let t^* be the cost given by the SIC algorithm. If $t < t^*$, set $t_{\min} = t$. Otherwise, set $t_{\max} = t$. After $O(\log(1/\epsilon))$ rounds of SIC algorithm calls, which is polynomial in terms of the input size, the precision is $\epsilon \cdot w_{\max}$. By setting $\epsilon = \frac{w_{\min} \cdot d_{\min}}{n \cdot w_{\max}}$, the solution we obtained $\mathsf{ALG} \leq 2(\mathsf{OPT} + \epsilon \cdot w_{\max}) = (2 + o(1)) \cdot \mathsf{OPT}$.

If the SIC algorithm provides an α-approximation to OPT, the solution is then a 2α-approximation for the drone delivery problem.

◀

The theorem provides a 2-approximation reduction from the package delivery problem with initial positions to the SIC problem. Following our results, there are 2-approximation algorithms for the cases 1) two distinct velocities; 2) a constant number of distinct velocities and the traverse intervals form a laminar family.