A Sub-quadratic Time Algorithm for the Proximity Connected k-center Problem on Paths via Modular Arithmetic

Binay Bhattacharya* Tsunehiko Kameda* Amirhossein Mozafari*†

Abstract

The k-center problem is one of the most well-known problems in combinatorial optimization which has been extensively studied in the past. In this paper, we introduce a generalized version of the k-center problem called *proximity connected k-center (PCkC)* problem. In this problem, we are given a set of demand points in a metric space and a parameter $\delta > 0$. We are going to locate k center points such that the maximum distance of a demand point to its nearest center is minimized and each pair of centers can communicate with each other either directly or via other centers assuming that each center can directly communicate with any other center within the range of δ of itself. Note that when δ is large enough, the problem turns to the k-center problem and when δ tends to zero, the problem turns to the 1-center problem. We consider the PCkC problem when the underlying space is a path and present a sub-quadratic time algorithm for both the unweighted and the weighted demand points cases.

1 Introduction

The k-center problem is one of the most important facility location problems which has been extensively studied in the past [4, 6, 10, 11, 16]. In this problem, we are given a set of n demand points $U = \{v_1 \dots, v_n\}$ in a metric space such that each demand point $v_i \in U$ has a non-negative weight w_i. The objective is to find a k-center (a set of k points in the space) C such that $cost(C) := \max_{v_i \in U}\{w_i d(v_i, C)\}$ is minimized, where $d(v_i, C) := min_{c \in C} d(v_i, c)$ (here $d(v_i, c)$ is the distance between v_i and c in the space). We call this minimum cost the *optimal cost* for the problem. If we have unit weights on all demand points, the problem is called *unweighted.* We say that a k-center C satisfies the *proximity connectedness condition (PCC)* with respect to a parameter $\delta > 0$ if the δ-distance graph of C is connected (the δ-distance graph of C is a graph with the vertex set C such that there is an edge between c_1 an c_2 in C if and only if $d(c_1, c_2) \leq \delta$). In the *proximity connected k-center (PCkC)* problem, in addition to U, we are also given a parameter $\delta > 0$ and we are going to find a k-center with the minimum cost that satisfies the PCC.

In practice, if we consider the centers as facility locations, the parameter δ can represent the range for which, each facility can directly communicate with any other facility within the range δ of itself. So, if the centers satisfy the PCC, each pair of facilities can communicate with each other directly or via other facilities. For example, suppose that we need to locate k communication/control equipment to observe n sensors while the equipment need to send/receive messages between themselves (directly or via other equipment). Also, each equipment can safely send/receive data with any other equipment within the range δ of itself. The problem of locating the equipment as close as possible to the sensors can be modeled as PCkC problem in the plane.

Note that if δ is sufficiently large, the problem reduces to the k-center problem which is known to be NP-hard in both the plane and metric graphs [6, 13] (a metric graph is a graph for which each of its edges has a length and the lengths satisfy the triangular inequality). This implies that the PCkC problem is also NP-hard in the plane and metric graphs and so it is not possible to solve it efficiently. In [6], Kariv and Hakimi showed that the k-center problem can be solved in polynomial time when the underlying space is a metric tree and gave an $O(n^2 \log n)$ time algorithm for the problem. In 1991, Frederickson [4, 5] showed that the unweighted k-center problem can be solved in linear time in trees. Finally, in 2018, Wang and Zhang [16] provided an $O(n \log n)$ time algorithm for the k-center problem in trees. The PCC condition first appeared in the context of wireless networks in 1992 [7]. Later, Huang and Tsai studied the 2-center problem in the plane, considering the proximity condition between the centers [8, 9]. As another work, in 2022, Bhattacharya et al. [2] presented an $O(n^2 \log n)$ time algorithm to solve the proximity connected 2-center problem in the plane improving the previous algorithm for the problem with $O(n^5)$ time complexity [7]. Although there are some related works in the context of theory of wireless sensor networks [1, 14], the k-center problem has not been studied when we have the proximity condition between the centers. In this paper, we address this problem by providing a sub-quadratic time algorithm for the k-center problem on paths having the PCC.

2 PCkC Problem for Unweighted Paths

Let $P = (v_1, \dots, v_n)$ be the given unweighted path (consisting of both the vertices and the edges between them) such that the vertices lie on the x-axis from left to right and v_1 lies on the origin. Without loss of generality, we assume that n is a power of 2. Also, we use the notation v_i $(1 \leq i \leq n)$ for both the vertex itself and the

*School of Computing Science, Simon Fraser University

†amozafar@sfu.ca, corresponding author

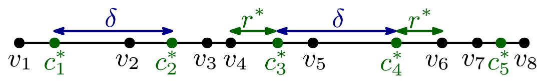
δ
r^*
δ
r^*
v_1 c_1^* v_2 c_2^* v_3 v_4 c_3^* v_5 c_4^* v_6 v_7 c_5^* v_8

Figure 1: An example of the PCkC problem on a path with 8 vertices.

x-coordinate of the vertex. Thus, we have an order on the vertices based on their x-coordinates. Also, if $v_i < v_j$, we denote the interval between v_i and v_j on the x-axis by $[v_i, v_j]$. In this section, we are going to find a k-center C^* for P such that C^* satisfies PCC and,

$$cost(C^*) = min\{cost(C) : C \text{ is a } k\text{-center for } P \text{ and satisfies PCC}\}$$

We call C^* an *optimal solution* and its cost the *optimal cost*. We denote the optimal cost by r^* and the centers in C^* by $(c_1^*, \ldots, c_k^*)$ from left to right on the x-axis. Figure 1 shows an example of the PCkC problem on a path and its corresponding optimal solution.

The idea of obtaining an optimal solution for the problem is first computing r^* and then, using it to build an optimal solution. In order to do that, we first design a feasibility test for the problem which gets a value $r \geq 0$ and determines whether it is feasible ($r \geq r^*$) or infeasible ($r < r^*$). Algorithm UPATH-FT presents such a feasibility test for the unweighted PCkC problem on a path. Note that if $r \geq r^*$, UPATH-FT(P,r) also gives us a k-center with a cost at most r. Using the feasibility test, we can check whether $r^* = 0$. In this case the trivial solution is putting a center at each vertex. So henceforth, we assume that $r^* > 0$. Note that in Algorithm 1, the vertices in V are

Algorithm 1 UPATH-FT(P, r)

```
1: Set $Counter = 1$ and $V = (v_2, \ldots, v_n)$.
2: Put a center at $x_c = r$.
3: while there is an element in $V$ do
4:     Eliminate all vertices $v \in V$ with $d(x_c, v) \leq r$.
5:     Put a center at $x_c = min\{x_c + \delta, V[1] + r\}$.
6:     $Counter = Counter + 1$.
7:     if $Counter > k$ then
8:         return infeasible.
9:     end if
10: end while
11: return feasible.
```

eliminated in order and so the time complexity of UPATH-FT would be $O(n + k)$. It is important to mention that we might have more than one optimal solution for a given problem instance but, having r^* (which is unique), the algorithm UPATH-FT gives us a unique optimal solution. In order to avoid confusion, henceforth we exclusively use the notation C^* for this optimal solution. We say that a vertex v is *covered* by a center $c_i^* \in C^*$ if $d(v, c_i^*) = d(v, C^*)$. Also, $d(v, c_i^*)$ is called the cost that c_i^* induces on v. We say that a sequence of t points $(c_1, \ldots, c_t)$ (the order is left to right on the x-axis) is a *t-train* if $\forall\, 1 \leq i < t,\ d(c_i, c_{i+1}) = \delta$.

Proposition 1 *There exists a pair of vertices (v_i, v_j) such that the subset $C' \subseteq C^*$ of centers in $[v_i, v_j]$ is a t-train (for some t) and $d(v_i, C') = d(v_j, C') = r^*$.*

The reason of the above proposition is that if such a pair does not exist, for any vertex v with $d(v, C^*) = r^*$, we can move the covering center of v (and possibly other centers to ensure the PCC) towards v to get a solution with a cost smaller than r^*, which contradicts the optimality of r^*. We call any pair (v_i, v_j) satisfying the condition of Proposition 1, a *determining pair* for the problem.

Proposition 2 *If $d(v_1, v_n) \geq k\delta$, then (v_1, v_n) is a determining pair for the problem.*

Proof. For any vertex v in $[c_1^*, c_k^*]$, $d(v, C^*)$ should be at most $\delta/2$ because of the PCC. So, if $d(v_1, v_n) \geq k\delta$, the cost of C^* should be greater than or equal to $\delta/2$ which means that (v_1, v_n) is a determining pair. □

Based on the above proposition, if $d(v_1, v_n) \geq k\delta$, we have $d(c_1^*, c_k^*) = (k-1)\delta$ and $d(v_1, c_1^*) = d(c_k^*, v_n)$. Therefore, $r^* = (d(v_1, v_n) - k\delta)/2$. Now, UPATH-FT($P, r^*$) will give us C^*. Henceforth in this section, we assume that $d(v_1, v_n) < k\delta$ and so $0 < r^* < \delta/2$ (because of the PCC). In order to find r^*, we build a set of candidate values $\mathcal{C}$ and iteratively use the feasibility test to discard its values until r^* becomes clear. Consider a pair of vertices (v_i, v_j) and a t-train T such that $d(v_i, v_j) > (t-1)\delta$. We say that T is *fitted* in $[v_i, v_j]$ if $d(v_i, T) = d(v_j, T)$. Note that if T is fitted in $[v_i, v_j]$, the *induced cost* of T on v_i and v_j would be $(d(v_i, v_j) - (t-1)\delta)/2$ and is denoted by $IC_t(v_i, v_j)$. If $d(v_i, v_j) \leq (t-1)\delta$, we say that (v_i, v_j) does not accept a t-train. Note that any pair of vertices accepts 1-train which is indeed the mid-point of the connecting segment of v_i and v_j. Based on Proposition 1, the set of candidate values $\mathcal{C}$ can be considered as follows:

$$\mathcal{C} = \{IC_t(v_i, v_j) \ : \ (v_i, v_j) \text{ accepts a } t\text{-}train\}$$

Because each pair of vertices can *generate* up to $O(k)$ candidate values, the size of $\mathcal{C}$ would be $O(n^2k)$. A naive algorithm to find r^* is computing the entire $\mathcal{C}$, then sort it and perform binary search using the feasibility test to find r^*. It is easy to see that the time complexity of this approach is $O(n^2k\log(n + k))$. In the rest, we show that how we can reduce this bound and get a sub-quadratic algorithm but before, it is useful to discuss about the geometric interpretation of the candidate values.

Geometric View: Let L_i and R_i be two half-lines from v_i with angles $\pi/4$ and $3\pi/4$ with the positive direction of the x-axis respectively. Note that the y-coordinate of the intersection of a vertical line at point x with $L_i \cup R_i$ is the cost that a center at x will induce on v_i (this is because we assumed that the vertices are unweighted). Based on this observation, for a pair (v_i, v_j), $IC_1(v_i, v_j)$ is the y-coordinate of the intersection point of R_i and L_j. Furthermore, if (v_i, v_j) accepts a t-train, $IC_t(v_i, v_j)$ would

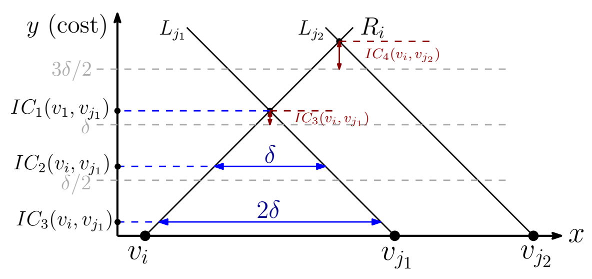

Figure 2: The geometric view of the candidate values generates by (v_i, v_{j_1}) and the effective candidate value generated by (v_i, v_{j_2}).

be the y-coordinate of the horizontal segment with length $(t-1)\delta$ with sides on R_i and L_j (see Figure 2). Based on this geometric view, the following observation can be concluded:

Observation 1 *If (v_i, v_j) accepts a t-train $(t > 1)$ then $IC_t(v_i, v_j) = IC_{t-1}(v_i, v_j) - \delta/2$.*

Consider a pair (v_i, v_j) and the non-zero candidate value $IC_{k'}(v_i, v_j)$ such that either $k' = k$ or (v_i, v_j) does not accept a $(k'+1)$-train (equivalently, k'-train is the longest train that can be fitted in (v_i, v_j)). According to Observation 1, $IC_{k'}(v_i, v_j)$ is the only candidate value that (v_i, v_j) can generate in $(0, \delta/2)$. If (v_i, v_j) generates a candidate value in $(0, \delta/2)$, we call this candidate value an *effective candidate value*. Because $r^* \in (0, \delta/2)$, we only need to search the effective candidates generated by the pairs in P in order to find r^*. Let us gather all the effective candidates into an $n \times n$ matrix M such that $M[i,j]$ is the effective candidate value generated by (v_i, v_j) if $i < j$ and zero otherwise. We can see that M is not a sorted matrix because for a fixed i, by increasing j, the number of centers in the train that induces $M[i,j]$ might change. Indeed, this is the main obstacle to get a linear time algorithm like [4, 5] for the unweighted PCkC problem. Precisely, the k-center problem is equivalent to the PCkC problem when $\delta = \infty$. In this case, all the effective candidates are generated by 1-trains. The key point here is that the effective cost generated by a 1-train on a pair (v_i, v_i) is an increasing function of $d(v_i, v_j)$. This monotonicity makes the matrix M sorted which plays a pivotal role in obtaining a linear time algorithm.

In order to search M in a sub-quadratic time, we define an auxiliary matrix $\bar{M}$ such that applying the feasibility test on its elements enables us to discard the elements of M in an efficient way. We define $\bar{M}$ as an $n \times n$ matrix such that:

$$\bar{M}[i,j] = max\{M[i,j'] : i < j' \leq j\}$$

Note that $\bar{M}$ is a row sorted (increasing) matrix but may not be sorted column-wise. We define the remainder function $rem_\delta(x)$ as follows:

$$rem_\delta(x) = x - \left\lfloor \frac{x}{\delta} \right\rfloor \times \delta$$

Observation 2 *If $i < j$, then we would have $M[i,j] = rem_\delta(d(v_i, v_j))/2$.*

This is from the fact that the size of the portion of $[v_i, v_j]$ not covered by the longest train in the interval is $rem_\delta(d(v_i, v_j))$.

Proposition 3 *If $M[i,j] = r^*$ then for all $i < j' < j$, $M[i,j'] \leq r^*$.*

Proof: We proceed by contradiction. Suppose that $M[i,j] = r^*$ and $\exists j' : i < j' < j$ such that $M[i,j'] > r^*$. Let $C' = (c^*_{h_1}, \ldots, c^*_{h_2}) \subseteq C^*$ be the train in $[v_i, v_j]$ that induces r^* on v_i and v_j. Also, let $C = (c_1, \ldots, c_q)$ be the longest train that can be fitted in $[v_i, v_{j'}]$ that induces the cost $M[i,j']$. Note that $|C| < |C'|$, otherwise because $v'_j < v_j$, $M[i,j']$ could not be greater than $M[i,j]$. Note that $c^*_{h_1} < c_1$ because we assumed $M[i,j'] > r^*$. Now, if $c^*_{h_1+q} < v_{j'}$, we can fit a $(q+1)$-train in $[v_i, v_{j'}]$, which contradicts the way we chose C. So, let us assume that $c^*_{h_1+q} > v_{j'}$ (see Figure 3).

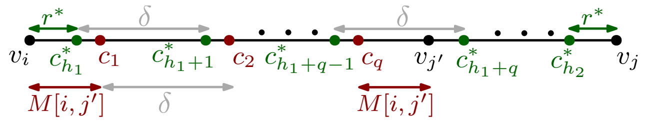

Figure 3: Proof of Proposition 3.

Here, $c^*_{h_1+q}$ is the center that covers $v_{j'}$ in C^*. If $d(v_{j'}, c^*_{h_1+q}) = r^*$, $d(v_i, v_{j'})$ would be a multiple of δ and so $M[i,j'] = 0$ which is against our assumption that $M[i,j'] > r^*$. Thus, we have $d(v_{j'}, c^*_{h_1+q}) < r^*$ but in this case we can fit a $(q+1)$-train in $[v_i, v_{j'}]$ which is a contradiction. $\square$

Example: In Figure 4, the fitted 4-train $(c^*_1, \ldots, c^*_4)$ between v_1 and v_j induces the optimal cost r^* for the problem. In order to have $M[i,j'] > r^*$ for some $1 < j' < j$, $v_{j'}$ should lie on a *forbidden region*, which are the set of points with distances greater than r^* to their closest center (these regions are specified in red in Figure 4).

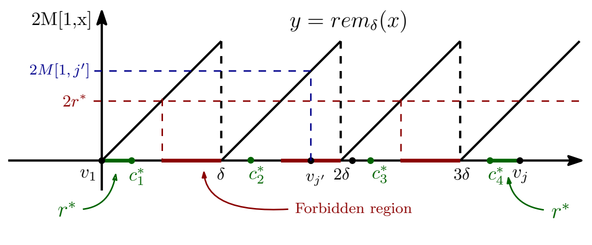

Figure 4: An example for Proposition 3.

Observation 3 *By applying the feasibility test on $\bar{M}[i,j]$, one of the following cases will happen:*

1. *$\bar{M}[i,j]$ is feasible. In this case, we can discard all $M[i,j']$ with $j' > j$ (based on Proposition 3).*

2. *$\bar{M}[i,j]$ is infeasible. In this case, we can discard all $M[i,j']$ with $j' \leq j$ (based on the definition of $\bar{M}$).*

Note that in the part 1 of the above observation, when $\bar{M}[i,j]$ is feasible, then either $\bar{M}[i,j] > r^*$ or $\bar{M}[i,j] = r^*$. For the former case, if $M[i,j'] = r^*$ for some $j' > j$, it contradicts Proposition 3 and for the later case we still have r^* in our undiscarded values. According to the above observation, we can find r^* by iteratively applying the feasibility test on the elements of $\bar{M}$ and discard the elements of M until r^* becomes clear. Algorithm DISC-ROUND(M) shows how we can discard 1/4th of the undiscarded elements in M at each iteration. We can see that at the beginning of each iteration the undiscarded elements of each row make a connected region. We call this region the *undiscarded region*. Because $\bar{M}$ is row sorted, if d_1 and d_2 are the first and the last indices of the undiscarded region of an i^{th}-row in M, if we know whether $\bar{M}[i, d_1 + \lfloor (d_1+d_2)/2 \rfloor]$ is feasible, we can discard half of the elements in the region. Note that in Algorithm 2, the variables d_1, d_2 and w_i can be updated after the discarding phase of the previous iteration (so we don't need to search the entire matrix to compute them at the beginning of the current iteration). Also, we compute the weighted median of the mid-indexes of the undiscarded region of the rows because at the beginning of an iteration, the undiscarded region of the rows in M may not have the same size. We can see that in each iteration, we need to compute the median of $O(n)$ values in $\bar{M}$. The bottleneck of the time complexity of DISC-ROUND is the cost of obtaining an element of $\bar{M}$. Precisely, if the time complexity of computing an element of $\bar{M}$ is $O(g(n))$, then the total time complexity of DISC-ROUND would be $O(ng(n)+k)$ and so the overall time complexity of our algorithm for the unweighted PCkC problem on paths would be $O((ng(n)+k)\log n)$ (because we have $O(\log n)$ iterations). In the next subsection, we discuss how we can compute an element of $\bar{M}$ efficiently.

Algorithm 2 DISC-ROUND(M)

```
1: for $i$ from 1 to $n$ do
2:     Set $d_1$, $d_2$ and $n_i$ as the first index, the last index
       and the number of elements in the undiscarded region
       of the $i^{th}$-row of $M$ respectively.
3:     Set $m_i$ as $\bar{M}[i, d_1 + \lfloor (d_1+d_2)/2 \rfloor]$.
4: end for
5: Compute the weighted median $m$ of $\{m_i : 1 \leq i \leq n\}$
   where $m_i$ has weight $n_i$.
6: Run UPATH-FT($P$,$m$).
7: if $m$ is feasible then
8:     For each $i$ with $m_i \geq m$, discard $M[i,j'] : j' > m_i$.
9: else
10:    For each $i$ with $m_i \leq m$, discard $M[i,j'] : j' \leq m_i$.
11: end if
```

2.1 Computing an Element of $\bar{\mathrm{M}}$

In this subsection, we provide a preprocessing phase that enables us to compute $\bar{M}[i,j]$ in sub-linear time. Let $\mathcal{M}_{i,j} = \{M[i,i+1], \ldots, M[i,j]\}$ and so, $\bar{M}[i,j] = max\ \mathcal{M}_{i,j}$. We first build a balanced binary tree $\mathcal{T}$ on top of the vertices in P (we assumed that n is a power of 2). Thus, each leave of $\mathcal{T}$ corresponds to a single vertex. For a node $\nu \in \mathcal{T}$, $span(\nu)$ is defined as the set of vertices that have ν as a common ancestor. Note that the root of $\mathcal{T}$ spans the entire P. Also, we denote the first and the last indexes of the vertices in $span(\nu)$ by $left(\nu)$ and $right(\nu)$ respectively. In each node $\nu \in \mathcal{T}$, we store the sequence $\sigma(\nu)$ obtained from sorting $\{2M[v_1, v] : v \in span(\nu)\}$ increasingly. It is easy to see that the time complexity of building $\mathcal{T}$ and the sequences in its nodes is $O(n \log n)$.

Observation 4 *For any two numbers a and b, we have:*

$$rem_\delta(a+b) = rem_\delta(rem_\delta(a) + rem_\delta(b))$$

Based on the above observation and Observation 2, for any $j' \geq i$ we can write $M[i,j']$ as:

$$\begin{aligned}&M[i,j'] = rem_\delta(d(v_i, v_{j'}))/2 = \\ &rem_\delta(d(v_1, v_{j'}) - d(v_1, v_i))/2 = \\ &rem_\delta(rem_\delta(d(v_1, v_{j'})) - rem_\delta(d(v_1, v_i)))/2 = \\ &rem_\delta(2M[v_1, v_{j'}] - 2M[v_1, v_i])/2\end{aligned}$$

Now, for each vertex ν with $\sigma(\nu) = (s_1, \ldots, s_t)$ and $i \leq left(\nu)$, we define $\sigma_i(\nu)$ as:

$$\sigma_i(\nu) = \big(rem_\delta(s_1 - 2M[v_1, v_i]), \ldots, rem_\delta(s_t - 2M[v_1, v_i])\big)$$

Let $\mu_i(\nu)$ be the maximum of $\sigma_i(\nu)$. Based on the above argument, we can see $max\{M[left(\nu), left(\nu)+1], \ldots, M[left(\nu), right(\nu)]\}$ is indeed $\mu_i(\nu)/2$. An important observation here is that because the elements of M are at most $\delta/2$, $\sigma_i(\nu)$ is a concatenation of two sorted sequences namely $\sigma_i^1(\nu)$ and $\sigma_i^2(\nu)$ (note that one of these sequences might be empty). So, in order to find $\mu_i(\nu)$, we need to compare the last elements of $\sigma_i^1(\nu)$ and $\sigma_i^2(\nu)$ (if they exist) and pick the greater value. Precisely, if $s_{j'} - 2M[v_1, v_i]$ is negative (resp. positive) for some $s_{j'} \in \sigma(\nu)$, $rem_\delta(s_{j'} - 2M[v_1, v_i])$ belongs to $\sigma_i^1(\nu)$ (resp. $\sigma_i^2(\nu)$). Thus, we can do binary search to obtain the index of the last element of $\sigma_i^1(\nu)$ and so $\mu_i(\nu)$ in $O(\log |span(\nu)|)$ time.

We can use the above data structure to find $\bar{M}[i,j]$ as follows: we first obtain two paths π_i and π_j and their split vertex ν_{split} from the root of $\mathcal{T}$ to v_i and v_j respectively. Let $\mathcal{V}_{i,j}$ be the set of right (resp. left) children of π_i (resp. π_j) from ν_{split} to its leaf (including v_j). Now, $\mathcal{M}_{i,j} = 1/2 \cup_{\nu \in \mathcal{V}_{i,j}} \sigma_i(\nu)$ where the multiplication is done element-wise. Therefore,

$$\bar{M}[i,j] = max\ \mathcal{M}_{i,j} = max\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\} \qquad (1)$$

because $|\mathcal{V}_{i,j}| = O(\log n)$ and computing each $\mu_i(\nu)$ in (1) also costs $O(\log n)$, the total complexity of computing $\bar{M}[i,j]$ would be $O(\log^2 n)$ which leads to an overall

$O\big((n\log^2 n + k)\log n\big)$ time complexity for the PCkC problem in unweighted paths.

Further improvements: First, we observe that if for two nodes $\nu, \nu' \in \mathcal{T}$, ν' is a parent of ν then $\sigma(\nu)$ is a sub-sequence of $\sigma(\nu')$. This property enables us to use a technique called *fractional cascading* [3] to avoid doing binary search on each of the nodes in $\mathcal{V}_{i,j}$ to find their maximum. Precisely, we equip each element s of $\sigma(\nu')$ with a pointer that points to the smallest element in ν larger than or equal to s. This structure can be constructed in $O(n\log n)$ time [3]. So, in order to obtain all $\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\}$, we only perform one binary search on $\sigma_i(root(\mathcal{T}))$ with cost $O(\log n)$ and follow the pointers along the paths to obtain each $\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}$ in a constant time. So, the total complexity of computing $\bar{M}[i,j]$ would be $O(\log n)$ and so, the total running time would be $O\big((n\log n + k)\log n\big)$.

As another improvement, note that we only need to do binary search on $\sigma_i(root(\mathcal{T}))$ once for each row i in the entire algorithm. Also, by spending $O(n\log n)$ time, for each root-leaf path π_i and each $\nu' \in \pi_i$, we can store $max\{\mu_i(\nu)\ :\ \nu \text{ is right child of a node in } \pi_i[\nu', v_i]\}$ in ν' ($\pi_i[\nu', v_i]$ is the portion of π_i from ν' to v_i) by walking along π_i twice. So, having ν_{split}, we only need to take care about computing $max\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}$ and hanging from $\pi_j\}$. To address this problem, consider a fixed i^{th}-row. Based on Algorithm 2, at each iteration r, the undiscarded region of the i^{th}-row corresponds to $span(\nu^r)$ for some $\nu^r \in \mathcal{T}$. Let ν^r_m be the left child of ν^r (if we are not at the last iteration) with $m^r = right(\nu^r_m)$. We can see that m^r is the median of the undiscarded region. Now, ν^{r+1}_m is either the left child of ν^r_m or the left child of the right neighbor of ν^r_m. Let r_0 be the last iteration for which $\nu^{r_0}_m$ is on π_i. For iterations $r \le r_0$, we only need to consider the maximum of the values in $\sigma_i(\nu')$ where ν' the first right child on π_i after ν^r_m. Also, for iterations $r > r_0$, we only need to have the set of maximum values in the left hanging nodes of $\pi_{m^r}[\nu_{split}, \nu^r_m]$ and ν^r_m itself. Now, it is easy to see that as r increases to $r+1$, these set of values can be updated in a constant time. Thus, we can conclude that computing $\bar{M}[i, m^r]$ for all iterations r only takes $O(\log n)$ time and because we have linear number of rows, we would have the following theorem:

Theorem 1 *The unweighted PCkC problem can be solved in $O((n+k)\log n)$ time.*

3 PCkC Problem for Weighted Paths

Let $P = (v_1, \ldots, v_n)$ be the given weighted path such that w_i is the weight of v_i. For a point x on P, we define $wd(v_i, x) = w_i d(v_i, x)$. Again each pair of vertices (v_i, v_j) generates $O(k)$ candidate values which corresponds to the trains that can be fitted in $[v_i, v_j]$. Here, because the weights of v_i and v_j might be different, a train may not be required to have the same distance from v_i and v_j in order to induce the same cost on them. Again, we denote the cost that a fitted t-train in $[v_i, v_j]$ induces on v_i and v_j by $IC_t(v_i, v_j)$. Suppose that $d(v_i, v_j) > t\delta$ for some $t > 1$. We define the *width* of (v_i, v_j) as $IC_{t-1}(v_i, v_j) - IC_t(v_i, v_j)$ and denote it by $W(v_i, v_j)$. Note that this value is independent of t and only depends on w_i and w_j and so, we can compute it in a constant time (in the unweighted case, the width of all pairs in P are $\delta/2$). Because here the widths of the pairs in P might not be equal, we first need to find an interval I^* such that each pair of vertices can generate at most one cost in I^*. But before going into that, we need to update our feasibility test to support weighted vertices. Algorithm 3 presents the feasibility test procedure WPATH-FT(P,r) which gets a weighted path P and a test value r and determines whether $r \ge r^*$ or $r < r^*$.

Algorithm 3 WPATH-FT(P, r)

```
 1: Set $Counter = 1$
 2: for i=1 to n do
 3:     Let $x_i$ be the point on the right side $v_i$ such that
        $wd(v_i, x_i) = r$.
 4: end for
 5: Let $X = (x_1, \ldots, x_n)$.
 6: Let $x_c = x_1$.
 7: while There is an element left in $X$ do
 8:     Eliminate $x_i$s from $X$ corresponding to the vertices
        for which $wd(v_i, x_c) \le r$.
 9:     Put a center at $x_c = min\{x_c + \delta, X[1]\}$.
10:     Counter = Counter + 1.
11:     if $Counter > k$ then
12:         return infeasible.
13:     end if
14: end while
15: return feasible.
```

Note that in the while loop of Algorithm 3, we eliminate x_is according to the order in the sequence X and so, the running time of the above feasibility test is $O(n+k)$. The geometric view for the weighted case is similar to the unweighted case but here, for each vertex v_i, the magnitude of the slopes of R_i and L_i is w_i. For each pair (v_i, v_j), the y-coordinate of the intersection point of R_i and L_j is the cost that a fitted 1-train (single point) in $[v_i, v_j]$ induces on v_i and v_j which is denoted by $IC_1(v_i, v_j)$. Similarly, if $d(v_i, v_j) > (t-1)\delta$, $IC_t(v_i, v_j)$ would be the y-coordinate of the horizontal segment with length $(t-1)\delta$ and endpoints on R_i and L_j (see Figure 5).

3.1 Matrix Search for Weighted Paths

First, we need to build an interval $I_1 = [a, b]$ such that $r^* \in I_1$ and it's interior does not contain any $IC_1(v_i, v_j)$ for any $i < j$ (note that $IC_1(v_i, v_j)$ is indeed the y-coordinate of the intersection point of R_i and L_j). If we use Lemma 2.5 [16] on all R_i and L_j ($1 \le i, j \le n$), we can get I_1 in $O((n+k)\log n)$ time. Let us define W^* as follows:

$$W^* = min\{W(v_i, v_j) : i < j \text{ and } IC_1(v_i, v_j) \ge b\}$$

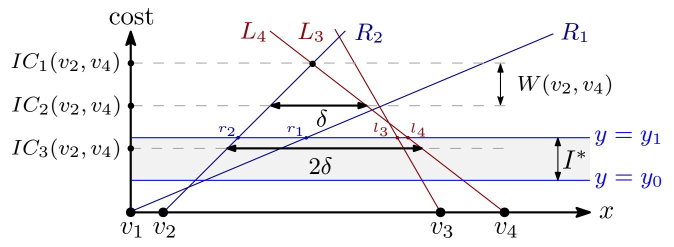

Figure 5: A weighted path (v_1, v_2, v_3, v_4), the width of (v_2, v_4) and three costs generated by the pair. Note that only one of them lies inside I^*.

We can see that $r_* \in I_2 := [b - kW^*, b] \cap I_1$. This is because r^* can't be smaller than the cost that a fitted k-train induces on the generating pair of W^*.

Proposition 4 *W^* can be computed in $O(n \log n)$ time.*

Proof. We first compute the intersection points of all L_i and R_j for $1 \leq i, j \leq n$ with the horizontal line $y = b$. Then, we sort these intersections on the line from left to right in $O(n \log n)$ time. So, each of these intersections corresponds to a line with a positive or a negative slope. We traverse these intersections from left to right and store the minimum positive slope and the minimum width we have seen in variables *min_slope* and *min_width* respectively. Finally, we set *min_width* as W^*. Precisely, when we visit an intersection point, if it came from a line with a positive slope, we update *min_slope* if necessary and if it came from a line with a negative slope, we compute the width it creates with the line that generated *min_slope* and update *min_width* if necessary. □

We can see that the length of I_2 is at most kW^*. This implies that by applying the feasibility test $O(\log k)$ times at the costs $b - iW^*$ $(0 \leq i \leq k)$ we get an interval $I_3 \subseteq I_2$ with length at most W^* containing r^*. Because W^* is the minimum width, each pair (v_i, v_j) with $IC_1(v_i, v_j) \geq b$ can generate at most one candidate value in I_3.

Consider the set of half-lines $\{R_1, \ldots, R_{n-1}\}$ (all with positive slopes) and their upper-envelope polygonal chain as a function $f_{UE}(x)$. We can see that f_{UE} is a piece-wise linear and an increasing function. Also, $f_{UE}(x)$ is the cost of covering all the vertices on the left side of x if we put a center at x. We can compute f_{UE} in linear time as follows: suppose that we have already computed the upper-envelope of $\{R_1, \ldots, R_{j-1}\}$ consisting of it's lines and break points. Now, when we add R_j and update our envelope, if R_j is below the last break points, we consider R_j and the last line of the envelope for a possible new break point. Otherwise, we find the first break point below the line (be checking the break points one by one from the last) and consider the line next to it (on its left) for a break point. Note that when we check a break point and it turns out it is below R_j, the line next to it (on its right) can never be a part of the envelope. Because we have linear number of lines, the time complexity of computing f_{UE} is linear.

Let $(x_1, \ldots, x_s)$ be the x-coordinates of the break points of f_{UE} where s is the number of break points. Then, we can use our feasibility test to do binary search on $\{f_{UE}(x_i) : 1 \leq i \leq s\}$ to find an interval $[x_q, x_{q+1}]$ such that $r^* \in [f_{UE}(x_q), f_{UE}(x_{q+1})]$. Let R_q (generated by v_q) be the line corresponding to the portion of f_{UE} in $[x_q, x_{q+1}]$. Then we have the following observation:

Observation 5 *If c_1^* induces r^*, then v_q is the first vertex of a determining pair.*

Based on the above observation, we can consider all pairs $\{(v_q, v_{q+1}), \ldots, (v_q, v_n)\}$, obtain the candidate value that each generates, sort them and do binary search (using our feasibility test) to get an interval $I^{(1)}$. Now, c_1^* can't generate any candidate value in the interior of $I^{(1)}$. Similarly, we can do the above process on $\{L_2, \ldots, L_n\}$ to get an interval $I^{(2)}$ such that c_k^* can't generate any candidate value in the interior of $I^{(2)}$. Let $I^* = I_3 \cap I^{(1)} \cap I^{(2)}$. So, it is only left to resolve the candidates in the interior of I^*.

Observation 6 *If (v_i, v_j) is a determining pair and a train $(c_{h_1}^*, \ldots, c_{h_2}^*)$ in $[v_i, v_j]$ induces r^* on the interior of I^*, then*

1. *$0 < d(v_i, c_{h_1}^*), d(v_j, c_{h_2}^*) < \delta/2$.*

2. *$(c_{h_1}^*, \ldots, c_{h_2}^*)$ is the longest train that can be fitted in $[v_i, v_j]$.*

The first part of the above observation comes from the fact that if r^* lies on the interior of I^*, then $h_1 \neq 1$ and $h_2 \neq k$. So, if for example $d(v_i, c_{h_1}^*) \geq \delta/2$ then because of the PCC, $c_{h_1-1}^*$ can cover v_i in the optimal solution. For the second part, note that if we are able to fit a longer train in $[v_i, v_j]$ then either $d(v_i, c_{h_1}^*)$ or $d(v_j, c_{h_2}^*)$ should be greater than $\delta/2$ which contradicts the first part.

Based on Observation 6, for any pair of vertices (v_i, v_j), we define our matrix M for the weighted case such that $M[i, j]$ is the cost r induced by the longest train $(c_1, \ldots, c_q)$ that can be fitted in $[v_i, v_j]$ if $r \in I^*$ and $0 < d(v_i, c_1), d(v_j, c_q) < \delta/2$. If we didn't have either of these two conditions, we assign $M[i, j] = 0$. It is clear that r^* is an element of M. Similar to the unweighted case, we define $\bar{M}[i, j]$ as $\max\{M[i, i+1], \ldots, M[i, j]\}$. Again, we can see that $\bar{M}$ is a row sorted matrix but may not be sorted column-wise. Next, we show Proposition 3 is still valid for our new definition of M and $\bar{M}$ in the weighted case.

Proposition 5 *If $M[i, j] = r^*$, then for all $i < j' < j$, $M[i, j'] \leq r^*$.*

Proof. We proceed by contradiction. Suppose that (v_i, v_j) induces r^* and $\exists i < j' < j$ such that $M[i, j'] > r^*$. Let $C = (c_1, \ldots, c_q)$ be the longest train that can be fitted in $[v_i, v_{j'}]$ and induces the cost $M[i, j']$ on v_i and $v_{j'}$. Also, let $C' = (c^*_{h_1}, \ldots, c^*_{h_2}) \subseteq C^*$ be the train that induces r^* in $[v_i, v_j]$. Now, $c^*_{h_1+q-1} < c_q$ (because we assumed that $M[i, j'] > r^*$) and $|C| < |C'|$ (because $v_j > v_{j'}$). We consider two cases:

case 1: $\boldsymbol{c^*_{h_1+q} \leq v_{j'}}$: In this case, we could fit a $(q+1)$-train namely $C'' = (c'_1, \ldots, c'_{q+1})$ in $[v_i, v'_j]$ which contradicts the fact that C was the longest train in $[v_i, v_{j'}]$.

case 2: $\boldsymbol{c^*_{h_1+q} > v_{j'}}$: In this case, $v_{j'}$ should be covered from its right in C^* (because $c^*_{h_1+q-1} < c_q$ and we assumed $M[i, j'] > r^*$). Also, the cost of covering $v_{j'}$ in C^* should be no more than r^*. So, if $w_i \leq w_{j'}$, $d(v_i, c^*_{h_1}) \geq d(v_{j'}, c^*_{h_1+q})$ and thus, we can fit a $(q+1)$-train in $[v_i, v_{j'}]$ which is a contradiction.

Now, assume that $w_i > w_{j'}$. Let t_1 and t_2 be the points on the right side of $v_{j'}$ such that $wd(v_{j'}, t_1) = r^*$ and $wd(v_{j'}, t_2) = M[i, j']$. Note that $t_2 > t_1$ and t_2 is the mirror image of c_q with respect to $v_{j'}$. Now, $d(c^*_{h_1}, c_1) < d(t_1, t_2)$ (because $w_i > w_{j'}$ and the cost that v_i induces on $c^*_{h_1}$ and c_1 are r^* and $M[i, j']$ respectively). Also, because $M[i, j'] \neq 0$, $d(c_q, v_{j'}) < \delta/2$ (based on the definition of M), $c_q + \delta > v_{j'} + \delta/2$ which implies that $[t_1, t_2] \subseteq [c^*_{h_1+q}, c_q + \delta]$. This contradicts the fact that $d(c^*_{h_1+q}, c_q + \delta) = d(c^*_{h_1}, c_1) < d(t_1, t_2)$ (see Figure 6). $\square$

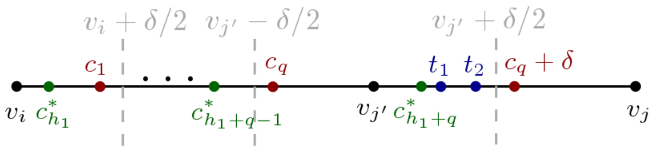

Figure 6: Proof of Proposition 5

The above proposition implies that Observation 3 is valid for M and $\bar{M}$ in the weighted case and so we can use Algorithm 2 to find r^* and get C^*. Based on Algorithm 2, the time complexity of finding r^* would be $O((ng(n)+k)\log n)$ where $g(n)$ is the time complexity for computing an element of $\bar{M}$. In the Appendix, we show how we can compute an element of $\bar{M}$ in $O(\log^3 n)$ time by spending $O(n \log^3 n)$ time for preprocessing. This gives us the following theorem:

Theorem 2 *The PCkC problem can be solved in $O((n \log^3 n + k) \log n)$ time.*

References

[1] Ahlberg M, Vlassov V, Yasui T. Router placement in wireless sensor networks. In *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems* 2006 Oct 9 (pp. 538-541). IEEE.

[2] Bhattacharya B, Mozafari A, Shermer TC. An efficient algorithm for the proximity connected two center problem. In *International Workshop on Combinatorial Algorithms* 2022 (pp. 199-213). Springer, Cham.

[3] De Berg M, Cheong O, Van Kreveld M, Overmars M. Computational geometry: introduction. Computational geometry: algorithms and applications. 2008:1-7.

[4] Frederickson GN. Parametric search and locating supply centers in trees. In *Workshop on Algorithms and Data Structures* 1991 Aug 14 (pp. 299-319). Springer, Berlin, Heidelberg.

[5] Frederickson GN. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms* 1991 Mar 1 (pp. 168-177).

[6] Hakimi SL. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations research.* 1965 Jun;13(3):462-75.

[7] Huang CH. Some problems on radius-weighted model of packet radio network, Doctoral dissertation, Ph. D. Dissertation, Dept. of Comput. Sci., Tsing Hua Univ., Hsinchu, Taiwan, 1992.

[8] Huang PH, Tsai YT, Tang CY. A near-quadratic algorithm for the alpha-connected two-center problem. *Journal of information science and engineering.* 2006 Nov 1;22(6):1317.

[9] Huang PH, Te Tsai Y, Tang CY. A fast algorithm for the alpha-connected two-center decision problem. *Information Processing Letters.* 2003 Feb 28;85(4):205-10.

[10] Jeger M, Kariv O. Algorithms for finding P-centers on a weighted tree (for relatively small P). *Networks.* 1985 Sep;15(3):381-9.

[11] Kariv O, Hakimi SL. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics.* 1979 Dec;37(3):513-38.

[12] Megiddo N. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM journal on computing.* 1983 Nov;12(4):759-76.

[13] Megiddo N, Supowit KJ. On the complexity of some common geometric location problems. *SIAM journal on computing.* 1984 Feb;13(1):182-96.

[14] Patel M, Chandrasekaran R, Venkatesan S. Energy efficient sensor, relay and base station placements for coverage, connectivity and routing. In *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference*, 2005. 2005 Apr 7 (pp. 581-586). IEEE.

[15] Toth CD, O'Rourke J, Goodman JE, editors. Handbook of discrete and computational geometry. CRC press; 2017 Nov 22.

[16] Wang H, Zhang J. An $O(n \log n)$-Time Algorithm for the k-Center Problem in Trees. *SIAM Journal on Computing.* 2021;50(2):602-35.

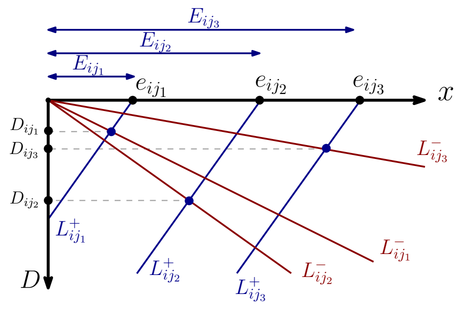

Figure 7: Three points e_{ij_1}, e_{ij_2} and e_{ij_3} located at distances E_{ij_1}, E_{ij_2} and E_{ij_3} respectively and their generating points. In this example, (v_i, v_{j_1}) generates the maximum of $\{M[i,j_1], M[i,j_2], M[i,j_3]\}$.

Appendix: Computing an Element of $\bar{M}$ for Weighted Paths

In this section, we build a data structure such that for any query pair (i,j) $(i < j)$, it enables us to compute $\bar{M}[i,j] = max\{M[i,j'] : i < j' \leq j\}$ in a sub-linear time. Suppose that $I^* = [y_0, y_1]$. We denote the x-coordinates of the intersection points of L_i and R_i $(1 < i < n)$ with line $y = y_1$ by l_i and r_i, respectively (see Figure 5). Note that if for a pair $(v_i, v_{j'})$, $l_{j'} < r_i$, it can not generate any candidate value in I^* (because of the way we built I^*) and so, $M[i,j'] = 0$. Thus, we only consider the pairs $(v_i, v_{j'})$ for which $l_{j'} \geq r_i$. Let us define the complement function with respect to δ as:

$$comp_\delta(x) = \left\lceil \frac{x}{\delta} \right\rceil \times \delta - x$$

We also denote $comp_\delta(l_i)$ and $comp_\delta(r_i)$ by $\hat{l}_i$ and $\hat{r}_i$ respectively where $1 < i < n$. For any pair (i,j') with $j' > i$ and $l_{j'} > r_i$, let $E_{ij'} = comp_\delta(l_{j'} - r_i) = rem_\delta(\hat{l}_{j'} - \hat{r}_i)$ and $D_{ij'} = E_{ij'}/(w_i^{-1} + w_{j'}^{-1})$ (note that w_i and $w_{j'}$ are the magnitudes of the slopes of R_i and $L_{j'}$ respectively). Based on the geometric view, it is easy to see that $M[i,j'] = y_1 - D_{ij'}$ if $D_{ij'} \leq |I^*|$ and zero otherwise. So, the problem of finding $\bar{M}[i,j]$ is equivalent to find $D_{min} = min\{D_{ij'} : i < j' \leq j\}$. It is convenient to visualize this set as follows: for each $i < j' \leq j$, we consider $e_{ij'}$ as the point located at $(E_{ij'}, 0)$ on the x-axis. Each $e_{ij'}$ has a half-line $L^+_{ij'}$ attached to it with slope w_i and a half-line $L^-_{ij'}$ from the origin with slope $-w_{j'}$ (see Figure 7). We can see that the distance between the intersection point of L^+_{ij} and L^-_{ij} from the x-axis is indeed D_{ij}. We call this distance the *D-coordinate* of the intersection (when a point moves downward, its D-coordinate increases). So, each value $E_{ij'}$ generates exactly one value $D_{ij'}$ call it the *D-value* of $E_{ij'}$. Like the unweighted case, we build a balanced binary tree $\mathcal{T}$ on top of the vertices and in each node $\nu \in \mathcal{T}$ we store $\{\hat{l}_h : v_h \in span(\nu)\}$ as an increasingly sorted sequence $\sigma(\nu)$. So, if we preprocess each $\nu \in \mathcal{T}$ such that for a given vertex v_i, we can quickly compute $\mu_i(\nu) = min\{D_{ih} : v_h \in span(\nu)\}$, we can decompose the set $\{v_j : i < j' \leq j\}$ into $\cup_{\nu \in \mathcal{V}_{i,j}} span(\nu)$ (as we did in Section 2.1) and set $D_{min} = min\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\}$

Let $\nu \in \mathcal{T}$ be a fixed node. In the rest, we show how we can preprocess ν such that given a query vertex v_i, we can efficiently compute $\mu_i(\nu)$. First, note that the set of half-lines $\{L^-_{ih} : v_h \in span(\nu)\}$ is independent of i. Also, for each i, $\{E_{ih} : v_h \in span(\nu)\}$ is the union of two sorted sequences $\sigma^1_i(\nu)$ and $\sigma^2_i(\nu)$, where $\sigma^1_i(\nu)$ (resp. $\sigma^2_i(\nu)$) is obtained by a shift (adding a constant value) of the elements in $\sigma(\nu)$ smaller than (resp. greater than or equal to) $\hat{r}_i$. Consider the lines $L^+_{ij'}(x) = w_i(x - e_{ij'})$ and $L^-_{ij'}(x) = -w_{j'}x$, where $e_{ij'}$ is a variable (see Figure 7). When $e_{ij'}$ increases, the D-value of $e_{ij'}$ (the intersection of $L^+_{ij'}$ and $L^-_{ij'}$) increases linearly. Let $f(x)$ be the minimum D-value generated by $\{e_{ij'} = \hat{l}_{j'} + x : j' \in span(\nu)\}$. We can see that $f(x)$ is the lower-envelope of a set of lines which can be computed in $O(|\nu| \log |\nu|)$ time ($|\nu|$ is the number of vertices in $span(\nu)$) using the divide-and-conquer algorithm (use the order in $\sigma(\nu)$ for breaking the vertices). Because we need to work with the sub-sequences of $\sigma(\nu)$, we store the entire *recursion tree* [15] (with the solutions of its subproblems) of the divide-and-conquer algorithm and denote this tree by $\mathcal{R}_i(\nu)$. Based on the above discussion, one way to preprocess ν is that for each $1 < i < n$, we compute and store $\mathcal{R}_i(\nu)$. Now, when we are given a vertex v_i, we first use binary search to get $\sigma^1_i(\nu)$ and $\sigma^2_i(\nu)$. Next, we use $\mathcal{R}_i(\nu)$ to get $\mu_i(\nu)$. Note that this process costs $O(\log^2 |\nu|)$ time (one $O(\log |\nu|)$ factor because of the height of $\mathcal{R}_i(\nu)$ and the other for binary search to get the minimum point of the envelopes in the nodes of $\mathcal{R}_i(\nu)$ needed to construct $\sigma^1_i(\nu)$(resp. $\sigma^2_i(\nu)$) at an specific x-coordinate determined by the shift in $\sigma^1_i(\nu)$(resp. $\sigma^2_i(\nu)$). Because the height of $\mathcal{T}$ is $O(\log n)$, the total time complexity of computing $\bar{M}[i,j]$ would be $O(\log^3 n)$. Note that the values $\sigma(\nu)$ of any (non root) node $\nu \in \mathcal{T}$ is a subset of the values of its parent node. So, using the fractional cascading technique, this cost can be reduced to $O(\log^2 n)$ time.

The problem here is that if we build $\mathcal{R}_i(\nu)$ for all $1 < i < n$ and all $\nu \in \mathcal{T}$, the time complexity of the preprocessing phase would be $O(n^2 \log^2 n)$. In order to make the preprocessing cost sub-quadratic, consider an internal node ω of the recursion tree of ν (note that the vertices in ω are independent of i). Let $\tau_i(\omega)$ be the sequence of points in ω who generate a line in its corresponding lower-envelope in $\mathcal{R}_i(\nu)$ where the order is according to the appearance of the lines in the envelope.

Proposition 6 *If for two indices i_1 and i_2, $w_{i_1} < w_{i_2}$, then $\tau_{i_2}(\omega) \subseteq \tau_{i_1}(\omega)$*

The proof of the above proposition is straightforward using elementary geometry. In order to use the above proposition, we first sort all the slopes increasingly into a sequence $(w_{i_1}, \ldots, w_{i_n})$. Now, when we preprocess ω, instead of building $\mathcal{R}_i(\omega)$ for all $1 < i < n$, we can use a binary tree structure for the slopes which leads to $O(|\omega| \log n)$ time complexity for preprocessing ω. This, reduces the overall preprocessing time to $O(n \log^3 n)$ and increases the time complexity of computing $M[i,j]$ to $O(\log^3 n)$.