Exact Subquadratic Algorithm for Many-To-Many Matching on Planar Point Sets with Integer Coordinates

Seongbin Park
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Eunjin Oh
Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea

Abstract

In this paper, we study the many-to-many matching problem on planar point sets with integer coordinates: Given two disjoint sets $R, B \subset [\Delta]^2$ with $|R|+|B| = n$, the goal is to select a set of edges between R and B so that every point is incident to at least one edge and the total Euclidean length is minimized. In the general case that R and B are point sets in the plane, the best-known algorithm for the many-to-many matching problem takes $\tilde{O}(n^2)$ time. We present an exact $\tilde{O}(n^{1.5} \log \Delta)$ time algorithm for point sets in $[\Delta]^2$. To the best of our knowledge, this is the first subquadratic exact algorithm for planar many-to-many matching under bounded integer coordinates.

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

Keywords and phrases Edge cover, many-to-many matching, similarity, geometric matching

Digital Object Identifier 10.4230/LIPIcs.SWAT.2026.36

Related Version *Full Version*: https://arxiv.org/abs/2604.16921

Funding Supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00440239, Sublinear Scalable Algorithms for Large-Scale Data Analysis) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00358505).

1 Introduction

Measuring the similarity between two point sets is a fundamental problem in computational geometry and pattern recognition [1, 22, 23]. A widely used measure is the *Earth Mover's Distance* (EMD), which captures the global distribution of point sets by finding an optimal correspondence [18, 19]. Formally, for two sets R and B of the same cardinality, the EMD is defined as the minimum cost of a perfect matching: $\min_{\sigma:R\to B} \sum_{r\in R} \|r - \sigma(r)\|$, where σ is a bijection. While EMD is effective at preserving the holistic structure of the sets, its definition is inherently restricted to cases where $|R| = |B|$. To accommodate sets of different cardinalities, one might consider *pointwise* heuristics such as the Chamfer distance: $\sum_{r\in R} \min_{b\in B} \|r - b\| + \sum_{b\in B} \min_{r\in R} \|b - r\|$. It provides a simple way to handle unequal set sizes as it treats each connection independently and sums $|R| + |B|$ distances without any global coordination. However, this often fails to capture the underlying structural correspondence, as it lacks the "assignment" nature that makes EMD robust.

A more principled approach to handling different sized sets, instead of resorting to pointwise measures, is to generalize the matching framework of EMD itself. By relaxing the strict one-to-one matching requirement to a *many-to-many* correspondence, we arrive at the *minimum link measure* introduced by [7]. Formally, a many-to-many matching is a set of edges $M \subseteq R \times B$ such that every point in $R \cup B$ is incident to at least one edge in M.

The cost is the total weight $\sum_{(r,b)\in M} \|r-b\|$. The minimum link measure is the minimum cost of a many-to-many matching between R and B. This "setwise" approach preserves the global structural integrity of EMD by seeking an optimal set of edges that covers both sets simultaneously. As a single edge in M can satisfy the coverage requirement for points in both R and B, the minimum link measure maintains a mutual correspondence that is more stable and geometrically intuitive than independent pointwise distances.

In this paper, we consider the problem of computing a minimum-cost many-to-many matching, also called the minimum link measure, between two point sets in Euclidean space. This problem was originally introduced by Eiter and Mannila [7]. While an efficient, optimal algorithm exists for points on a line, the problem becomes significantly more difficult in higher dimensions. Even for planar point sets, the best-known exact algorithm still suffers from a quadratic bottleneck [3], mirroring the challenges found in various other geometric bipartite matching problems [11]. To bridge this gap, we consider the case where the inputs lie on an *integer grid* as an intermediate step toward the general planar case.

Our results. In this paper, we present the first subquadratic-time exact algorithm for the many-to-many matching problem for point sets on an integer grid. Our main result is summarized in the following theorem.

▶ **Theorem 1.** *Given two disjoint sets*[1] R *and* B *of points in* $[\Delta]^2 = \{1, 2, \ldots, \Delta\} \times \{1, 2, \ldots, \Delta\}$*, we can compute a minimum-cost many-to-many matching between* R *and* B *in* $\tilde{O}(n^{1.5} \log \Delta)$ *time,*[2] *where* $n = |R| + |B|$.

A key technical challenge in achieving this bound is efficiently handling subproblems where points can be either matched to the opposite set or remain unmatched by paying a certain cost. To this end, we introduce the *minimum-cost matching with penalties* problem: Given two point sets R and B where each point $p \in R \cup B$ is associated with a real-valued penalty $\pi(p) \geq 0$, we seek a matching $M \subseteq R \times B$ that minimizes $\sum_{(r,b)\in M} \|r-b\| + \sum_{p\in \mathrm{free}(M)} \pi(p)$, where $\mathrm{free}(M)$ denotes the set of points in $R \cup B$ not incident to any edge in M. To the best of our knowledge, this problem has not been explicitly studied before. Although this can be solved in $O(n^3)$ time via standard reductions to the minimum-cost perfect matching problem for bipartite graphs, the cubic-time algorithm is too slow for our purpose. As a subroutine for our main algorithm, we provide an optimal algorithm for this penalty variant in one dimension, which we believe is of independent interest, stated as follows.

▶ **Theorem 2.** *Given two disjoint sets* R *and* B *of points on a real line, where each point has a real-valued penalty, a minimum-cost matching with penalties can be computed in* $O(n \log n)$ *time, where* $n = |R| + |B|$.

Related works. The study of the minimum-cost many-to-many matching problem was introduced by Eiter and Mannila [7]. They established that the problem is solvable in $O(n^3)$ time by reducing it to the minimum-cost perfect matching problem on graphs [7]. For the one-dimensional case where points lie on a real line, Colannino et al. [5] provided an optimal $O(n \log n)$-time algorithm, improving upon previous $O(n^2)$ results [6]. In the Euclidean plane, the best-known exact algorithm runs in $\tilde{O}(n^2)$ time [3]. On the other hand, in higher dimensions, no non-trivial algorithm is known for this problem while an $O(n^3)$-time algorithm

[1] With a slight modification, we can deal with the non-disjoint case without increasing the running time.
[2] The $\tilde{O}$-notation hides polylogarithmic factors in n.

can be obtained easily by reducing it to a minimum-cost perfect matching of a graph of complexity. Given the challenge of computing exact solutions in subquadratic time, several approximation algorithms have been explored. The Chamfer distance serves as a simple 2-approximation for the many-to-many matching cost [3]. Furthermore, $(1+\varepsilon)$-approximate solutions can be computed in $(1/\varepsilon)^{O(d)} \cdot n \log n$ time for d-dimensional Euclidean space, or any metric space with a constant doubling dimension [2, 4].

As EMD is a more classical setting, there are numerous results on computing the EMD between two points (also known as the minimum-cost perfect matching). By implementing the Hungarian algorithm [15] using a dynamic weighted nearest neighbor data structure [12], one can compute the exact EMD of two point sets in the plane in $\tilde{O}(n^2)$ time [11]. While the exact bipartite matching problem in the plane suffers from a quadratic bottleneck, the problem becomes easier if there are constraints on the input points. For instance, if the points in B and R are drawn independently and identically from a fixed distribution that is not known to the algorithm, the exact EMD between B and R can be computed in $O(n^{7/4} \log \Phi)$ time, where Φ denotes the spread of $B \cup R$ (i.e, the ratio of the maximum distance to the minimum distance). If B and R come from $[\Delta]^2$, Sharathkumar [20] presented an $\tilde{O}(n^{1.5} \log \Delta)$-time algorithm.[3]

2 Preliminaries and Algorithm Overview

In this section, we first describe an alternative view for the problem and then provide an overview of our algorithm. An alternative way is to view the many-to-many matching problem as a geometric graph problem. Consider a complete bipartite graph $G = ((R, B), E)$ with vertex set $V = R \cup B$ and with edge set $E = R \times B$ where the cost of an edge $(r, b) \in E$ is the Euclidean distance $\|r - b\|$ between their endpoints. An *edge cover* of G is a subset of E such that every vertex of V is incident to at least one edge of it. The cost of an edge cover of G is defined as the sum of the costs of its edges. Then the many-to-many matching problem on R and B is equivalent to the problem of finding an edge cover of G with the minimum cost.

For a general graph with n vertices and m edges, a minimum-cost edge cover can be solved in $O(n(n \log n + m))$ time via a reduction to the minimum-cost perfect matching [8, 9, 13]. Since the complete bipartite graph G has complexity $\Theta(n^2)$, we cannot use these graph algorithms directly to obtain a subquadratic-time algorithm. Nevertheless, as our approach extensively adapts several combinatorial principles from these classical algorithms, we introduce the following graph-theoretic terminology to be used throughout this paper.

Terminology for matching. Let H be a bipartite graph with edge costs. A *matching* M in H is a set of pairwise vertex-disjoint edges, and its cost defined as the sum of its edge costs. A matching M in H is *perfect* if every vertex is incident to exactly one edge of M. A vertex of the graph is *free* (with respect to M) if no edge of M is incident to it. An *alternating path* (with respect to M) is a path in the graph that begins from a free vertex and whose edges alternately belong to M and not to M. An *augmenting path* (with respect to M) is an alternating path whose two endpoints are both free. Augmenting M along an augmenting path P by taking symmetric difference $(M \oplus P)$ increases its cardinality by

[3] The paper [20] states that their algorithm takes $O(n^{1.5+\delta} \log(n\Delta))$ time for a constant δ, but by replacing a nearest neighbor data structure used in this paper with the most recent one [12], the factor n^{δ} can be replaced with $\operatorname{polylog} n$.

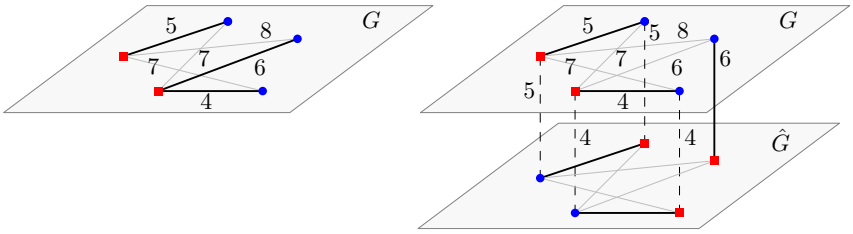

Figure 1 Illustration of the prism graph (right) of the original graph (left). The thick segments in the left figure show the min-cost many-to-many matching, and the thick segments in the right figure show the min-cost perfect matching.

one. A matching M has maximum cardinality if no augmenting path exists. Classically, maximum-cardinality (or minimum-cost) matchings are computed by repeatedly augmenting M along paths with certain properties until none remain.

2.1 Reduction to the Min-Cost Perfect Matching

In this section, we describe a reduction from the minimum-cost many-to-many matching problem to the minimum-cost perfect matching problem. While this is identical to the folklore reduction, we refer to the resulting graph as the *prism graph* for notational convenience. It preserves the original geometric embedding and induces independent vertex penalties.

Let R and B be two point sets in $[\Delta]^2$, and let G be the complete bipartite graph between R and B, where the cost of each edge is the Euclidean distance between its endpoints. The reduced instance is called the *prism graph* of G, denoted by $\tilde{G} = (\tilde{V}, \tilde{E})$, with an associated edge cost $\tilde{c} : \tilde{E} \to \mathbb{R}$.

Prism graph. The prism graph $\tilde{G}$ is a two-layered bipartite graph consisting of an "upper" layer, which is the original graph G, and a "lower" layer, which is a mirrored copy $\hat{G}$ of G. For each vertex v in the upper layer, we add a *link edge* between v and its corresponding copy $\hat{v}$ in the lower layer. Note that $\tilde{G}$ remains bipartite, as its vertex set can be partitioned into $R \cup \{\hat{v} \mid \hat{v} \text{ is the copy of } v \in B\}$ and $B \cup \{\hat{v} \mid \hat{v} \text{ is the copy of } v \in R\}$. See Figure 1. For an edge $e = (u, v)$ in $\tilde{G}$, the cost $\tilde{c}(e)$ is defined as follows:

- If u and v are both in the upper layer, $\tilde{c}(e) = \|u - v\|$.
- If u and v are both in the lower layer, $\tilde{c}(e) = 0$.
- If e is a link edge $(v, \hat{v})$, $\tilde{c}(e) = \mu(v)$, where $\mu(v)$ is the distance from v to its nearest neighbor in the opposite set (R or B).

Note that an optimal edge cover M of G consists of disjoint stars. For a star of M of size larger than two, let c be the center of the star. Then for every vertex of the star other than c, its closest neighbor in the opposite set is c. Otherwise, we can reconnect it with its closest neighbor without violating the feasibility of the solution. This reduces the problem into the minimum-cost matching problem with penalties where the penalty of each point is $\mu(\cdot)$. The construction of the prism graph is based on this intuition. Matching a vertex via a link edge to the lower layer simulates leaving that vertex uncovered by the primary matching in the upper layer, instead paying the penalty of connecting it to its nearest neighbor.

The values $\mu(v)$ for all $v \in R \cup B$ can be computed in $O(n \log n)$ time by constructing the Voronoi diagrams of R and B and using a planar point location data structure [14]. Although $\tilde{G}$ has $\Theta(n^2)$ edges in the worst case, we can store it using complexity of $O(n)$ by maintaining only the vertices and the link edges explicitly. The edges in G and $\hat{G}$ can be represented implicitly as they form complete bipartite graphs.

The following lemma establishes the equivalence between the minimum-cost edge cover of G and the minimum-cost perfect matching of $\tilde{G}$. For illustration, see Figure 1.

▶ **Lemma 3** ([3]). *Given a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{G}$, a minimum-cost edge cover of G can be computed in $O(n)$ time.*

Due to the lack of space, some proofs are omitted. The missing proofs can be found in the full version of this paper.

2.2 Overview of Our Algorithm

Due to Lemma 3, the problem reduces to computing a minimum-cost perfect matching of $\tilde{G}$. The algorithm consists of two phases.

In the first phase, we compute an approximate perfect matching of $\tilde{G}$ with its corresponding dual weights. We use the approximation algorithm of Bandyapadhyay et al. [3] which is based on the scaling algorithm [10], with dynamic additively weighted nearest-neighbor data structures [12] to obtain these dual weights in $\tilde{O}(n^{1.5} \log \Delta)$ time.

In the second phase, we leverage these dual weights to construct a sparse candidate subgraph $\tilde{G}_{\mathsf{cand}}$ of $\tilde{G}$ containing an optimal perfect matching of $\tilde{G}$. Here, a crucial property is that it is *almost* planar. Specifically, the upper layer of $\tilde{G}_{\mathsf{cand}}$ possesses a planar *skeleton*, meaning no two edges of $\tilde{G}_{\mathsf{cand}}$ cross unless their endpoints are collinear, and the lower layer of $\tilde{G}_{\mathsf{cand}}$ is a mirrored copy of its upper layer. We then compute the exact minimum-cost perfect matching of $\tilde{G}_{\mathsf{cand}}$ by applying a divide-and-conquer strategy on a balanced separator of this planar skeleton.

While this overall strategy comes from the subquadratic-time algorithm for the one-to-one setting on integer grids [20], we address two major technical hurdles unique to the many-to-many setting.

- During the construction of $\tilde{G}_{\mathsf{cand}}$ and within the recursion step, we must efficiently solve subproblems where points can either be matched to the opposite set or left unmatched by paying a penalty. To handle this, we develop an $O(n \log n)$-time algorithm for the 1D minimum-cost matching problem with penalties (Section 3), replacing the simple greedy approach used for standard 1D perfect matching.
- In the one-to-one setting [20], the candidate graph $\tilde{G}_{\mathsf{cand}}$ is strictly planar, allowing direct application of the separator-based algorithm in [17]. In our case, $\tilde{G}_{\mathsf{cand}}$ is not strictly planar. We overcome this by applying the balanced separator exclusively to the planar skeleton of $\tilde{G}_{\mathsf{cand}}$ (Section 4.2). A tricky part is to handle the vertices of $\tilde{G}_{\mathsf{cand}}$ not appearing on the planar skeleton. We address this using their 1D nature: these vertices are fully contained in the open intervals (edges) of the planar skeleton, which allows us to process them efficiently using our 1D matching subroutine.

In the following, some proofs and details are omitted due to page limits. All missing proofs and details can be found in the full version of this paper.

3 Subroutine: 1D Minimum Cost Bipartite Matching with Penalties

In this section, we prove Theorem 2, that is, we present an $O(n \log n)$-time algorithm for the 1D minimum-cost bipartite matching problem with penalties. As input, we are given two sets B and R lying on a horizontal line where each point $p \in B \cup R$ is associated with a penalty w_p. For a matching $M \subseteq B \times R$ between B and R, its cost $c(M)$ is defined as $\sum_{(r,b)\in M} \|r - b\| + \sum_{p\in(R\cup B)\setminus V(M)} w_p$. That is, its cost is the total edge length of M plus the total penalty of the unmatched vertices. The goal is to compute a matching between R and B of minimum total cost.

To the best of our knowledge, there is no work explicitly studying the minimum-cost bipartite matching problem with penalties even for the one-dimensional case. However, there are several ways to address this problem. First, this problem reduces to the min-cost flow problem on planar directed graphs. Also, if points of $B \cup R$ have integer coordinates, and the penalties w_p are all integers, then the resulting instance has integer costs. In this case, we can solve the problem in $\tilde{O}(n \log N)$ time, where N is the maximum of the maximum cost and the diameter of $B \cup R$. However, the penalty w_p can be an arbitrary real number in our case, which makes the aforementioned algorithm inapplicable in our setting. Another simple way is to use dynamic programming, which leads to an $O(n^2)$-time algorithm. Since our main algorithm is based on this dynamic programming algorithm, we also describe it here.

3.1 Quadratic-Time DP Algorithm

Let ℓ be the line containing B and R. Without loss of generality, assume that ℓ is horizontal. For a point $x \in \ell$, let B_x and R_x be the sets of points in B and R, respectively, lying to the left of x (including the points lying on x). For any point $x \in \ell$ and a positive integer k, we let $F_x(k)$ denote the minimum matching cost with penalties between $\bar{B}_x$ and R_x, where $\bar{B}_x$ is the set obtained from B_x by adding k points at x with penalty ∞. For a negative integer k, we let $F_x(k)$ denote the minimum matching cost with penalties between B_x and $\bar{R}_x$, where $\bar{R}_x$ is the set obtained from R_x by adding $|k|$ points at x with penalty ∞. Then the minimum matching cost between B and R is $F_\infty(0)$. Therefore, the problem reduces to computing $F_x(\cdot)$ for all points x in ℓ and for all values k with $-n \le k \le n$.

▶ **Lemma 4.** *For any value k and any two points x and x' of ℓ such that no point of $R \cup B$ lies between x and x' (including x and x'), we have $F_x(k) = F_{x'}(k) + |k| \cdot \|x - x'\|$.*

Thus it is sufficient to compute $F_p(k)$ for all points $p \in R \cup B$ and all values $-n \le k \le n$. But to avoid a degeneracy issue, for each point $p \in R \cup B$, we compute $F_{\bar{p}}(k)$ for a conceptual point $\bar{p}$ located infinitesimally to the right of p. For all distance calculations, we treat $\|p - \bar{p}\| = 0$. As initialization, let x_0 be any point lying strictly to the left of the first point in $B \cup R$. Then $F_{x_0}(0) = 0$ and $F_{x_0}(k) = \infty$ for any $k \neq 0$. Starting from this base case, we process the points of $R \cup B$ from left to right along ℓ. We show how to handle a point $p \in R \cup B$. Without loss of generality, assume that $p \in R$. The other case can be handled symmetrically. Let p' be the point of $B \cup R$ lying immediately to the left of p along ℓ, and let $d = \|p - p'\|$. Then we have the following recurrence relation. There are two possibilities: either p is matched with a blue point in $\bar{B}_{\bar{p}}$, or p is not used in the matching by paying the penalty. We consider both cases, and take the one with minimum cost.

$$F_{\bar{p}}(k) = \min\Big(\underbrace{F_{\bar{p}'}(k-1) + |k-1|d}_{\text{Match } p}, \quad \underbrace{F_{\bar{p}'}(k) + |k|d + w_p}_{\text{Pay Penalty}}\Big) \tag{1}$$

The recurrence relation immediately shows how to compute $F_{\bar{p}}(\cdot)$ for all points $p \in B \cup R$. Each table entry can be computed in $O(1)$ time, yielding an overall running time of $O(n^2)$.

In the following, to make the description easier, we simply let $F_p(k) := F_{\bar{p}}(k)$ as the following analysis is based only on the recurrence relation (1).

3.2 Convexity of the Cost Function

In this subsection, we show that the function $F_x(\cdot)$ is *convex* for any fixed point $x \in \ell$. We say $f : \mathbb{Z} \to \mathbb{R}$ is *convex* if $f(i-1) + f(i+1) \ge 2f(i)$ for every integer i. If $f(i+1) = \infty$ or $f(i-1) = \infty$, the inequality holds immediately. We let $\mathrm{dom}(f)$ be the set of integers k such that $f(k)$ is finite. Let $\Delta f(k) := f(k) - f(k-1)$.

▶ **Lemma 5.** *The function $F_x(\cdot)$ is convex for any fixed point $x \in R \cup B$.*

Proof. We use induction on the points of $B \cup R$ encountered from left to right. Let x_0 be a point to the left of the leftmost point of $B \cup R$. We define $F_{x_0}(0) = 0$ and $F_{x_0}(k) = \infty$ for $k \neq 0$, which is trivially convex.

Let $p \in B \cup R$ be the current point and p' be the previous point in $B \cup R \cup \{x_0\}$. Let $d = \|p - p'\|$. Without loss of generality, assume that $p \in B$. By the induction hypothesis, $F_{p'}(\cdot)$ is convex. Note that $|k-1|d$ is a convex function of k. Since the sum of two convex functions is also convex, $F_{p'}(k-1) + |k-1|d$ is convex. Similarly, $F_{p'}(k) + |k|d + w_p$ is convex. Although the minimum of two convex functions is not convex in general, it is convex in our setting. This holds due to the following claim with $f(k) = F_{p'}(k) + |k|d$.

▷ Claim 6. Let $f : \mathbb{Z} \to \mathbb{R}$ be a convex function. Then $F_p(k) := \min\{f(k-1), f(k) + w\}$ is also convex for any value w.

Therefore, the lemma holds for any fixed point $x \in R \cup B$. ◀

As we showed in the proof of Claim 6, the following corollary holds.

▶ **Corollary 7.** *Let k^* be the largest integer with $\Delta f(k) \leq -w_p$. Then for $k \leq k^*$, we have $F_p(k) = f(k) + w_p$, and for $k > k^*$, we have $F_p(k) = f(k-1)$.*

3.3 Near-Linear-Time Algorithm

In this subsection, we present an $O(n \log n)$-time algorithm for the 1D minimum-cost bipartite matching problem with penalties. For each $p \in B \cup R$, let $\mathrm{dom}(F_p) = [L_p, R_p]$ be the maximal interval of integers for which $F_p(k) < \infty$. By the recurrence, $\mathrm{dom}(F_x)$ is always a contiguous interval. Basically, we compute $F_p(k)$ for all integers $k \in \mathrm{dom}(F_p)$ and all points $p \in B \cup R$. By maintaining $F_p(k)$ using a data structure for a fixed p, we show that $F_p(k)$ can be computed in $O(\log n)$ time for all integers k for a fixed p assuming that we have $F_{p'}(\cdot)$, where p' is the point of $B \cup R$ lying immediately to the left of p.

Representation of the cost function. Fix $x \in \ell$, and let $\Delta F_x(k) := F_x(k) - F_x(k-1)$. We maintain $F_x(0)$ and $\Delta F_x(k)$ for all indices k. Note that they fully represent $F_x(\cdot)$. To make the update efficiently, we store $\Delta F_x(\cdot)$ using two balanced binary search trees. Let H_x^+ and H_x^- be two sets where

$$H_x^+ = \{\Delta F_x(k) \mid 0 < k \leq R_x\} \text{ and } H_x^- = \{\Delta F_x(k) \mid L_x < k \leq 0\}.$$

Since $F_x(\cdot)$ is convex, $\Delta F_x(k)$ is non-decreasing with respect to k. We maintain the two sets using two binary search trees T_x^+ and T_x^- and offsets o^+ and o^-. The offsets allow us to shift all stored values uniformly without updating each element explicitly. Each leaf of a tree of T_x^+ (and T_x^-) corresponds to exactly one index $k > 0$ (and $k < 0$), and the leaf stores the value $\Delta F_x(k) - o^+$ (and $\Delta F_x(k) - o^-$). Thus, the actual value of $\Delta F_x(k)$ is obtained by adding the corresponding offset o^+ or o^-. Each internal node additionally stores the size of its subtree and the sum of the stored values in its subtree (excluding the offset). Hence, the true sum of the values in any subtree can be obtained by adding the offset multiplied by the subtree size. The leaves of T_x^+ (and T_x^-) are stored in increasing order of their indices k. Since $F_x(\cdot)$ is convex, the stored values are non-decreasing along this order. Therefore, T_x^+ and T_x^- can be implemented as binary search trees augmented with subtree sizes and subtree sums, supporting insertion, deletion, and prefix-sum queries in $O(\log n)$ time. We call the tuple of $F_x(0)$, o^+, o^-, T_x^+ and T_x^- the *representation* of $F_x(\cdot)$. Here, since the roots store the size of the trees, we do not need to explicitly maintain $\mathrm{dom}(F_p)$.

▶ **Observation 8.** *Given the representation of $F_x(\cdot)$, we can compute $F_x(k)$ for any integer k in $O(\log n)$ time.*

Proof. We show how to compute $F_x(k)$ for any integer $k > 0$. For $k = 0$, we maintain $F_x(0)$ explicitly, and for $k < 0$, we can do this symmetrically. By telescoping, we have

$$F_x(k) = F_x(0) + \sum_{i=1}^{k} \Delta F_x(i).$$

Thus, it suffices to compute the prefix sum of the first k values in H_x^+.

Since $\Delta F_x(k)$ is non-decreasing in k, the values stored in T_x^+ are already ordered by index. We augment each node of T_x^+ with the size of its subtree and the sum of the stored values in its subtree (excluding the offset o^+). Using standard binary search tree operations, we can retrieve the sum of the smallest k elements in T_x^+ in $O(\log n)$ time. Adding this sum and the accumulated offset $k \cdot o^+$ to $F_x(0)$ yields $F_x(k)$. Therefore, given the representation of $F_x(\cdot)$, we can compute $F_x(k)$ for any integer k in $O(\log n)$ time. ◀

Update of the representation. Assume that we are given the representation of $F_{p'}(\cdot)$, and we wish to compute the representation of $F_p(\cdot)$, where p is the next point to the right of p' on ℓ. We describe the case $p \in B$; the case $p \in R$ is symmetric. Let $d := \|p - p'\|$.

Step 1. We first consider the transportation cost. Let $f(k) := F_{p'}(k) + |k|d$. Thus, before handling the matching/penalty decision at p, we first transform the representation of $F_{p'}$ into the representation of f. For $k > 0$, we have $\Delta f(k) = \Delta F_{p'}(k) + d$, and for $k \leq 0$, we have $\Delta f(k) = \Delta F_{p'}(k) - d$. Hence, this update can be performed by increasing the offset o^+ by d, and decreasing the offset o^- by d, while keeping the trees unchanged. Also, $F_{p'}(0)$ remains unchanged since $|0|d = 0$. After this step, the representation corresponds to $f(\cdot)$.

Step 2. We then find the largest integer k^* with $\Delta f(k) \leq -w_p$ by a binary search over the ordered trees $T_{p'}^+$ and $T_{p'}^-$ in $O(\log n)$ time. Since $\Delta f(k)$ is non-decreasing in k, the threshold k^* can be found by searching for the largest index whose discrete derivative is at most $-w_p$. Recall that $F_p(k) = \min\{f(k-1),\, f(k) + w_p\}$ from (1). By Corollary 7, for $k \leq k^*$, we have $F_p(k) = f(k) + w_p$, and for $k > k^*$, we have $F_p(k) = f(k-1)$.

Step 3. We now update the representation of $f(\cdot)$ to obtain that of $F_p(\cdot)$. We first compute $F_p(0)$. Since $F_p(0) = \min\{f(-1),\, f(0) + w_p\}$, this value can be computed in $O(\log n)$ time using Observation 8.

We next determine the new discrete derivatives. For $k \leq k^*$, both $F_p(k)$ and $F_p(k-1)$ lie in the same case, and hence

$$\Delta F_p(k) = F_p(k) - F_p(k-1) = f(k) - f(k-1) = \Delta f(k).$$

For $k > k^* + 1$, we have

$$\Delta F_p(k) = F_p(k) - F_p(k-1) = f(k-1) - f(k-2) = \Delta f(k-1).$$

At the boundary $k = k^* + 1$, we have

$$\Delta F_p(k^* + 1) = F_p(k^* + 1) - F_p(k^*) = f(k^*) - (f(k^*) + w_p) = -w_p.$$

Therefore, the sequence $\Delta F_p(\cdot)$ is obtained from $\Delta f(\cdot)$ by inserting the single value $-w_p$ (minus the corresponding offset) at position $k^* + 1$. As a result, all indices strictly larger than k^* are shifted by one position to the right.

Since the discrete derivatives are stored in a binary search tree, this transformation can be implemented by a single rank-based insertion at index $k^* + 1$. If $k^* < 0$, the insertion shifts the element originally at index 0 to index 1. Because index 0 belongs to $T^-_{p'}$ and index 1 belongs to $T^+_{p'}$, we must extract the maximum element from $T^-_{p'}$ and insert it as the minimum element in $T^+_{p'}$ (after adjusting the values with respect to the offsets for the two trees). All subtree sizes and subtree sums are updated along the search path. The insertion and any necessary boundary shifts take $O(\log n)$ time. Thus, the entire update from the representation of $F_{p'}(\cdot)$ to that of $F_p(\cdot)$ takes $O(\log n)$ time.

▶ **Lemma 9.** *Given the representation of $F_{p'}(\cdot)$, we can compute the representation of $F_p(\cdot)$ in $O(\log n)$ time.*

We consider the points of $B \cup R$ from left to right. The initialization takes $O(n)$ time, and each update takes $O(\log n)$ time. Therefore, the total running time is $O(n \log n)$.

4 Minimum-Cost Perfect Matching for Prism Graphs

In this section, we present an exact algorithm for computing a minimum-cost perfect matching of the prism graph $\tilde{G}$. To simplify the exposition, we identify the vertices in the *upper layer* of the prism graph $\tilde{G}$ directly with the original points in R and B. Specifically, we let the upper layer vertex set be $R \cup B$. We then define the *lower layer* as a mirrored copy of the upper layer, where each vertex $v \in R \cup B$ has a corresponding copy $\hat{v}$ in the lower layer.

Our strategy involves two main phases: computing an approximate minimum-cost perfect matching of $\tilde{G}$ along with its associated dual weights, and utilizing these dual weights to derive the exact solution. We begin by establishing the properties of an approximate solution and the associated dual weights.

1-optimality. Given a *scaling factor* $\theta > 0$, let $\tilde{G}_\theta = (\tilde{V}, \tilde{E})$ denote the *θ-scaled graph* of $\tilde{G}$ with the edge-cost function $\tilde{c}_\theta : \tilde{E} \to \mathbb{Z}_{\geq 0}$, defined as $\tilde{c}_\theta(e) = \lceil \tilde{c}(e)/\theta \rceil$ for all $e \in \tilde{E}$. Let $y(v)$ denote the *dual weight* of a vertex $v \in \tilde{V}$. A matching $\tilde{M}$ in $\tilde{G}_\theta$ and the associated dual weights $y(\cdot)$ are said to be *1-feasible* if

$$y(u) + y(v) \leq \tilde{c}_\theta(e) + 1 \quad \text{for all } e = (u, v) \in \tilde{E}, \tag{2}$$

$$y(u) + y(v) = \tilde{c}_\theta(e) \quad \text{for all } e = (u, v) \in \tilde{M}. \tag{3}$$

We refer to (2) and (3) as the *1-feasibility conditions*. An edge $e = (u, v) \in \tilde{E} \setminus \tilde{M}$ is called *admissible* if $y(u) + y(v) = \tilde{c}_\theta(e) + 1$. The *admissible graph* of $\tilde{G}_\theta$ is the subgraph of $\tilde{G}_\theta$ consisting of all admissible edges and the edges in $\tilde{M}$. If a matching $\tilde{M}$ and dual weights $y(\cdot)$ satisfy the 1-feasibility conditions and $\tilde{M}$ is a perfect matching, we call the pair $(\tilde{M}, y)$ *1-optimal*. Sharathkumar and Agarwal [21] showed that a 1-optimal matching of $\tilde{G}_\theta$ provides an approximate minimum-cost perfect matching of $\tilde{G}$ with an additive error ε of at most $3n\theta$.

We now outline our algorithm for computing an exact minimum-cost perfect matching of $\tilde{G}$. First, we compute a 1-optimal matching $\tilde{M}^*_1$ and associated dual weights $y(\cdot)$ of $\tilde{G}_\theta$ using a scaling factor of $\theta = 1/(n\Delta^{33})$, following the procedure described in the full version of this paper. In Section 4.1, we utilize these dual weights to identify a set of *eligible edges* E_{elig} of the upper layer of $\tilde{G}$ and exploit their geometric properties to construct a set P of non-crossing line segments having endpoints at $B \cup R$. Using P, we systematically extract a

sparse candidate subgraph $\tilde{G}_{\mathsf{cand}}$ of size $O(n)$ that is guaranteed to contain a minimum-cost perfect matching of $\tilde{G}$. Finally, Section 4.2 presents a divide-and-conquer algorithm to compute an exact optimal solution in $\tilde{G}_{\mathsf{cand}}$ by applying the planar separator theorem to the graph induced by P.

4.1 Construction of the Candidate Subgraph

We construct a *candidate subgraph* $\tilde{G}_{\mathsf{cand}} = (\tilde{V}, \tilde{E}_{\mathsf{cand}})$ of $\tilde{G}$ that is guaranteed to contain an optimal matching of $\tilde{G}$. We begin by utilizing the 1-optimal matching $\tilde{M}_1^*$ and the 1-optimal dual weights of $\tilde{G}_\theta$ with a scaling factor of $\theta = 1/(n\Delta^{33})$ to characterize the *eligible edges* in the upper layer that can participate in an optimal solution. This characterization reveals a crucial geometric property of these edges. Using this property, we apply the technique of Sharathkumar [20] to construct a planar skeleton (planar straight-line graph) P. Finally, we use P to generate the set of *candidate edges* $\tilde{E}_{\mathsf{cand}}$.

The overall construction comes from Sharathkumar [20]: It applies the approach to a Euclidean complete bipartite graph while we apply it to the upper layer of the prism graph $\tilde{G}$. As the upper layer is a Euclidean complete bipartite graph, the analysis immediately follows from [20], but the detailed implementation should be tailored for our setting.

Eligible edge decomposition and its induced planar skeleton. We define the *scaled dual weight* of a vertex $v \in \tilde{V}$ as $y_\theta(v) = \theta \cdot y(v)$. The following lemma establishes a lower bound on the sum of scaled dual weights for the edges participating in an optimal matching of $\tilde{G}$.

▶ **Lemma 10.** *Let $y_\theta(\cdot)$ be the scaled dual weights associated with a 1-optimal matching of $\tilde{G}_\theta$ using the scaling factor $\theta = 1/(n\Delta^{33})$. Let $\tilde{M}^*$ be a minimum-cost perfect matching of $\tilde{G}$. Then, for every edge $(u, v) \in \tilde{M}^*$ in the upper layer,*

$$y_\theta(u) + y_\theta(v) > \|u - v\| - \frac{2}{\Delta^{33}}. \tag{4}$$

The proof is same as the argument in [20] and is thus omitted. We define the set of *eligible edges*, denoted by E_{elig}, as the set of all edges in the upper layer of $\tilde{G}$ satisfying the condition (4). Lemma 10 implies that for any optimal matching $\tilde{M}^*$ of $\tilde{G}$, all edges of $\tilde{M}^*$ that belong to the upper layer are contained in E_{elig}. Therefore, even if we remove all non-eligible edges in the upper layer from the prism graph, the optimal solution remains the same.

The following lemma shows that if no four vertices of $R \cup B$ are collinear, then the graph obtained from the upper layer of the prism graph by removing all non-eligible edges is planar. Thus our main focus is to handle collinear points in the following.

▶ **Lemma 11** ([20])**.** *Let $Q \subset [\Delta]^2$ be a set of points. Let $a, b, c, d \in Q$ be four distinct points that are not collinear. If the line segments $\overline{ab}$ and $\overline{cd}$ intersect, then*

$$\|a - d\| + \|b - c\| + \frac{1}{\Delta^{32}} < \|a - b\| + \|c - d\|.$$

Therefore, for any two distinct edges (a, b) and (c, d) in E_{elig} such that their four endpoints are not collinear, the line segments $\overline{ab}$ and $\overline{cd}$ do not intersect.

We show that the non-crossing property enables us to compute an *eligible edge decomposition* $P = \{(p_1, q_1), \ldots, (p_k, q_k)\}$, which is a set of non-crossing line segments, such that for every edge of E_{elig}, it is contained in a segment of P. Note that P induces a plane graph whose vertices come from $R \cup B$.

Since the definition of eligible edges in our setting is identical to that in Sharathkumar [20], we can adopt their algorithm for computing the eligible edge decomposition. However, a minor technical hurdle arises in the initialization phase. It uses an approximate *perfect* matching M obtained from the first phase, but in our setting, an approximate matching we have is not necessarily perfect if we look at the upper layer only. More specifically, it first computes all point-line pairs (p, ℓ) with $p \in R \cup B$ such that ℓ contains an eligible edge incident to p. The number of such pairs is $O(n)$. For a point $p \in R \cup B$, one can compute all eligible edges incident to it in $O(\text{polylog}\, n)$ time per edge using a dynamic weighted nearest neighbor data structure. However, it is possible that a single line ℓ contains many different eligible edges incident to p. Then a single line can be discovered multiple times, which leads to quadratic running time in the worst case. To avoid this, as initialization, it computes just one line ℓ for each point p in $R \cup B$ containing an eligible edge incident to p in $O(n \,\text{polylog}\, n)$ time in total using M. Later, it considers each point p in $B \cup R$ one by one and computes all pairs (p, ℓ) such that ℓ is not found in the initialization. Even in this case, a single line ℓ can be discovered multiple times, but the total number of pairs (p, ℓ) found in this way is $O(n)$, which leads to near-linear running time. The perfect matching M can be used in the initialization; since an edge (u, v) of M is eligible, its extension contains an eligible edge incident to u (and v). But we can perform this initialization using a dynamic weighted nearest-neighbor data structure, and this does not increase the overall running time. The full details of this modified construction are deferred to the full version of this paper.

Construction of the candidate subgraph. We now construct the candidate edge set $\tilde{E}_{\mathsf{cand}}$ of size $O(n)$ using the eligible edge decomposition P. As mentioned earlier, if no four points of $R \cup B$ are collinear, we can simply construct $\tilde{E}_{\mathsf{cand}}$ by adding all eligible edges (the edges of P), their mirrored edges in the lower layer, and all link edges between the layers. However, in the general case, the number of eligible edges can be $\Theta(n^2)$, making it impossible to include all of them.

Instead, we select a restricted subset of eligible edges as follows. For each pair $(p, q) \in P$, let I be the set of points in $R \cup B$ lying on the open segment between p and q. For a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{G}$, any edge incident to a vertex $v \in I$ must have its other endpoint in $I \cup \{\hat{v}, p, q\}$. Consequently, for the endpoints p and q, there are four possible configurations based on whether p and q are matched with points in I. For each configuration, we can identify the specific edges of $\tilde{M}^*$ incident to points in I in $O(|I| \log |I|)$ time without computing the entire matching. More specifically, let $I' \subseteq \{p, q\}$ be the set of points matched with elements in I under $\tilde{M}^*$. By solving the local matching problem for $I \cup I'$, we extract the necessary edges to be included in $\tilde{E}_{\mathsf{cand}}$, ensuring the total size remains $O(n)$. For the local matching problem, since the vertices of $I \cup I'$ are collinear, the problem is restricted to one dimension. This problem reduces to the minimum-cost matching problem on $I \cup I'$, where leaving a vertex unmatched incurs a penalty of $\mu'(\cdot)$, where $\mu'(v) = \mu(v)$ for a vertex $v \in I$ and $\mu'(v) = \infty$ for a vertex $v \in I'$. Therefore, the local matching problem can be solved in $O(|I| \log |I|)$ time by Theorem 2. We do this for all the four configurations, and then take the union of the resulting edges. Then $\tilde{E}_{\mathsf{cand}}$ is defined as the union of all such edges in P, their mirrored edges in the lower layer, and all the link edges. Then the size of $\tilde{E}_{\mathsf{cand}}$ is $O(n)$, and it can be computed in $O(n \log n)$ time.

We show that $\tilde{E}_{\mathsf{cand}}$ contains a minimum-cost perfect matching of $\tilde{G}$. For this, we use the following technical lemma justifying that focusing on the upper layer during the construction of $\tilde{E}_{\mathsf{cand}}$ is sufficient. We say a matching M is *symmetric* if an edge (u, v) in the upper layer is contained in M if and only if its mirrored edge $(\hat{u}, \hat{v})$ is contained in M.

▶ **Lemma 12.** *Let $\tilde{H}$ be a subgraph of $\tilde{G}$ induced by edges in the upper layer, their mirrored edges in the lower layer, and all link edges. There exists a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{H}$ that is symmetric.*

Proof. Let $\tilde{M}^*$ be an arbitrary minimum-cost perfect matching of $\tilde{H}$. Let m be the number of edges in $\tilde{M}^*$ that belong to the upper layer. Each such edge (u, v) covers one vertex in R and one in B. Because $\tilde{M}^*$ is a perfect matching, any vertex in the upper layer not matched with a vertex in the upper layer must be matched to its corresponding copy in the lower layer via a link edge. This implies that exactly m vertices in the lower layer are not covered by link edges. To satisfy the perfect matching requirement, these m vertices must be matched to each other using $m/2$ edges in the lower layer.

Now, consider the cost of these $m/2$ edges. By construction, the cost of any edge in the lower layer is zero. Therefore, we can replace the existing lower-layer edges in $\tilde{M}^*$ with the "mirrored" edges of the upper-layer matching without changing the total cost of the matching. This substitution results in a new perfect matching M' that remains optimal and satisfies the symmetry condition: (u, v) is in M' if and only if $(\hat{u}, \hat{v})$ is in M'. Therefore, the lemma holds. ◀

▶ **Lemma 13.** *There exists a minimum-cost perfect matching $\tilde{M}^*$ of $\tilde{G}$ such that $\tilde{M}^* \subseteq \tilde{E}_{\mathsf{cand}}$.*

Proof. Let $\tilde{M}^*$ be a minimum-cost perfect matching of $\tilde{G}$ such that (i) all edges of $\tilde{M}^*$ belonging to the upper layer are contained in E_{elig}, and (ii) $\tilde{M}^*$ is symmetric, which always exists by Lemma 10 and by Lemma 12.

To show that $\tilde{M}^* \subseteq \tilde{E}_{\mathsf{cand}}$, we consider the eligible edge decomposition P. Every edge $(u, v) \in \tilde{M}^*$ in the upper layer must be contained within some segment $\overline{pq} \in P$. For each such segment, let I be the set of points of $B \cup R$ lying in the open interval between p and q. By the construction of P, any edge in $\tilde{M}^*$ incident to a vertex in I must have its other endpoint in $I \cup \{\hat{v}, p, q\}$. This local constraint implies that the matching within each segment $\overline{pq}$ can be optimized independently based on the four possible configurations of the endpoints p and q. Since $\tilde{E}_{\mathsf{cand}}$ is constructed by taking the union of the optimal local matchings for all four configurations across all segments in P, it follows that the edges of $\tilde{M}^*$ are included in $\tilde{E}_{\mathsf{cand}}$. Thus, the subgraph of $\tilde{G}$ induced by $\tilde{E}_{\mathsf{cand}}$ contains a minimum-cost perfect matching of $\tilde{G}$. ◀

Therefore, the candidate edge set $\tilde{E}_{\mathsf{cand}}$ can be computed in $\tilde{O}(n)$ time in total, and the problem reduces to computing a minimum-cost perfect matching on the subgraph $\tilde{G}_{\mathsf{cand}}$ of $\tilde{G}$ induced by $\tilde{E}_{\mathsf{cand}}$. For an illustration of $\tilde{G}_{\mathsf{cand}}$, see Figure 2.

It is worth noting the structural distinction between our construction and the one-to-one setting addressed by Sharathkumar [20]. In the one-to-one case, for each segment $(p, q) \in P$, it is sufficient to consider at most *two* configurations of the endpoints p and q, which makes the resulting graph planar. This is because the difference between the numbers of points in R and of points in B restricts the possible configurations of p and q. In the many-to-many setting, however, this difference does not inherently reduce the number of configurations. Consequently, we must consider the union of local optimal matchings for all *four* configurations, which renders the resulting candidate graph $\tilde{G}_{\mathsf{cand}}$ non-planar.

4.2 Divide-and-Conquer Algorithm via Planar Separators

In this section, we present a divide-and-conquer algorithm to compute a minimum-cost perfect matching of the candidate graph $\tilde{G}_{\mathsf{cand}} = (\tilde{V}, \tilde{E}_{\mathsf{cand}})$. Our algorithm recursively partitions $\tilde{G}_{\mathsf{cand}}$ into disjoint vertex sets by removing a subset of vertices. We first compute

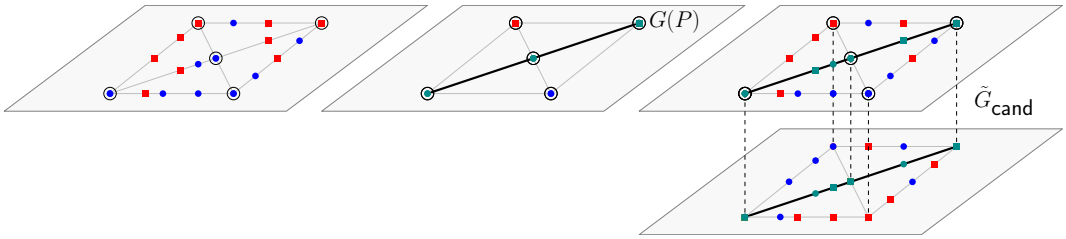

Figure 2 Illustration of the upper layer of $\tilde{G}_{\mathsf{cand}}$ (left), the planar skeleton $G(P)$ (middle), and the candidate subgraph $\tilde{G}_{\mathsf{cand}}$ (right). Red squares and blue disks denote vertices in $R \cup \hat{B}$ and vertices in $B \cup \hat{R}$, respectively. In the upper layer of $\tilde{G}_{\mathsf{cand}}$, the segment endpoints represent skeleton vertices, while intermediate points on the segments represent interior vertices. The green vertices in $G(P)$ constitute the balanced separator of $G(P)$, and those in $\tilde{G}_{\mathsf{cand}}$ form the prism separator $\tilde{S}$.

the optimal matching for the subgraph induced by the disjoint sets. Subsequently, in the conquer phase, we reintroduce the removed vertices and gradually update the matching to obtain a minimum-cost perfect matching of $\tilde{G}_{\mathsf{cand}}$ using augmenting paths.

We begin by recalling the planar separator theorem, which serves as the foundation for our graph partitioning strategy. Lipton and Tarjan [16] established the following result for planar graphs.

▶ **Theorem 14** ([16]). *Let H be a planar graph with n_H vertices. The vertices of H can be partitioned in $O(n_H)$ time into three disjoint sets X, Y, S such that (i) no edge connects a vertex in X with a vertex in Y, (ii) $|X|, |Y| \leq 2n_H/3$, and (iii) $|S| \leq 2\sqrt{2}\sqrt{n_H}$.*

We call S a *balanced separator* of H. While $\tilde{G}_{\mathsf{cand}}$ is not necessarily planar, its inherent planar structure is derived from the eligible edge decomposition P. Let $V(P)$ be the set of endpoints of the edges (segments) in P, and let $G(P) = (V(P), P)$ be the planar skeleton defined by P, that is, it is the planar graph induced by the edges of P.

We apply the planar separator theorem to $G(P)$ to induce a separation of $\tilde{G}_{\mathsf{cand}}$. Let X, Y, S be a partition of $V(P)$ obtained by applying Theorem 14 to $G(P)$, where S is a balanced separator of $G(P)$. Let $\hat{X}, \hat{Y}$ and $\hat{S}$ denote the mirrored sets of X, Y and S, respectively, in the lower layer. We make the following observation regarding the sizes of these sets.

▶ **Observation 15.** *Let $\hat{V}(P)$ be the mirrored set of $V(P)$ in the lower layer. Then, $|X \cup \hat{X}|, |Y \cup \hat{Y}| \leq 2/3 \cdot (|V(P)| + |\hat{V}(P)|)$, and $|S \cup \hat{S}| \leq 2\sqrt{2}(|V(P)| + |\hat{V}(P)|)^{1/2}$.*

Using the partition (X, Y, S) of $V(P)$, we partition the vertex set of $\tilde{G}_{\mathsf{cand}}$ into $\tilde{X}, \tilde{Y}$ and $\tilde{S}$ with $X \subseteq \tilde{X}, Y \subseteq \tilde{Y}$ and $S \subseteq \tilde{S}$ as follows. For this, it suffices to partition the vertices of $\tilde{G}_{\mathsf{cand}}$ contained in the open segments of P. We call such vertices *interior vertices*. For each interior vertex v of $\tilde{G}_{\mathsf{cand}}$, we assign it to the set based on the endpoints of the segment of P containing v. Specifically, if the segment of P containing v has an endpoint in X (in Y), we put v and $\hat{v}$ to $\tilde{X}$ (to $\tilde{Y}$). Otherwise, both endpoints of the segment are contained in S. In this case, we put v and $\hat{v}$ to $\tilde{S}$. Here, notice that no segment of P has endpoints both in X and Y since S separates X and Y.

▶ **Lemma 16.** *No edge in $\tilde{E}_{\mathsf{cand}}$ connects a vertex in $\tilde{X}$ to a vertex in $\tilde{Y}$.*

Proof. We prove this by examining the three types of edges e in $\tilde{E}_{\mathsf{cand}}$. In the first case that e belongs to the upper layer, every eligible edge in the upper layer is contained in a segment s in P. If an endpoint of s is contained in X (or in Y), both endpoints of e are in $\tilde{X}$ (or $\tilde{Y}$), or one belongs to $\tilde{X}$ (or $\tilde{Y}$) and the other to $\tilde{S}$. If both endpoints of s are contained in S, both endpoints of e are contained in $\tilde{S}$. Thus in any case, the lemma holds.

In the other cases, the lemma holds due to the symmetric construction of $\tilde{E}_{\mathsf{cand}}$ following Lemma 12. Specifically, a vertex $v \in B \cup R$ and its copy $\hat{v}$ belong to the same set among $\tilde{X}, \tilde{Y}$ and $\tilde{S}$. Therefore, in the case that e belongs to the lower layer, or e is a link edge, the lemma holds immediately. ◀

We refer to $\tilde{S}$ as the *prism separator* of $\tilde{G}_{\mathsf{cand}}$. We classify each vertex of the prism separator $\tilde{S}$ into two types. We call a vertex of $S \cup \hat{S}$ a *skeleton vertex* of $\tilde{S}$ and a vertex of $\tilde{S} \setminus (S \cup \hat{S})$ an *interior vertex* of $\tilde{S}$. See Figure 2.

Divide-and-Conquer Algorithm. We now present a detailed divide-and-conquer algorithm for computing a minimum-cost perfect matching of $\tilde{G}_{\mathsf{cand}}$. The algorithm first computes the partition $(\tilde{X}, \tilde{Y}, \tilde{S})$ of the vertex set in linear time and recursively finds minimum-cost perfect matchings for the subgraphs induced by $\tilde{X}$ and $\tilde{Y}$. Note that the subgraph induced by $\tilde{X}$ (and $\tilde{Y}$) has a perfect matching due to its prism structure. Since no edges connect $\tilde{X}$ and $\tilde{Y}$, the union of their optimal matchings is a minimum-cost perfect matching for the subgraph induced by $\tilde{X} \cup \tilde{Y}$. To obtain the optimal matching for the entire graph, one might consider extending this matching by adding all vertices of the separator $\tilde{S}$ one by one using the following lemma. Here, a *minimum-cost maximum-cardinality matching* is defined as the matching with the minimum total cost among all matchings of maximum possible cardinality.

▶ **Lemma 17** ([17]). *Let H be a graph with n_H vertices and m_H edges associated with edge costs, and let v be a vertex of H. Given a minimum-cost maximum-cardinality matching of $H - \{v\}$, a minimum-cost maximum-cardinality matching of H can be computed in $O(m_H \log n_H)$ time.*

However, a naive application of Lemma 17 to all vertices in $\tilde{S}$ would take $O(|\tilde{S}| \cdot n \log n)$ time, which is inefficient since $|\tilde{S}|$ can be as large as $\Theta(n)$. To address this, we observe that $\tilde{S}$ consists of $O(\sqrt{n})$ *skeleton vertices* and a potentially large number of *interior vertices*. Our strategy is to handle the skeleton vertices using Lemma 17, while the interior vertices are handled efficiently by exploiting their 1D nature and pre-computed optimal matchings.

The algorithm processes the subgraph $\tilde{G}'$ of $\tilde{G}_{\mathsf{cand}}$ in each recursive step as follows. Using the subgraph of $G(P)$ induced by $V(P) \cap V(\tilde{G}')$, we compute a balanced separator S and a partition $(\tilde{X}, \tilde{Y}, \tilde{S})$ of $\tilde{G}'$. As mentioned earlier, we compute a minimum-cost perfect matching of $\tilde{X} \cup \tilde{Y}$ in linear time. Now consider the internal vertices of $\tilde{S}$. We decompose them into subsets such that each subset consists of the internal vertices lying on the same segment of P. For each subset W, we retrieve its optimal matching of $W \cup \hat{W}$ using the pre-computed 1D matching from Section 4.1, where $\hat{W}$ is the copy of W in the lower layer. An optimal matching of $W \cup \hat{W}$ can be obtained from a minimum-cost matching of W with penalties, which can be found in near-linear time. Since no edge of $\tilde{G}'$ connects two vertices from different subsets, the union M' of the all these matchings is a minimum-cost maximum-cardinality matching of $\tilde{X} \cup \tilde{Y} \cup \{\text{internal vertices of } \tilde{S}\}$. Finally, we reintroduce the $O(\sqrt{n'})$ skeleton vertices of $\tilde{S}$ into the matching using Lemma 17, where $n' = |V(\tilde{G}') \cap V(P)|$. We iteratively apply Lemma 17 for each skeleton vertex of $\tilde{S}$ to update $\tilde{M}'$. In this way, each recursion step takes $\tilde{O}(|V(\tilde{G}')|\sqrt{n'})$ time in total.

In the base case that the number of (skeleton) vertices of $V(P) \cap V(\tilde{G}')$ is $O(1)$, we compute a minimum-cost maximum-cardinality matching of the (non-skeleton) vertices of $V(\tilde{G}') \setminus V(P)$ using their 1D nature in $\tilde{O}(|V(\tilde{G}')|)$ time. Then we reintroduce the $O(1)$ skeleton vertices with Lemma 17. This takes $\tilde{O}(|V(\tilde{G}')|)$ time in total.

Complexity Analysis. To bound the overall time complexity, let $\eta = O(n)$ be the number of vertices of $V(P)$. At depth d of the recursion, let $H_{d,1}, H_{d,2}, \dots$ denote the disjoint subgraphs, where each $H_{d,i}$ contains $m_{d,i}$ edges and $\eta_{d,i}$ skeleton vertices along with their copies. By Theorem 14 and Observation 15, the number of skeleton vertices shrinks by a factor of at least 2/3 per level, yielding $\eta_{d,i} \leq (2/3)^d \eta$. Furthermore, since the subgraphs at any depth are edge-disjoint, $\sum_i m_{d,i} \leq |\tilde{E}_{\mathsf{cand}}| = O(n)$.

In the conquer phase, merging the sub-solutions for $H_{d,i}$ requires $O(\sqrt{\eta_{d,i}})$ augmentations. Since each augmentation takes $O(m_{d,i} \log n)$ time by Lemma 17, the total time for depth d is:

$$\sum_i O\left(\sqrt{\eta_{d,i}} \cdot m_{d,i} \log n\right) \leq O\left(\sqrt{(2/3)^d \eta} \log n\right) \sum_i m_{d,i} \leq O\left(\left(\sqrt{2/3}\right)^d \cdot n^{1.5} \log n\right).$$

Because $\sqrt{2/3} < 1$, the time for each level decreases geometrically. The total running time of the divide-and-conquer algorithm is thus dominated by the root level, which is $O(n^{1.5} \log n)$.

Overall complexity and proof of Theorem 1. We compute an approximate matching and its dual weights for the scaled graph with a suitable scaling factor in $\tilde{O}(n^{1.5} \log \Delta)$ time. The candidate graph can be computed in $O(n \log n)$ time, and the divide-and-conquer algorithm takes $O(n^{1.5} \log n)$ time. Therefore, the total time complexity of our algorithm remains $\tilde{O}(n^{1.5} \log \Delta)$.

5 Conclusion

In this paper, we presented an exact $\tilde{O}(n^{1.5} \log \Delta)$-time algorithm for the many-to-many matching problem on planar point sets with integer coordinates. Our approach relies on a reduction to perfect matching on a prism graph, a scaling framework, and a planar separator-based divide-and-conquer strategy.

Extending this framework to non-integer coordinates or higher dimensions ($d \geq 3$) poses fundamental challenges. In particular, our framework relies on Lemma 11. In continuous domains, the additive error in the inequality in Lemma 11 is no longer bounded, and thus it seems hard to apply our framework to continuous domains. For higher dimensions, geometric graphs lack planarity, preventing the use of a balanced vertex separator of $O(\sqrt{n})$ size. Moreover, in higher dimensions, maintaining the dynamic weighted nearest-neighbor data structures required in the scaling step becomes substantially more difficult. We leave the problem of designing a subquadratic-time algorithm for many-to-many matching on higher dimensions or continuous domains as an open problem.

References

1 Helmut Alt, Kurt Mehlhorn, Hubert Wagener, and Emo Welzl. Congruence, similarity, and symmetries of geometric objects. *Discrete & Computational Geometry*, 3(3):237–256, 1988. doi:10.1007/BF02187910.

2 Shinwoo An, Eunjin Oh, and Jie Xue. Approximation algorithms for the geometric multimatching problem. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC 2025)*, pages 2317–2328, 2025. doi:10.1145/3717823.3718147.

3 Sayan Bandyapadhyay, Anil Maheshwari, and Michiel Smid. Exact and approximation algorithms for many-to-many point matching in the plane. In *Proceedings on the 32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *LIPIcs*, pages 44:1–44:14, 2021. doi:10.4230/LIPIcs.ISAAC.2021.44.

4 Sayan Bandyapadhyay and Jie Xue. An $O(n \log n)$-time approximation scheme for geometric many-to-many matching. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG 2024)*, pages 12:1–12:15, 2024. doi:10.4230/LIPIcs.SOCG.2024.12.

5 Justin Colannino, Mirela Damian, Ferran Hurtado, Stefan Langerman, Henk Meijer, Suneeta Ramaswami, Diane L. Souvaine, and Godfried Toussaint. Efficient many-to-many point matching in one dimension. *Graphs and Combinatorics*, 23:169–178, 2007. doi:10.1007/S00373-007-0714-3.

6 Justin Colannino and Godfried Toussaint. A faster algorithm for computing the link distance between two point sets on the real line. Technical report, Technical Report SOCS-TR-2005.5, McGill University, School of Computer Science, 2005.

7 Thomas Eiter and Heikki Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):109–133, 1997. doi:10.1007/S002360050075.

8 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987. doi:10.1145/28869.28874.

9 Harold N. Gabow. A data structure for nearest common ancestors with linking. *ACM Trans. Algorithms*, 13(4):45:1–45:28, 2017. doi:10.1145/3108240.

10 Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989. doi:10.1137/0218069.

11 Akshaykumar Gattani, Sharath Raghvendra, and Pouyan Shirzadian. A robust exact algorithm for the Euclidean bipartite matching problem. *Proceedings of the 37th Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, 36:51706–51718, 2023.

12 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.

13 Judith Keijsper and Rudi Pendavingh. An efficient algorithm for minimum-weight bibranching. *Journal of Combinatorics Theory, Series B*, 73(2):130–145, 1998. doi:10.1006/JCTB.1998.1817.

14 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983. doi:10.1137/0212002.

15 Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

16 Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

17 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980. doi:10.1137/0209046.

18 Qin Lv, Moses Charikar, and Kai Li. Image similarity search with compact data structures. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM 2004)*, pages 208–217, 2004. doi:10.1145/1031171.1031213.

19 David Mumford. Mathematical theories of shape: Do they model perception? In *Geometric methods in computer vision*, volume 1570, pages 2–10, 1991.

20 R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proceedings of the 29th Annual Symposium on Computational Geometry (SoCG 2013)*, pages 9–16, 2013. doi:10.1145/2462356.2480283.

21 R. Sharathkumar and Pankaj K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 306–317, 2012. doi:10.1137/1.9781611973099.29.

22 Ranjith Unnikrishnan and Martial Hebert. Measures of similarity. In *Proceedings of the 7th IEEE Workshops on Applications of Computer Vision (WACV/MOTION 2005)*, volume 1, pages 394–394. IEEE, 2005.

23 Remco C. Veltkamp and Michiel Hagedoorn. State of the art in shape matching. *Principles of visual information retrieval*, pages 87–119, 2001. doi:10.1007/978-1-4471-3702-3_4.