


Geometric Bipartite Matching Based Exact Algorithms for Server Problems

Sharath Raghvendra 

North Carolina State University, Raleigh, NC, USA

Pouyan Shirzadian 

Virginia Tech, Blacksburg, VA, USA

Rachita Sowle 

Virginia Commonwealth University, Richmond, VA, USA

Abstract

For any given metric space, obtaining an offline optimal solution to the classical k -server problem can be reduced to solving a minimum-cost partial bipartite matching between two point sets A and B within that metric space.

For d -dimensional ℓ_p metric space, we present an $\tilde{O}(\min\{nk, n^{2-\frac{1}{2d+1}} \log \Delta\} \cdot \Phi(n))$ time algorithm for solving this instance of minimum-cost partial bipartite matching; here, Δ represents the spread of the point set, and $\Phi(n)$ is the query/update time of a d -dimensional dynamic weighted nearest neighbor data structure. Our algorithm improves upon prior algorithms that require at least $\Omega(nk\Phi(n))$ time. The design of minimum-cost (partial) bipartite matching algorithms that make sub-quadratic queries to a weighted nearest-neighbor data structure, even for bounded spread instances, is a major open problem in computational geometry. We resolve this problem at least for the instances that are generated by the offline version of the k -server problem.

Our algorithm employs a hierarchical partitioning approach, dividing the points of $A \cup B$ into rectangles. It maintains a partial minimum-cost matching where any point $b \in B$ is either matched to another point $a \in A$ or to the boundary of the rectangle it is located in. The algorithm involves iteratively merging pairs of rectangles by erasing the shared boundary between them and recomputing the minimum-cost partial matching. This continues until all boundaries are erased and we obtain the desired minimum-cost partial matching of A and B . We exploit geometry in our analysis to show that each point participates in only $\tilde{O}(n^{1-\frac{1}{2d+1}} \log \Delta)$ number of augmenting paths, leading to a total execution time of $\tilde{O}(n^{2-\frac{1}{2d+1}} \Phi(n) \log \Delta)$.

We also show that, for the ℓ_1 norm and d dimensions, any algorithm that can solve instances of the offline n -server problem with an exponential spread in $T(n)$ time can be used to compute minimum-cost bipartite matching in a complete graph defined on two $(d-1)$ -dimensional point sets under the ℓ_1 norm within $T(n)$ time. This suggests that removing spread from the execution time of our algorithm may be difficult as it immediately results in a sub-quadratic algorithm for bipartite matching under the ℓ_1 norm.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry

Keywords and phrases Minimum-Cost Bipartite Matching, Server Problems, Primal-Dual Approach

Digital Object Identifier 10.4230/LIPIcs.SoCG.2025.72

Related Version Full Version: <https://arxiv.org/abs/2504.06079> [19]

Funding This research was supported by NSF grants CCF 2514753 and CCF 2223871.

1 Introduction

This paper considers two classical optimization problems: the *offline k -server* problem and the *minimum-cost bipartite matching* problem in geometric settings.



© Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle;
licensed under Creative Commons License CC-BY 4.0

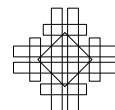
41st International Symposium on Computational Geometry (SoCG 2025).

Editors: Oswin Aichholzer and Haitao Wang; Article No. 72; pp. 72:1–72:15

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Offline k -server problem and its variant. Consider a sequence of requests $\varsigma = \langle r_1, \dots, r_m \rangle$ in a metric space equipped with the cost function $d(\cdot, \cdot)$. The cost of a single server *servicing* the requests in ς is the sum of the distances between every consecutive pair of points in the sequence, i.e., $c(\varsigma) = \sum_{i=1}^{m-1} d(r_i, r_{i+1})$. Given a sequence $\sigma = \langle r_1, \dots, r_n \rangle$ of n requests and an integer $1 \leq k \leq n$, the *k -sequence partitioning problem* (or simply the *k -SP problem*) requires partitioning the requests in σ into k subsequences $\varsigma_1, \dots, \varsigma_k$ so that $\sum_{i=1}^k c(\varsigma_i)$ is minimized. The optimal solution for the k -sequence partitioning problem is the cheapest way for k servers to service all of the requests in σ . We also consider a variant: the *k -sequence partitioning with initial locations problem* (or the *k -SPI problem*). Here, in addition to the requests σ , we are also given the initial locations $\eta = \langle s_1, \dots, s_k \rangle$ for the k servers. The objective of this problem is to partition $\sigma' = \eta\sigma$ into k subsequences $\varsigma_1, \dots, \varsigma_k$ so that the initial location s_j of server j appears in the first element of the subsequence ς_j and the cost $\sum_{i=1}^k c(\varsigma_i)$ is minimized. The optimal solution to the k -SPI problem is the offline optimal solution to the well-known k -server problem.

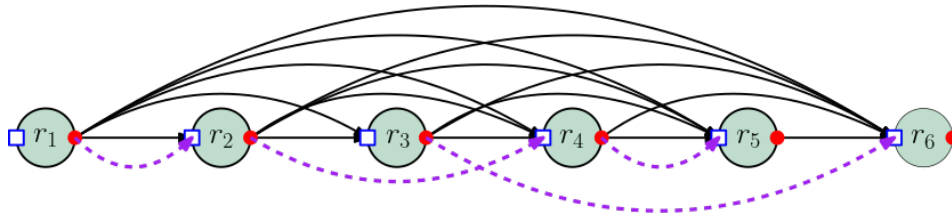
We assume that the requests in σ and the initial locations in η are scaled and translated so that they are contained inside the unit hypercube $[0, 1]^d$. Such scaling and translation do not impact the optimal solutions for the k -SP and k -SPI problems. We define the diameter $\text{Diam}(\sigma)$ to be the largest distance between any two request locations in σ and the *closest pair distance*, denoted by $\text{CP}(\sigma)$, to be the smallest non-zero distance between any two requests in σ . The *spread*, denoted by Δ , is the ratio of the diameter to the closest-pair distance, i.e., $\Delta = \text{Diam}(\sigma)/\text{CP}(\sigma)$.

Minimum-cost bipartite matching. Consider a weighted bipartite graph $G(A \cup B, E \subseteq A \times B)$, where each edge between u and v is assigned a real-valued cost. Let $n := \min\{|A|, |B|\}$. A *matching* of size $t \leq n$, or simply a *t -matching*, is a set of t edges in G that are vertex-disjoint. The *cost* of any matching M is the sum of the costs of its edges. Given a parameter $t \leq n$, the *minimum-cost bipartite t -matching problem* seeks to find a t -matching with a minimum cost. When $|A| = |B| = n$, a matching of size n is called a *perfect matching*. When A and B are d -dimensional point sets and the cost of any edge between $a \in A$ and $b \in B$ is the ℓ_p distance $\|a - b\|_p$, the problem of finding the minimum-cost bipartite t -matching is also called the (partial) *geometric bipartite matching problem*.

Relating the two problems. Chrobak *et al.* [4] established a reduction from the minimum-cost bipartite t -matching problem to the k -SPI problem.

► **Lemma 1** ([4, Theorem 11]). *Any algorithm that computes an optimal solution to the n -SPI in an arbitrary metric space in $T(n)$ time can also find, in $T(n) + O(n^2)$ time, a minimum-cost perfect matching in any complete bipartite graph with real-valued costs.*

We strengthen the connection between the two problems by showing a reduction in the reverse direction, i.e., we reduce the k -SP (resp. k -SPI) problem to the minimum-cost bipartite t -matching problem. Given an input sequence σ of requests to the k -SP problem, we construct a bipartite graph \mathcal{G}_σ with a vertex set $A \cup B$ and a set of edges \mathcal{E} as follows. *Vertex Set:* For each request r_i , we create a vertex b_i (resp. a_i) in B (resp. A) and designate it as the *entry* (resp. *exit*) gate for request r_i . *Edge Set:* The exit gate a_i of request r_i is connected to the entry gate b_j of every subsequent request r_j with $j > i$ with an edge. The cost of this edge is $d(a_i, b_j) = \|r_i - r_j\|_p$. Any minimum-cost $(n - k)$ -matching M in \mathcal{G}_σ can be used to find an optimal solution to the k -SP problem. See Figure 1.



■ **Figure 1** The graph \mathcal{G}_σ constructed for $\sigma = \langle r_1, r_2, \dots, r_6 \rangle$. The vertex set A (red disks) and B (blue squares) represent the exit and entry gates of each request, and the purple dashed lines show an $(n - 2)$ -partial matching on \mathcal{G}_σ representing a 2-partitioning $\langle r_1, r_2, r_4, r_5 \rangle$ and $\langle r_3, r_6 \rangle$.

For the k -SPI problem, for each server j , we add a vertex a^j at the initial location s_j to A and connect a^j to the entry gate b_i of every request r_i in σ . The cost of this edge is $d(a^j, b_i) = \|s_j - b_i\|_p$. A minimum-cost n -matching in this graph can be converted to an optimal solution for the k -SPI problem.

A different reduction from k -SP and k -SPI problems to the minimum-cost flow problem has been presented in previous works [4, 22, 24], leading to the development of $O(n^2k)$ time algorithm. However, unlike our reduction, their approach generates instances that include edges whose costs are $-\infty$ and fails to maintain the metric properties of costs.

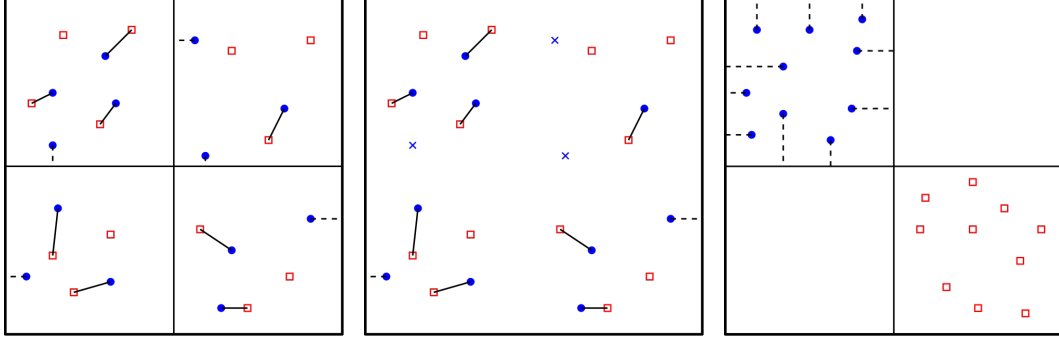
► **Lemma 2.** *An optimal solution for an instance σ (resp. σ') of k -SP (resp. k -SPI) problem can be found by computing a minimum-cost matching of size $n - k$ (resp. n) in \mathcal{G}_σ (resp. $\mathcal{G}_{\sigma'}$).*

Finally, we extend the reduction Chrobak *et al.* [4] to geometric settings and provide a reduction from the geometric minimum-cost matching problem under the ℓ_1 -norm to the geometric version of the n -SPI problem. This reduction, however, creates an instance of the n -SPI problem with a spread of 3^n . Lemma 3 follows directly from this reduction.

► **Lemma 3.** *Any algorithm that can solve the d -dimensional n -SPI problem under the ℓ_1 costs in $T(n)$ time can also be used to solve an instance of minimum-cost bipartite matching under the ℓ_1 costs on a complete graph in $T(n)$ time.*

Related work. For a graph with m edges and n vertices, the classical Hungarian algorithm computes a minimum-cost t -matching for all values of $0 \leq t \leq n$ [11, 16]. Starting with an empty matching, the Hungarian algorithm iteratively builds a minimum-cost i -matching from the maintained minimum-cost $(i - 1)$ -matching in $O(m + n \log n)$ time by finding a *minimum net-cost augmenting path*, i.e., an augmenting path that increases the matching cost by the smallest value. The overall execution time of the Hungarian algorithm is $O(nm + n^2 \log n)$, or $O(n^3)$ when $m = \Theta(n^2)$. Despite substantial efforts, this remains the most efficient algorithm for the problem. Notable exceptions include specialized cases, such as graphs with small vertex separators [13] or integer edge weights [2, 5, 6].

In geometric settings, Vaidya showed that each iteration of the Hungarian algorithm can be implemented in $\tilde{O}(n\Phi(n))$ time, where $\Phi(n)$ represents the query/update time of a dynamic weighted nearest neighbor (DWNN) data structure. Thus, the minimum-cost t -matching can be computed in $O(n^2\Phi(n))$ time, which is sub-cubic in n provided $\Phi(n)$ is sub-linear. For instance, for the ℓ_1 norm and d dimensions, $\Phi(n) = O(\log^d n)$ [27] and for the ℓ_2 norm and 2 dimensions, $\Phi(n) = \log^{O(1)} n$ [8]. In these cases, the Hungarian algorithm can be implemented in near-quadratic time.



■ **Figure 2** Illustration of the GRS algorithm.

Designing algorithms that compute a minimum-cost matching using sub-quadratic DWNN queries remains a central open problem in computational geometry. There are three notable exceptions. First, for points with integer coordinates and the ℓ_1 and ℓ_∞ norms, Sharathkumar and Agarwal [26] adapted a cost-scaling algorithm [5] and presented an algorithm that executes in $\tilde{O}(n^{3/2}\Phi(n))$ time. Second, Sharathkumar [25] extended this result to 2-dimensional point sets with integer coordinates and the ℓ_2 costs. Their result, however, relies on planarity as well as the edge costs being the square root of integers and does not extend to d -dimensional points with real-valued coordinates. Third, Gattani *et al.* [7] presented a divide-and-conquer algorithm (GRS algorithm) with a worst-case runtime of $\tilde{O}(n^2\Phi(n)\log \Delta)$, where Δ is the spread of the points. For 2-dimensional *stochastic* points drawn from an unknown distribution, however, the expected runtime of the GRS algorithm improves to $\tilde{O}(n^{7/4}\Phi(n)\log \Delta)$.

The GRS algorithm uses a randomly-shifted quadtree to divide the problem into smaller subproblems. Within each square \square of the quadtree, it recursively computes a *minimum-cost extended matching*, allowing each point of B in \square to match either to a point of A in \square or to the boundaries of \square (Figure 2(left)). The algorithm then combines the matchings of the child squares by *erasing* their common boundaries, freeing points of B that were matched to those boundaries (Figure 2(middle)), and then it iteratively matches the freed points.

Gattani *et al.* [7] showed that for stochastic point sets A and B , most points in B will match to a close-by point in A and only $\tilde{O}(n^{3/4})$ points will match to boundaries. Thus, erasing these boundaries creates $\tilde{O}(n^{3/4})$ free points, each again matched in $O(n\Phi(n))$ time, resulting in an overall runtime of $\tilde{O}(n^{7/4}\Phi(n))$. However, this efficiency does not extend to arbitrary point sets. For instance, if all points in A lie in one child \square_1 and all points in B lie in another child \square_2 , then all edges of a minimum-cost matching cross the boundaries. In this case, the minimum-cost extended matching in \square_2 matches all points in B to its boundary (Figure 2(right)), and erasing the boundary creates n free points. Hence, the merge step takes $\Omega(n^2\Phi(n))$ time. Furthermore, unlike the Hungarian algorithm, the GRS algorithm does not guarantee optimal intermediate matchings and cannot compute minimum-cost t -matchings.

Our results. The optimal solutions to the k -SP (resp. k -SPI) problems can be computed in $\tilde{O}(nk\Phi(n))$ time by non-trivially adapting the Hungarian algorithm to find a minimum-cost $(n - k)$ -matching (resp. n -matching) in \mathcal{G}_σ (resp. $\mathcal{G}_{\sigma'}$). This algorithm, however, makes quadratic queries to a DWNN data structure when $k = \Theta(n)$. The main contribution of this paper is the design of a novel algorithm that, for any k , computes the optimal solution to the k -SP and k -SPI problems using only a sub-quadratic number of DWNN queries.

► **Theorem 4.** *Given any sequence σ (resp. $\sigma' = \eta\sigma$) of n requests (resp. n requests and k initial server locations) in 2 dimensions with a spread of Δ , and a value $1 \leq k \leq n$, there exists a deterministic algorithm that computes the optimal solution for the instance of k -SP (resp. k -SPI) problem under the ℓ_p norm in $\tilde{O}(\min\{nk, n^{1.8} \log \Delta\} \cdot \Phi(n))$ time.*

Our algorithm also extends to higher dimensions, and for any dimension $d \geq 2$, it computes optimal solutions to the k -SP and k -SPI problems in $\tilde{O}(\min\{nk, n^{2-\frac{1}{2d+1}} \log \Delta\} \cdot \Phi(n))$ time.

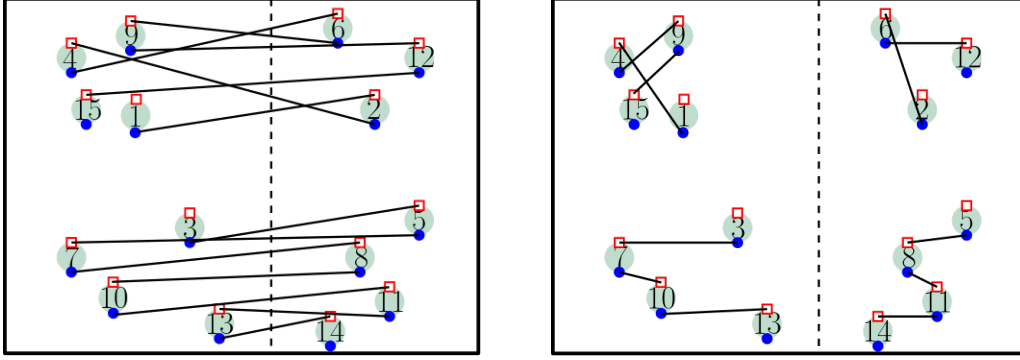
Designing geometric bipartite matching algorithms that perform a sub-quadratic number of queries to a DWNN data structure is challenging. Nevertheless, Theorem 4 shows that the bipartite matching instances generated by k -SP and k -SPI problems with bounded spread can be solved using sub-quadratic DWNN queries. For unbounded spread cases, the challenge remains elusive, at least for the k -SPI problem since, by Lemma 3, such an algorithm yields a sub-quadratic solution for the d -dimensional minimum-cost bipartite matching problem under ℓ_1 costs.

Our algorithm uses a hierarchical partitioning tree, whose nodes (referred to as cells) are axis-parallel rectangles with an aspect ratio of at most 3, and each cell splits into two smaller rectangles, forming its children. Our algorithm maintains a set of *current* cells \mathcal{C} that partition the input points and computes a minimum-cost extended $(n - k)$ -matching, where the points in B are allowed to match to the cell boundaries. It then replaces a pair of sibling rectangles in \mathcal{C} with their parent, effectively erasing their shared boundary and creating additional free points. These free points are then matched again by finding minimum net-cost augmenting paths. When all the boundaries are erased, the algorithm terminates with the desired minimum-cost $(n - k)$ -matching.

Our efficiency analysis faces three main hurdles. First, while finding a minimum net-cost augmenting path typically requires a search on the entire graph, we show that for extended matchings, such paths lie entirely within a current cell, allowing efficient local searches inside each current cell and selecting the global minimum across cells. Second, bounding the time to merge two cells is challenging, especially since some current cells may hold $\Theta(n)$ points, where each point is matched to a far away point in the optimal solution (Figure 3(left)). We address this challenge by showing that any minimum-cost extended matching has only $O(n^{0.8})$ points matched to the shared boundary. To establish this, we critically use the fact that every sub-problem has a hidden low-cost high-cardinality matching with sub-linearly many free points (see Figure 3 (right)). Finally, the k free points in our extended matching M may differ from those in the minimum-cost $(n - k)$ -matching M^* . For instance, a point $b \in B$ may be free in M^* but matched to the boundary in M , leaving another point b' unmatched. While correcting such mismatches via alternating and augmenting paths can take $\Theta(n)$ time each (leading to a naïve bound of $O(nk)$), we use geometry to show that only $O(n^{0.8})$ such corrections occur for each cell, yielding an overall runtime to $\tilde{O}(n^{1.8}\Phi(n) \log \Delta)$.

Our proof techniques also refine the analysis of the GRS algorithm when the cost between points $a \in A$ and $b \in B$ is $\|a - b\|_2^q$ for any $q \geq 2$. Prior work showed that the GRS algorithm computes the minimum-cost perfect matching between two 2-dimensional stochastic point sets in $\tilde{O}(n^{2-\frac{1}{2(q+1)}} \Phi(n))$ time [7], which implies a quadratic number of queries to the DWNN data structure as $q \rightarrow \infty$. Raghvendra *et al.* [20] observed that, for any $q \geq 1$, there exists a low-cost, high-cardinality matching with sublinear free points. Building on a similar observation and our novel analysis techniques, we show that the GRS algorithm makes only a sub-quadratic number of DWNN queries, regardless of the value of q .

► **Theorem 5.** *Suppose U is a set of $2n$ points inside the unit square and A is a subset chosen uniformly at random from all subsets of size n . Let $B = U \setminus A$. Then, there exists an algorithm that computes the minimum-cost perfect matching on the complete bipartite graph on A and B under ℓ_2^q costs in $\tilde{O}(n^{7/4}\Phi(n) \log \Delta)$ expected time.*



■ **Figure 3** (left) A sub-problem of the k -SP problem, where the optimal solution has a high cost, and (right) there exists a low-cost high-cardinality matching inside the sub-problems.

The setting originally considered in the analysis of the GRS algorithm [7], where A and B are samples from the same distribution, is a special case of randomly partitioning a set of $2n$ points into two sets A and B of n points, considered in Theorem 5.

Due to space limitations, we restrict the description of our algorithm and its analysis to the k -SP problem in 2 dimensions. All other results, including extensions to higher dimensions, are presented in the full version of our paper.

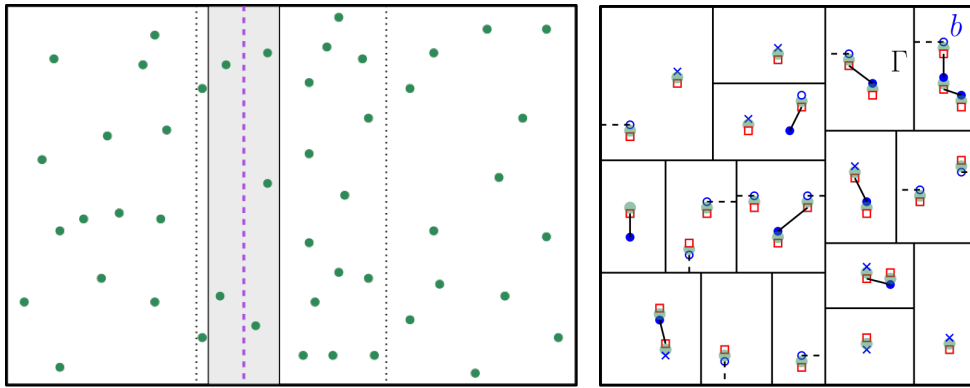
2 Geometric Primal-Dual Framework

Let σ be an input to the k -SP problem, where the distance between two locations a and b is given by $d(a, b) = \|a - b\|_p$. In this section, we introduce a primal-dual framework based on hierarchical partitioning to compute a minimum-cost $(n - k)$ -matching in $\mathcal{G}_\sigma = (A \cup B, E \subset A \times B)$. We begin by describing the hierarchical partitioning scheme.

2.1 Hierarchical Partitioning

Using $\lambda := 9n^{-1/5}$, we construct a hierarchical partitioning \mathcal{H} recursively. Each node of \mathcal{H} is an axis-parallel rectangle, referred to as a *cell*. The root node, $\square^* := [-3n, 3n]^2$, contains all points in $A \cup B$. For each node \square , let A_\square and B_\square be the points of A and B inside \square and let $n_\square = |A_\square \cup B_\square|$. If $n_\square \leq 2$ (i.e., \square is empty or contains the entry and exit gates of a single request), then \square is marked as a leaf node. Otherwise, we partition \square into two smaller rectangles as follows. Let ℓ_\square be the larger of the length and width of rectangle \square . Without loss of generality, assume that ℓ_\square is the width of \square and let x_{\min} be the x -coordinate of the bottom-left corner of \square . For any value $\hat{x} \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}]$, define $\Lambda(\hat{x}) := \{u \in A_\square \cup B_\square : |u_x - \hat{x}| \leq \ell_\square \lambda\}$; here, u_x denotes the x coordinate of the point u . Let $x^* := \arg \min_{\hat{x} \in [x_{\min} + \frac{\ell_\square}{3}, x_{\min} + \frac{2\ell_\square}{3}]} |\Lambda(\hat{x})|$. We partition \square into two smaller rectangles by using a vertical line defined by $x = x^*$ and add them as the children of \square to \mathcal{H} . We refer to the segment partitioning \square into its two children as its *divider* and denote it by Γ_\square . See Figure 4(left). For any cell \square , the four sides of its rectangle are defined by the dividers of its ancestor or the boundaries of the root square. This completes the construction of \mathcal{H} . Note that the height of the tree is $O(\log n\Delta)$. A simple sweep-line algorithm can compute x^* in $O(n_\square \log n_\square)$ time. Using this procedure, we construct \mathcal{H} in $\tilde{O}(n \log(n\Delta))$ time.

► **Lemma 6.** *For each cell \square of \mathcal{H} , the ratio of the largest to the smallest side of \square is at most 3. Furthermore, the number of points of $A_\square \cup B_\square$ with a distance smaller than $\ell_\square \lambda$ to the divider Γ_\square is $O(n_\square \lambda)$.*



■ **Figure 4** (left) Partitioning a rectangle into two children (the purple dashed vertical line is the divider), and (right) a \mathcal{C} -extended 20-matching with 9 matched points (blue discs), 11 boundary-matched points (blue circles), and 8 free points (blue crosses). The matching cost is the total length of the solid and dashed lines. The boundary-matched point b is matched to the divider Γ .

Recollect that a matching M in \mathcal{G}_σ is a subset of vertex-disjoint edges. We refer to a point $b \in B$ as *unmatched* in M if it does not have an edge of M incident on it and *matched* otherwise. The following structural property of the k -SP problem will be critical in bounding the efficiency of our algorithm.

► **Lemma 7.** *For any cell \square of \mathcal{H} , there exists a matching M' between $A_\square \cup B_\square$ that matches all except $O(n_\square^{4/5})$ points of B_\square and has a cost $O(\ell_\square n_\square^{3/5})$.*

2.2 Extended Bipartite Matching

Suppose we are given a subset \mathcal{C} of cells from \mathcal{H} that partitions \square^* , i.e., the cells of \mathcal{C} are interior disjoint and these cells cover the root square \square^* . Let $d(u, \square)$ denote the shortest distance from u to the boundaries of the cell \square . For any point u inside \square^* , let \square_u be the cell of \mathcal{C} that contains u . We define $d(b, \mathcal{C}) = d(b, \square_b)$. We extend the definition of matching to allow points in B to match to the boundaries of cells in \mathcal{C} . A \mathcal{C} -*extended matching* consists of a matching M as well as a subset $B^{\mathcal{C}}$ of points of B that are unmatched in M but instead matched to the boundaries of the cells of \mathcal{C} that contain them. See Figure 4(right). The cost of a \mathcal{C} -extended matching $M^{\mathcal{C}}$ is

$$w_{\mathcal{C}}(M^{\mathcal{C}}) := \sum_{(a,b) \in M} d(a,b) + \sum_{b \in B^{\mathcal{C}}} d(b, \mathcal{C}). \quad (1)$$

We refer to all points of $B^{\mathcal{C}}$ as *boundary-matched*. All points of B that are neither matched in M nor boundary-matched are considered *free*. All points of A that are not matched in M are also considered free. The size of $M^{\mathcal{C}}$ is equal to $|M| + |B^{\mathcal{C}}|$. When \mathcal{C} is clear from the context, we refer to $M^{\mathcal{C}}$ as an extended matching. An extended matching $M^{\mathcal{C}}$ of size t with the minimum cost is called a *minimum-cost \mathcal{C} -extended t -matching*.

Consider the partitioning $\mathcal{C}^* = \{\square^*\}$, where \square^* is the root cell of \mathcal{H} . Using the fact that the input points are far from the boundaries of \square^* , we show in Lemma 8 below that any minimum-cost \mathcal{C}^* -extended matching $M^{\mathcal{C}^*}$ of size t is also a minimum-cost t -matching, i.e., no point of B is boundary-matched in $M^{\mathcal{C}^*}$.

► **Lemma 8.** *Suppose $\mathcal{C}^* = \{\square^*\}$, where \square^* is the root cell of \mathcal{H} . Let $M^{\mathcal{C}^*} = (M, B^{\mathcal{C}^*})$ be a minimum-cost extended t -matching on \mathcal{G}_σ , for $t < n$. Then, the matching M is a minimum-cost t -matching.*

Let $M^C = (M, B^C)$ denote a \mathcal{C} -extended matching. Any path P on the graph \mathcal{G}_σ whose edges alternate between matching and non-matching edges in M is called an *alternating path*. An alternating path P is called an *augmenting path* if P starts from a free point $b \in B$ and ends with either (i) a free point $a \in A$, or (ii) a point $b' \in B$. We *augment* the extended matching M^C along an augmenting path P by updating its matching $M \leftarrow M \oplus P$ and for case (ii), we match b' to the boundary and update B^C to include b' . The *net-cost* of P in case (i) is $\phi(P) := \sum_{(a,b) \in P \setminus M} d(a,b) - \sum_{(a,b) \in P \cap M} d(a,b)$, and in case (ii) is

$$\phi(P) := d(b', \mathcal{C}) + \sum_{(a,b) \in P \setminus M} d(a,b) - \sum_{(a,b) \in P \cap M} d(a,b).$$

From Lemma 8, given a sequence σ of n requests in the unit square, one can compute an optimal solution to the k -SP problem on σ by computing a minimum-cost \mathcal{C}^* -extended $(n-k)$ -matching on \mathcal{G}_σ . Given a partitioning \mathcal{C} , to compute a minimum-cost \mathcal{C} -extended $(n-k)$ -matching, similar to the Hungarian algorithm, one can start from an empty extended matching M^C and iteratively augment M^C along a minimum net-cost augmenting path. In the next section, we present a primal-dual framework that our algorithm uses to efficiently compute minimum net-cost augmenting paths.

2.3 A Constrained Dual Formulation for Extended Matchings

Suppose \mathcal{C} is a set of cells of \mathcal{H} partitioning the root cell \square^* . Consider a \mathcal{C} -extended matching $M^C = (M, B^C)$ on \mathcal{G}_σ along with a set of non-negative dual weights $y : A \cup B \rightarrow \mathbb{R}_{\geq 0}$. Let A_F be the set of free points of A with respect to M^C . We say that $M^C, y(\cdot)$ is *feasible* if,

$$y(b) - y(a) \leq d(a,b), \quad \forall (a,b) \in E, \quad (2)$$

$$y(b) - y(a) = d(a,b), \quad \forall (a,b) \in M, \quad (3)$$

$$y(b) \leq d(b, \mathcal{C}), \quad \forall b \in B, \quad (4)$$

$$y(b) = d(b, \mathcal{C}), \quad \forall b \in B^C, \quad (5)$$

$$y(a) = 0, \quad \forall a \in A_F. \quad (6)$$

For any edge $(a,b) \in E$, the *slack* of (a,b) is defined as $s(a,b) := d(a,b) - y(b) + y(a)$. The edge (a,b) is *admissible* if $s(a,b) = 0$. For any point $b \in B$, the slack of b is defined as $s(b) := d(b, \mathcal{C}) - y(b)$. For any feasible extended matching $M^C, y(\cdot)$, the slack of every edge as well as every point $b \in B$ is non-negative. Recall that an augmenting path P starts at a free point $b \in B$ and ends at (i) a free point $a \in A$ or (ii) a point $b' \in B$. The path P is *admissible* if all edges of P are admissible and in case (ii), the slack of the end-point b' is $s(b') = 0$. The following properties of extended feasible matchings are critical in the design of an efficient and correct algorithm.

► **Lemma 9.** *Given a feasible \mathcal{C} -extended t -matching $M^C = (M, B^C)$ and a set of non-negative dual weights $y(\cdot)$ on $A \cup B$, let P be a minimum net-cost augmenting path with respect to M^C . Let, for any $\square \in \mathcal{C}$, $y_\square = \max_{b' \in B_\square} y(b')$. Then,*

- (a) *all points of P lie inside a single cell of \mathcal{C} , and*
- (b) *if, for every cell $\square \in \mathcal{C}$, $y_\square \leq \phi(P)$ and for all free points $b \in B_\square$, $y(b) = y_\square$, then M^C is a minimum-cost extended t -matching.*

The property described in Lemma 9(a) is important for the design of an efficient algorithm. Unlike the Hungarian algorithm, which searches the entire graph for the minimum net-cost augmenting path, our algorithm can find the minimum net-cost augmenting path by searching

for the cheapest augmenting path inside each cell $\square \in \mathcal{C}$ and then taking the smallest among them. Thus, we can replace a global search with a search inside each cell of \mathcal{C} . The property in Lemma 9(b) is important for the design of a correct algorithm since it provides conditions under which an extended matching is a minimum-cost extended matching. Our algorithm is designed to maintain these conditions as invariants during its execution. Lemma 10 provides a method to update the dual weights, which is essential during the process of merging cells.

► **Lemma 10.** *Suppose $M^{\mathcal{C}}, y(\cdot)$ is a feasible \mathcal{C} -extended matching and P is a minimum net-cost augmenting path. For any cell $\square \in \mathcal{C}$, define $y_{\square} = \max_{b' \in B_{\square}} y(b')$, and suppose $y_{\square} \leq \phi(P)$ and $y(b_f) = y_{\square}$ for all free points $b_f \in B_{\square}$. Then, one can update the dual weights in $\tilde{O}(n_{\square} \Phi(n_{\square}))$ time such that $M^{\mathcal{C}}, y(\cdot)$ remains feasible, $y(b) \leq \phi(P)$ for all $b \in B_{\square}$, and $y(b_f) = \phi(P)$ for all free points $b_f \in B_{\square}$.*

The next lemma provides critical properties that allow for correctly and efficiently merging cells in \mathcal{C} .

► **Lemma 11.** *For a cell \square of \mathcal{H} , suppose \square' and \square'' denote its two children, and let \mathcal{C} denote a partitioning containing \square' and \square'' . Let $\mathcal{C}' = \mathcal{C} \cup \{\square\} \setminus \{\square', \square''\}$. Given a feasible \mathcal{C} -extended matching $(M, B^{\mathcal{C}}), y(\cdot)$, let $B_{\square}^{\mathcal{C}} \subseteq B^{\mathcal{C}}$ denote the subset of boundary-matched points that are matched to the divider Γ_{\square} of \square . Then,*

- (a) *the \mathcal{C}' -extended matching $(M, B^{\mathcal{C}} \setminus B_{\square}^{\mathcal{C}}), y(\cdot)$ is also feasible, and,*
- (b) *$|B_{\square}^{\mathcal{C}}| = O(n^{4/5})$.*

From Lemma 11(a), erasing a divider does not cause the feasibility conditions to be violated. The proof of Lemma 11(a) relies on the fact that when we erase the divider of a cell in \mathcal{C} , the RHS of (4) will only increase and so it is not violated. Despite preserving the feasibility conditions, erasing the boundary may result in the violation of Lemma 9(b), i.e., the matching may no longer be a minimum-cost extended matching. In our algorithm in Section 3, we describe a process to adjust the matching $M^{\mathcal{C}}$ and dual weights $y(\cdot)$ and obtain a minimum-cost extended matching.

Residual Graph. Similar to the Hungarian algorithm, we define a residual graph that assists in finding the minimum net-cost augmenting path. Consider a feasible \mathcal{C} -extended matching $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ along with a set of dual weights $y(\cdot)$ on points of $A \cup B$. For each cell $\square \in \mathcal{C}$, we define a residual graph \mathcal{G}_{\square} . The vertex set of \mathcal{G}_{\square} is a source vertex s and the points in $A_{\square} \cup B_{\square}$. For any edge $(a, b) \in E$ inside \square , if $(a, b) \in M$ (resp. $(a, b) \notin M$), there is an edge directed from a to b (resp. from b to a) with a weight $s(a, b)$ in \mathcal{G}_{\square} . Furthermore, there is an edge directed from s to every free point $b \in B$ with a weight $y(b)$.

3 A Sub-Quadratic Algorithm for the k -SP Problem

In this section, we describe an algorithm that, given a sequence σ of n requests in 2-dimensions, computes the optimal solution to the k -SP problem in $\tilde{O}(n^{1.8} \Phi(n) \log(n\Delta))$ time.

3.1 Algorithm

Initialize \mathcal{C} to the leaf cells of \mathcal{H} . Let $M^{\mathcal{C}} = (M, B^{\mathcal{C}})$ be the extended matching maintained by the algorithm and initialized to $M = \emptyset$ and $B^{\mathcal{C}} = \emptyset$. For each point $v \in A \cup B$, let $y(v)$ denote its dual weight initialized to $y(v) = 0$. Let B_F , initialized to B , be the free points of B with respect to $M^{\mathcal{C}}$. For each cell \square that contains at least one free point $b \in B_F$, let

P_\square denote the minimum net-cost augmenting path inside \square . Initially, since \square is a leaf of \mathcal{H} , it contains only one point $b \in B_F$, and therefore, P_\square is this point with a net-cost equal to $d(b, \mathcal{C})$. Our algorithm maintains a priority queue PQ storing every leaf cell $\square \in \mathcal{C}$ with at least one free vertex with a key of $\phi(P_\square)$. At any time during the execution of our algorithm, let \square_{\min} be the cell with the smallest key in PQ and let φ be the key of \square_{\min} . Execute the following steps until PQ becomes empty:

- While $|B_F| > k$,
 - *Extended Hungarian search step*: Extract the cell \square with the minimum key of φ from PQ. Augment the matching M^C along P_\square and update the key of \square in PQ (See Section 3.1.1 for details).
 - *Merge step*: If $\mathcal{C} = \{\square^*\}$, remove \square^* from PQ and return the matching M of M^C . Otherwise, pick a cell $\square' \in \mathcal{C}$ with the smallest perimeter and let \square and \square'' be the parent and sibling of \square' in \mathcal{H} , respectively. Erase the divider of \square , i.e., set $\mathcal{C} = \mathcal{C} \setminus \{\square', \square''\} \cup \{\square\}$. We execute a Merge procedure that updates the matching M^C inside the cell \square so that $\phi(P_\square)$ is at least φ (See Section 3.1.2 for details). At this point, the size of the updated extended matching M^C may not be $(n - k)$, i.e., there may be more than k free points.

Invariants. For each cell $\square \in \mathcal{C}$, let $y_\square = \max_{b \in B_\square} y(b)$. During the execution of our algorithm,

- (I1) the extended matching $M^C, y(\cdot)$ is feasible,
- (I2) For each cell $\square \in \mathcal{C}$, $y_\square \leq \varphi$ and for all free point $b \in B_\square$, $y(b) = y_\square$, and,
- (I3) The φ -value is non-decreasing. Furthermore, after each step of the algorithm, φ denotes the smallest net-cost of all augmenting paths with respect to M^C .

3.1.1 Details of the Extended Hungarian Search Step

Given a feasible extended matching $M^C, y(\cdot)$ and a cell $\square \in \mathcal{C}$, the extended Hungarian search procedure computes the minimum net-cost augmenting path P_\square and augments M^C along P_\square . It then computes the new minimum net-cost augmenting path and updates the key of \square in PQ to be its net-cost. This procedure is similar to the classical Hungarian search procedure executed on \mathcal{G}_\square and is mildly modified to include the augmenting paths that end at the boundary of \square . Details of the procedure are as follows.

1. *Update duals*: With s as the source vertex, execute Dijkstra's shortest path algorithm on the residual graph \mathcal{G}_\square . Let P_v be the shortest path from s to each $v \in A_\square \cup B_\square$ and let κ_v be the cost of P_v . Let

$$\kappa = \min\left\{\min_{a \in A_\square^F} \kappa_a, \min_{b \in B_\square} \kappa_b + s(b)\right\}. \quad (7)$$

For any $v \in A_\square \cup B_\square$, if $\kappa_v < \kappa$, set $y(v) \leftarrow y(v) + \kappa - \kappa_v$.

2. *Augment*: Let $u \in A_\square^F \cup B_\square$ be the point realizing the minimum distance in Equation (7). Let P be the augmenting path obtained by removing s . Augment M^C along P .
3. *Update key*: If B_\square has no free points, then remove \square from PQ. Otherwise,
 - a. Recompute the residual graph \mathcal{G}_\square with respect to the updated matching,
 - b. With s as the source, execute the Dijkstra's shortest path algorithm on \mathcal{G}_\square . For each $v \in A_\square \cup B_\square$, let κ_v be the distance from s to v .
 - c. Update the key of \square in PQ to $\kappa_\square = \min\{\min_{a \in A_\square^F} \kappa_a, \min_{b \in B_\square} \kappa_b + s(b)\}$.

Lemma 12 establishes the properties of the extended Hungarian search procedure.

► **Lemma 12.** *After the execution of the extended Hungarian search procedure for a cell \square , the extended matching $M^C, y(\cdot)$ remains feasible, $y(v) \leq \varphi$ for all points $v \in A_\square \cup B_\square$, and $y(b_f) = \varphi$ for all free points $b_f \in B_\square$. Furthermore, the path P computed by the procedure is a minimum net-cost augmenting path inside \square . After augmenting along P , the updated key for \square is the smallest net-cost of all augmenting paths inside \square and is at least φ .*

3.1.2 Details of the Merge Step

Given a feasible extended matching $M^C = (M, B^C), y(\cdot)$, any cell \square of \mathcal{H} where both of its children \square' and \square'' are in \mathcal{C} , and a value φ , the merge procedure uses the algorithm in Lemma 10 to update the dual weights $y(\cdot)$ inside \square' (resp. \square'') so that $M^C, y(\cdot)$ remains feasible, the dual weights of all points in \square' (resp. \square'') are at most φ , and the dual weight of all free points $b'_f \in B_{\square'}$ (resp. $b''_f \in B_{\square''}$) is $y(b'_f) = \varphi$ (resp. $y(b''_f) = \varphi$). The procedure then updates $\mathcal{C} \leftarrow \mathcal{C} \cup \{\square\} \setminus \{\square', \square''\}$ and makes the points matched to the divider Γ_\square free. While there exists a free point $b \in B_\square$ with $y(b) < \varphi$,

1. With s as the source, execute Dijkstra's shortest path algorithm on the residual graph \mathcal{G}_\square . For each $v \in A_\square \cup B_\square$, let P_v be the shortest path from s to v in \mathcal{G}_\square and let κ_v be its cost. Define

$$\kappa = \min \left\{ \min_{a \in A_\square^F} \kappa_a, \min_{b \in B_\square} \kappa_b + s(b), \min_{b \in B_\square} \kappa_b + \varphi - y(b) \right\}. \quad (8)$$

2. Let $u \in A_\square^F \cup B_\square$ be the point realizing the minimum value in Equation (8). Let P be the path obtained by removing s from the path P_u .
 - a. If $u \in B_\square$ and $\kappa = \kappa_u + \varphi - y(u)$ (i.e., κ is determined by the third element in the RHS of Equation (8)), then P is a path from a free point $b \in B_\square$ to the matched point u . For each $v \in A_\square \cup B_\square$, if $\kappa_v < \kappa$, update its dual weight to $y(v) \leftarrow y(v) + \kappa - \kappa_v$. The path P is an admissible alternating path with respect to the updated dual weights. Set $M \leftarrow M \oplus P$. Note that u is now a free point with $y(u) = \varphi$.
 - b. Otherwise, P is an augmenting path. For each $v \in A_\square \cup B_\square$, if $\kappa_v < \kappa$, update its dual weight to $y(v) \leftarrow y(v) + \kappa - \kappa_v$. The path P is an admissible augmenting path with respect to the updated dual weights. Augment M along P .

At the end of this execution, all free points of B in \square have a dual weight of φ . Finally, we update the key for \square by executing steps 3(a)–(c) from the extended Hungarian search step.

The following lemma states the useful properties of the merge step.

► **Lemma 13.** *After the execution of the merge procedure on a cell \square , the updated extended matching $M^C, y(\cdot)$ is feasible, the dual weights $y(v)$ for every $v \in A_\square \cup B_\square$ is at most φ , and $y(b_f) = \varphi$ for all free points $b_f \in B_\square$. Furthermore, the updated key for \square is the smallest net-cost of all augmenting paths within \square and is at least φ .*

3.2 Analysis

We begin by showing in Section 3.2.1 that the three invariants (I1)–(I3) hold during the execution of our algorithm and use them to show the correctness of our algorithm. We then show in Section 3.2.2 that the running time of our algorithm is $\tilde{O}(n^{9/5} \Phi(n) \log n \Delta)$.

3.2.1 Correctness

Our algorithm initializes M^C with a feasible \mathcal{C} -extended matching and sets all dual weights and φ to 0. Therefore, (I1) and (I2) hold at the start of the algorithm. The extended Hungarian search (Lemma 12) as well as the merge step (Lemma 13) do not violate the feasibility of

the extended matching and therefore (I1) holds during the execution of the algorithm. The extended Hungarian search (Lemma 12) and the merge (Lemma 13) procedures keep the dual weight of every point inside \square at or below φ , while ensuring that the dual weight of free points inside \square is φ ; hence, (I2) holds. Finally, both the extended Hungarian search (Lemma 12) and the merge (Lemma 13) procedures update the key of \square to the smallest net-cost of all augmenting paths inside \square and do not decrease the key of any cell. From this observation, (I3) follows in a straightforward way.

From (I1) and (I2), our algorithm maintains a feasible \mathcal{C} -extended matching where, for each cell $\square \in \mathcal{C}$, $y_\square \leq \varphi$. From (I3), φ is equal to the minimum net-cost of all augmenting paths with respect to $M^{\mathcal{C}}$; combining this with Lemma 9(b), we conclude that $M^{\mathcal{C}}$ is a minimum-cost \mathcal{C} -extended matching. Upon termination, the algorithm returns a minimum-cost \mathcal{C}^* -extended $(n - k)$ -matching for $\mathcal{C}^* = \{\square^*\}$, which from Lemma 8 has no boundary-matched points. Therefore, this matching is also a minimum-cost $(n - k)$ -matching, as desired.

3.2.2 Efficiency

Both the merge step and the extended Hungarian search step require an execution of Dijkstra's shortest path algorithm on \mathcal{G}_\square . In the full version of our paper, we show that this execution takes $\tilde{O}(n_\square \Phi(n_\square))$ time. Using this, we analyze the execution time.

We begin by establishing a bound on the execution time of the merge step, which combines \square' and \square'' into a single cell \square . At the start of this step, the dual weight of all free points in \square' and \square'' are raised to φ (from Lemma 10). As a result, once the divider is removed, the only free points with dual weights below φ are those that are matched to the divider Γ_\square . From Lemma 11(b), the number of points matched to Γ_\square is $O(n_\square^{4/5})$. Therefore, the while-loop in the merge procedure executes only $O(n_\square^{4/5})$ times. Since each iteration takes $\tilde{O}(n_\square \Phi(n_\square))$ time, the total execution time of a single execution of the merge step is $\tilde{O}(n_\square^{4/5} n_\square \Phi(n_\square))$. Given that each point is in only $O(\log n \Delta)$ many cells of \mathcal{H} , the total time taken by the merge step across all cells of \mathcal{H} is $\tilde{O}(n^{9/5} \Phi(n) \log n \Delta)$.

Similarly, if the execution time for the extended Hungarian search within a single cell \square is bounded by $O(n_\square^{4/5} n_\square \Phi(n_\square))$, then the cumulative execution time of the extended Hungarian search across all cells – and consequently, the total runtime of the entire algorithm – can be bounded by $\tilde{O}(n^{9/5} \Phi(n) \log n \Delta)$. In the remainder of our analysis, we establish a bound of $O(n_\square^{4/5} n_\square \Phi(n_\square))$ for the time taken by the extended Hungarian search within a single cell \square . Recall that the algorithm selects a cell \square containing the minimum net-cost augmenting path from the priority queue PQ and performs an extended Hungarian search procedure within \square , requiring $\tilde{O}(n_\square \Phi(n_\square))$ time. To analyze the total execution time of the extended Hungarian search procedure, we show that any cell \square can be selected by the algorithm at most $O(n_\square^{4/5})$ times. In particular, we categorize the selection of \square as a *low-net-cost* case if $\varphi \leq \ell_\square n^{-1/5}$ and a *high-net-cost* case otherwise. First, we show that in the high-net-cost case ($\varphi > \ell_\square n^{-1/5}$), the number of free points remaining in \square is $O(n_\square^{4/5})$, and as a result, the total number of high-net-cost selections of \square is $O(n_\square^{4/5})$.

► **Lemma 14.** *Given a feasible extended matching $M^{\mathcal{C}}, y(\cdot)$ and any cell $\square \in \mathcal{C}$, if the net-cost of the minimum net-cost augmenting path inside \square is greater than $\ell_\square n^{-1/5}$, then the number of free points of $M^{\mathcal{C}}$ is $O(n_\square^{4/5})$.*

Next, we bound the number of low-net-cost selections of \square . Define \mathcal{C}_\square as the set of all cells $\square' \in \mathcal{H}$ that are processed by the merge procedure while $\square \in \mathcal{C}$ and $\varphi \leq \ell_\square n^{-1/5}$ (Figure 5). Our algorithm always picks the smallest perimeter cells to merge, and as a result, we obtain the following lemma.

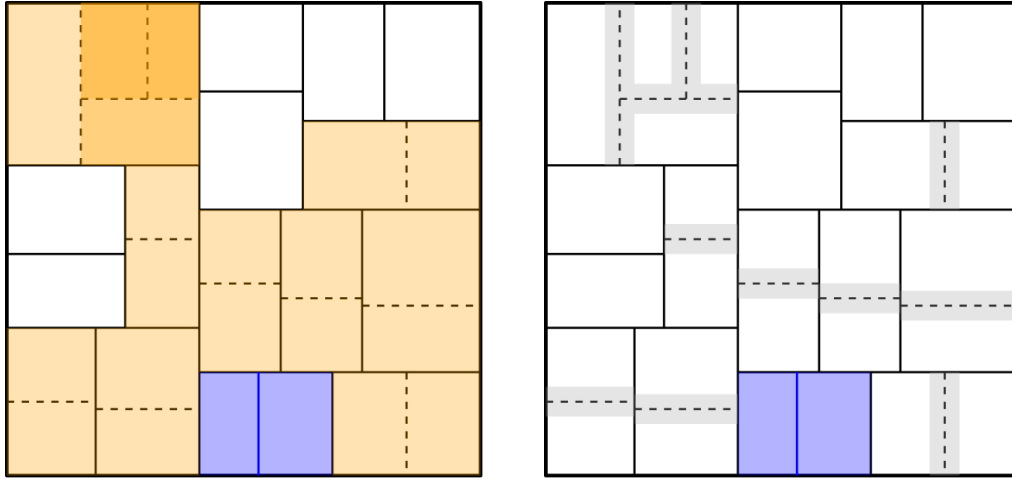


Figure 5 (left) The set \mathcal{C}_\square (the cells shaded in orange with dashed lines as their divider) for a cell \square (shaded in blue), and (right) any boundary-matched point in the gray area might cause a low-net-cost execution of the extended Hungarian search procedure on \square .

► **Lemma 15.** *For any non-leaf cell \square in \mathcal{H} and any cell $\square' \in \mathcal{C}_\square$, $\frac{1}{3}\ell_{\square'} \leq \ell_\square \leq \frac{9}{4}\ell_{\square'}$.*

For any cell $\square' \in \mathcal{C}_\square$, the merge step erases the divider $\Gamma_{\square'}$ and makes the points that are matched to $\Gamma_{\square'}$ free. Each of these new free points might cause \square to be selected by the algorithm. Therefore, to bound the number of low-net-cost selections of \square , we show that the total number of points that are matched to the dividers of the cells in \mathcal{C}_\square is at most $O(n^{4/5})$. For any cell $\square' \in \mathcal{C}_\square$, let $\mathcal{B}_{\square'}$ denote the set of points of B^C that are matched to the divider $\Gamma_{\square'}$ when our algorithm starts the merge step on \square' . Thus, we have to bound $\sum_{\square' \in \mathcal{C}_\square} |\mathcal{B}_{\square'}|$ by $O(n^{4/5})$. By invariant (I2), for each point $b \in \mathcal{B}_{\square'}$, $y(b) \leq \varphi \leq \ell_{\square'} n^{-1/5}$. Furthermore, by Condition (5), $y(b) = d(b, \square') = d(b, \Gamma_{\square'})$. Therefore, the point b is within a distance $\ell_{\square'} n^{-1/5}$ from the divider $\Gamma_{\square'}$. From Lemma 15, for each cell $\square' \in \mathcal{C}_\square$ and each point $b \in \mathcal{B}_{\square'}$, the point b is within a distance $\ell_{\square'} n^{-1/5} < \ell_{\square'} \lambda$ from the divider $\Gamma_{\square'}$, and from the construction of \mathcal{H} (Lemma 6), $|\mathcal{B}_{\square'}| = O(n_{\square'} n^{-1/5})$. Furthermore, from Lemma 15 and the construction of \mathcal{H} , each point $b \in B$ can lie inside a constant number of cells in \mathcal{C}_\square ; hence, $\sum_{\square' \in \mathcal{C}_\square} |\mathcal{B}_{\square'}| = O(\sum_{\square' \in \mathcal{C}_\square} n_{\square'} n^{-1/5}) = O(n^{4/5})$. Therefore, the total number of low-net-cost selections of \square by the extended Hungarian search procedure is $O(n^{4/5})$.

4 Conclusion and Open Questions

We presented an exact algorithm for the k -SP and k -SPI problems on d -dimensional point sets with bounded spread Δ , achieving sub-quadratic queries to a DWNN data structure and only logarithmic dependence on Δ . We highlight two open directions: First, can the $O(\log \Delta)$ dependence be removed? This would yield sub-quadratic algorithms for geometric bipartite matching, addressing a central open problem in computational geometry. Second, our reductions frame the k -server problem as a mild variant of online metric matching. This connects with extensive work on the online k -server problem [1, 3, 9, 10, 12, 14, 21, 23]. The reduction suggests simplifications and improvements to work function-based implementations, including the scalable algorithm by Raghvendra and Sowle [21]. Moreover, can we adapt the RM algorithm [15, 17, 18], which is the best-known online metric algorithm, to establish tighter bounds on its competitive ratio for k -server?

References

- 1 Sébastien Bubeck, Michael B Cohen, Yin Tat Lee, James R Lee, and Aleksander Mądry. K-server via multiscale entropic regularization. In *Proc. 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 3–16, 2018.
- 2 Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *Proc. 63rd IEEE Annual Sympos. Foundations of Computer Science*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.
- 3 M. Chrobak and L. L. Larmore. An optimal on-line algorithm for k servers on trees. *SIAM Journal of Computing*, 20(1):144–148, 1991. doi:10.1137/0220008.
- 4 Marek Chrobak, H Karloof, Tom Payne, and Sundar Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991. doi:10.1137/0404017.
- 5 H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal of Computing*, 18(5):1013–1036, October 1989. doi:10.1137/0218069.
- 6 Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for general graph-matching problems. *Journal ACM*, 38(4):815–853, 1991. doi:10.1145/115234.115366.
- 7 Akshaykumar Gattani, Sharath Raghvendra, and Pouyan Shirzadian. A robust exact algorithm for the Euclidean bipartite matching problem. *Advances in Neural Information Processing Systems*, 36, 2024.
- 8 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*, 64:838–904, 2020. doi:10.1007/S00454-020-00243-7.
- 9 E. Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. doi:10.1016/J.COSREV.2009.04.002.
- 10 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.
- 11 Harold W Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.
- 12 James R. Lee. Fusible HSTs and the randomized k-server conjecture. In Mikkel Thorup, editor, *Proc. 59th IEEE Annual Symposium on Foundations of Computer Science*, pages 438–449, 2018. doi:10.1109/FOCS.2018.00049.
- 13 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.
- 14 Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.
- 15 Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science*, pages 505–515, 2017. doi:10.1109/FOCS.2017.53.
- 16 Abhijeet Phatak, Sharath Raghvendra, Chittaranjan Tripathy, and Kaiyi Zhang. Computing all optimal partial transports. In *Proc. 11th Internat. Conference on Learning Representations*, 2022.
- 17 Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In *Approximation, Randomization, and Combinatorial Optimization*, 2016.
- 18 Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *34th International Symposium on Computational Geometry*, pages 67:1–67:14, 2018. doi:10.4230/LIPICS.SOCG.2018.67.
- 19 Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle. Geometric bipartite matching based exact algorithms for server problems, 2025. arXiv:2504.06079.
- 20 Sharath Raghvendra, Pouyan Shirzadian, and Kaiyi Zhang. A new robust partial p-Wasserstein-based metric for comparing distributions. In *41st Internat. Conference on Machine Learning*, 2024.

- 21 Sharath Raghvendra and Rachita Sowle. A scalable work function algorithm for the k-server problem. In *18th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 227, 2022. doi:10.4230/LIPICS.SWAT.2022.30.
- 22 Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. A fast implementation of the optimal off-line algorithm for solving the k-server problem. *Mathematical Communications*, 14(1):119–134, 2009.
- 23 Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. A fast work function algorithm for solving the k-server problem. *Central European Journal of Operations Research*, 21:187–205, 2013. doi:10.1007/S10100-011-0222-7.
- 24 Tomislav Rudec and Robert Manger. A new approach to solve the k-server problem based on network flows and flow cost reduction. *Computers & operations research*, 40(4):1004–1013, 2013. doi:10.1016/J.COR.2012.11.006.
- 25 R. Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proc. 29th Annual Symposium on Computational Geometry*, pages 9–16, 2013. doi:10.1145/2462356.2480283.
- 26 R Sharathkumar and Pankaj K Agarwal. Algorithms for the transportation problem in geometric settings. In *Proc. 23rd Annual ACM-SIAM Sympos. on Discrete Algorithms*, pages 306–317, 2012. doi:10.1137/1.9781611973099.29.
- 27 Pravin M Vaidya. Geometry helps in matching. *SIAM Journal of Computing*, 18(6):1201–1225, 1989. doi:10.1137/0218080.