A Faster Maximum Cardinality Matching Algorithm with Applications in Machine Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Maximum cardinality bipartite matching is an important graph optimization problem with several applications. For instance, maximum cardinality matching in a δ-disc graph can be used in the computation of several distance metrics between distributions that are popular in machine learning and statistics. These metrics include the bottleneck distance, the Wasserstein distance and the Lévy-Prokhorov distance. For any point sets $A, B \subset \mathbb{R}^2$, the δ-disc graph is a bipartite graph formed by connecting every pair of points $(a, b) \in A \times B$ by an edge if the Euclidean distance between them is at most δ. Using the classical Hopcroft-Karp algorithm, a maximum-cardinality matching on any δ-disc graph can be found in $\tilde{O}(n^{3/2})$ time. [1] In this paper, we present a simplification of a recent algorithm (Lahn and Raghvendra, JoCG 2021) for the maximum cardinality matching problem and describe how a maximum cardinality matching in a δ-disc graph can be computed asymptotically faster than $O(n^{3/2})$ time for moderately dense point sets. As applications, we show that if A and B are point sets drawn uniformly at random from a unit square, an exact bottleneck distance can be computed in $\tilde{O}(n^{4/3})$ time. On the other hand, experiments suggest that the Hopcroft-Karp algorithm seems to take roughly $\Theta(n^{3/2})$ time for this case. This translates to substantial improvements in execution time for larger inputs.

1 Introduction

The problem of finding a maximum cardinality matching is a fundamental graph optimization problem. With origins in economics and logistics, matchings have found numerous applications. Computing popular distances between distributions, such as the Wasserstein distance, the bottleneck distance and the Lévy-Prokhorov distance, can be reduced to a bipartite matching problem. In this paper, we consider the *δ-disc graph matching* which is the following:

Given two sets A and B of n two dimensional points and a parameter $\delta > 0$, a δ-disc graph G_δ is a bipartite graph obtained by connecting any pair of vertices $(a, b) \in A \times B$ with an edge provided the Euclidean distance between a and b is at most δ, i.e., $\|a - b\| \leq \delta$. Let m be the number of edges in the graph G_δ. A *matching* M is a set of vertex-disjoint edges in G_δ. In the δ-disc graph matching problem, we wish to compute a *maximum cardinality matching* in G_δ, i.e., a matching that has the largest number of edges.

Any algorithm for computing a δ-disc graph matching can be used to compute the bottleneck distance and the Lévy-Prokhorov distance. We define these distances next. Let $M \subset A \times B$ be any *perfect* matching of A and B, which is a matching where every vertex of A is matched, i.e., $|M| = n$.

[1] We use $\tilde{O}(\cdot)$ to suppress poly-logarithmic terms in the complexity.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

The edge of M with the largest Euclidean length is its bottleneck edge. The *bottleneck matching* is a perfect matching M^* whose bottleneck edge length is minimized. The Euclidean length of the bottleneck edge in M^* is the *bottleneck distance* between A and B.

Distances between distributions: Next, consider the case where A and B define discrete distributions $\mathcal{P}_A$ and $\mathcal{P}_B$ and let every point $a \in A$ (resp. $b \in B$) carry a probability of $1/n$, i.e., $\mathcal{P}_A(a) = 1/n$ (resp. $\mathcal{P}_B(b) = 1/n$). The ∞-Wasserstein distance between $\mathcal{P}_A$ and $\mathcal{P}_B$, denoted by $W_\infty(\mathcal{P}_A, \mathcal{P}_B)$, is simply the bottleneck distance between the point sets A and B. The *Lévy-Prokhorov* distance between $\mathcal{P}_A$ and $\mathcal{P}_B$ is defined as follows: For $\varepsilon > 0$ and any subset $X \subseteq A$ (resp. $Y \subseteq B$), let $X^\varepsilon = \{b \in B \mid \exists a \in X \text{ such that } \|a - b\| \leq \varepsilon\}$ (resp. $Y^\varepsilon = \{a \in A \mid \exists b \in Y \text{ such that } \|a - b\| \leq \varepsilon\}$). Note that $X^\varepsilon \subseteq B$ and $Y^\varepsilon \subseteq A$. We say that the *Lévy-Prokhorov* [30] distance $\pi(\mathcal{P}_A, \mathcal{P}_B)$ is equal to the smallest value of ε for which the following property is true:

$$\forall X \subseteq A, \mathcal{P}_A(X) \leq \mathcal{P}_B(X^\varepsilon) + \varepsilon \text{ and, } \forall Y \subseteq B, \mathcal{P}_B(Y) \leq \mathcal{P}_A(Y^\varepsilon) + \varepsilon;$$

here $\mathcal{P}_A(X) = \sum_{a \in X} \mathcal{P}_A(a) = |X|/n$ and $\mathcal{P}_B(Y) = \sum_{b \in Y} \mathcal{P}_B(b) = |Y|/n$. We can, therefore, write this condition as

$$\forall X \subseteq A, |X| \leq |X^\varepsilon| + \varepsilon n \text{ and } \forall Y \subseteq B, |Y| \leq |Y^\varepsilon| + \varepsilon n. \tag{1}$$

Determining if the exact bottleneck distance (equivalently the $W_\infty(\mathcal{P}_A, \mathcal{P}_B)$) is $\leq \delta$ can be done by simply finding the maximum cardinality matching M in G_δ. It is easy to see that M is perfect if and only if the bottleneck distance is at most δ. Similarly, by applying Hall's theorem, one can show that $\pi(\mathcal{P}_A, \mathcal{P}_B) \leq \varepsilon$ if and only if the maximum cardinality matching M in G_δ has a size of at least $(1 - \varepsilon)n$. Thus, the δ-disc graph matching directly relates to computing ∞-Wasserstein distance as well as the Lévy-Prokhorov distance between probability distributions.

Computing δ-disc graph matching: One can use any of-the-shelf matching algorithm [16, 27, 29, 35] to compute a maximum cardinality matching in a δ-disc graph. For instance, using the well-known Hopcroft-Karp algorithm will lead to an execution time of $O(m\sqrt{n})$ on any δ-disc graph. The HK-Algorithm executes in phases. Each phase takes $O(m)$ time and in the worst-case, the algorithm converges to a maximum matching in $O(\sqrt{n})$ phases. The best-known exact algorithm for computing the exact bottleneck matching combines geometric data structures with the HK-algorithm to reduce the execution time of each phase from m to $n\mathrm{poly}(\log \mathrm{n})$. As a result, they obtain an exact bottleneck matching in $\tilde{O}(n^{3/2})$ time.

Inspired by a series of algorithms for weighted matching in graphs with small separators [4, 23], Lahn and Raghvendra [24] presented a weighted approach to the maximum cardinality matching problem. This LR algorithm identifies a set of "edge separators" incident on ω "boundary vertices". These edges are assigned a weight of 1 and all other edges receive a weight of 0. A property satisfied by these separator vertices is that after their removal, every connected component in the graph has no more than r vertices. Then, they present an algorithm to compute a perfect matching in $\mathcal{O}(m\sqrt{r} + m\sqrt{w} + mrw/n \log n)$ time. As an application of their result, they show how to compute the bottleneck distance of any point sets A and B within a multiplicative factor of $(1 + \varepsilon)$ in $\tilde{O}(n^{4/3}/\mathrm{poly}(1/\varepsilon))$ time. The LR algorithm assigns dual weights to vertices and is similar in style to the Kuhn-Munkres [21] and the Gabow-Tarjan [13] algorithms. The dual weights on vertices play a vital role in the proof of correctness and efficiency of the LR algorithm.

In this paper, we make the following contributions:

- We remove the need to maintain dual weights in the LR algorithm resulting in a significantly simpler algorithm.
- Using this algorithm, we show how to find a maximum cardinality matching in a unit-disc graph G_δ in time $\tilde{O}(n^{4/3}k^{1/3})$ where k is the maximum number of points of $A \cup B$ contained in any disc of radius δ. Note that our algorithm is superior to the classical Hopcroft-Karp based algorithm when $k = o(\sqrt{n})$.
- Using our algorithm for δ-disc graph matching, we show how to compute the exact bottleneck distance between point sets A and B. When $\mathcal{P}_A$ and $\mathcal{P}_B$ are distributions with each point having probability $1/n$, the bottleneck distance can be used to compute the distances $W_\infty(\mathcal{P}_A, \mathcal{P}_B))$ and $\pi(\mathcal{P}_A, \mathcal{P}_B)$. We are not aware of any previous polynomial time algorithms to compute the Lévy-Prokhorov distance. When A, B are chosen uniformly

at random from a unit square, our algorithm for the exact bottleneck distance runs in $\tilde{O}(n^{4/3})$ time. All previous algorithms take $\Omega(n^{3/2})$ time.

- We run experiments for the case where A and B are chosen uniformly at random from a unit square. Our experiments suggest that the Hopcroft-Karp algorithm on G_δ takes $\Theta(n^{3/2})$ time. In contrast, our algorithm runs in roughly $\tilde{O}(n^{4/3})$ time. This translates to substantial improvements in execution time.

Note that, for the HK-algorithm, the upper bound of $O(\sqrt{n})$ phases is only in the worst-case. As noted by Motwani [28], the Hopcroft-Karp algorithm converges, with high probability, to a maximum matching in $O(\log n)$ phases for expander graphs in general and Erdős-Rényi random graphs in particular. Similarly, can we show asymptotically fewer iterations than $\sqrt{n}$ for the bottleneck matching problem for the HK algorithm? Interestingly, our experiments suggest that the answer to this question may be in the negative. When A and B are drawn uniformly at random from a unit square and when δ is set to the bottleneck distance, the number of phases seem to grow at the rate of $\Omega(\sqrt{n})$. Therefore, bottleneck matching on random point sets may represent a natural hard instance for the HK-Algorithm. In this paper, we show how to overcome the $\Omega(n^{3/2})$ barrier for uniformly distributed point sets by developing an $\tilde{O}(n^{4/3})$ time algorithm.

Applications: Wasserstein distance has found numerous applications [3, 5, 6, 8, 12, 32]. Due to these applications, computing approximations of Wasserstein distances has received substantial attention [2, 7, 10, 22, 26, 31]. However, exact algorithms (even for discrete distributions) have a relatively high execution time [13, 21, 35]. The Lévy-Prokhorov distance have been extensively studied for its theoretical properties [9, 34]. For instance, Lévy-Prokhorov distance metrizes weak convergence on any separable metric space [17]. However, a brute-force algorithm based on the definition of this metric will require a search on exponentially many possible subsets causing it to seldom be used in practice [14]. Somewhat surprisingly, we show in this paper that computing the Lévy-Prokhorov metric reduces to the δ-disc graph matching problem and so, it is only as hard as computing the ∞-Wasserstein distance, at least for discrete distributions of the type $\mathcal{P}_A$ and $\mathcal{P}_B$ described above. We are not aware of any prior mention of this reduction in the literature.

In topological data analysis, bottleneck distance is the primary metric for comparing *persistence diagrams* [15, 20, 1]. This has lead to development of practical implementations of bottleneck matching algorithms [15, 20]. The special case of computing bottleneck matching of point sets A and B that are drawn uniformly at random from a unit square have also received considerable attention. For instance, it has been used in the context of testing pseudo-random generators, average case analysis of bin packing algorithms [25], and also in statistics for analyzing the Glivenko-Cantelli convergence of empirical measures [33]. δ-disc graphs have other applications as well including in the modeling of the topology of ad-hoc wireless networks [18]. Matchings in δ-disc graphs can be used to match clients to servers in wireless networks. In the context of logistics, it can also be used to match requests to taxis that are within a fixed radius.

2 Matching algorithms

In this section, we present and compare two algorithms for solving the maximum cardinality matching problem on an arbitrary graph: The Hopcroft-Karp (HK) algorithm [16], and the LR algorithm [24]. In section 2.1, we introduce the basic definitions used by most combinatorial matching algorithms and give an overview of the HK algorithm. In section 2.2, we present the LR algorithm, highlighting the differences it has from the HK algorithm.

2.1 Preliminaries

Given any matching M, let A_F and B_F denote the vertices of A and B respectively that are not matched in M. We refer to these vertices as *free* vertices. An *alternating path* P is a path that alternates between edges that are in the matching and those that are not in the matching. An *augmenting path* is an alternating path that starts and ends at a free vertex. We define the *length* of P as the number of edges in P.

We can augment a matching M along an augmenting path P by updating the matching as follows: $M \leftarrow M \oplus P$; here $\oplus$ denotes the set symmetric difference operator. It is easy to see that augmenting

a matching along an augmenting path P increases the size of the matching M by 1. Furthermore, it can be shown that G has no remaining augmenting paths with respect to a matching M if and only if M has maximum cardinality. These observations are the basis of the following commonly-used approach for computing a maximum-cardinality matching: repeatedly compute an augmenting path P with respect to M and augment M along P until M has maximum cardinality. Since the largest possible matching has size at most n, any such algorithm will arrive at a maximum-cardinality matching after $O(n)$ augmentations. This is the approach used by the classical Ford-Fulkerson and HK algorithms as well as the recent LR algorithm. However, the algorithms differ in the details of *how* these augmenting paths are found.

Residual graph: Given a matching M, the *residual graph* G_M is a directed graph that assists in finding augmenting paths. The graph G_M contains the same set of vertices V as G. For any edge (a, b) in G, if $(a, b) \in M$ then we add an edge directed from a to b in G_M. Otherwise, we add an edge directed from b to a in G_M. Furthermore, we create a source vertex s and a sink vertex t. For each free vertex $b \in B_F$, we add an edge (s, b) directed from s to b in G_M and for each free vertex $a \in A_F$ we add an edge from a to t in G_M. Note that, for the residual graph, we use (u, v) to denote an edge directed from u to v. On the other hand, for the undirected graph G, we may use (u, v) and (v, u) interchangeably to represent the same edge between vertices u and v. Also note that, in a residual graph, any directed path from a free vertex $a \in B_F$ to a free vertex $a \in A_F$ is an augmenting path. In the Ford-Fulkerson algorithm, a single augmenting path can be found in $O(m)$ time using any common graph search algorithm such as breadth-first search (BFS) or depth-first search (DFS), leading to an $O(mn)$ time algorithm. The HK and LR algorithms both improve upon this running time by finding potentially many augmenting paths in each iteration.

HK algorithm: Initially, let $M = \emptyset$. The HK algorithm executes in *phases*. A phase is divided up into two *stages*. The first stage executes a BFS starting from s and identifies the length of the shortest path (path with the fewest edges) in G_M from s to every other vertex in G_M. Let ℓ_u be the length of the shortest path from s to u and let $\ell = \ell_t$. The algorithm then computes an *admissible graph* $\mathcal{A}$ consisting of all edges (u, v) in G_M such that (a) ℓ_u and ℓ_v are at most ℓ and (b) $\ell_v = \ell_u + 1$. We note that these edges capture the set of all minimum-length augmenting paths in G_M. The second stage iteratively conducts multiple *partial DFS* that start from s and that terminate when a path to t is found. Between the termination of one DFS and the start of the following DFS, all edges visited by first DFS are removed from the residual graph. It can be shown that in each phase the HK algorithm finds a maximal set of vertex-disjoint shortest augmenting paths in G_M. Each phase involves execution of a single BFS and multiple partial DFS. Since no two executions of DFS visit the same edge, the combined execution time of the multiple partial DFS is bounded by $O(m)$.

Hopcroft and Karp showed that, after executing a phase, the length of the shortest augmenting path increases by at least 1. Therefore, after $\sqrt{n}$ phases, the shortest augmenting path has length at least n. After this point, it can be shown that the number of augmenting paths found by the algorithm is only $O(\sqrt{n})$. Since the algorithm finds at least one augmenting path each phase, the total number of phases is bounded by $O(\sqrt{n})$. Since each phase takes $O(m)$ time, the total time taken by the algorithm is $O(m\sqrt{n})$. A surprising fact established by Hopcroft and Karp [16] is that the total length of all the n augmenting paths across all phases is $O(n \log n)$.

2.2 A simplified implementation of the LR algorithm

Recently, Lahn and Raghvendra presented an algorithm [24] that can be seen as a variant of the HK algorithm. Apart from the bipartite graph $G(V, E)$, we are also given a subset $E_S \subseteq V$ of "separator edges" as input. For any separator edge $(u, v) \in E_S$, we denote the vertices u and v as *boundary vertices*. Let B be the set of all boundary vertices. The analysis of the algorithm depends on $\omega = |B|$ and another parameter r that is defined next. Consider the graph $G'(V, E \setminus E_S)$. Let $\mathbb{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_t\}$ be the set of connected components of G' and let $\mathcal{V}_i$ and $\mathcal{E}_i$ be the set of vertices and edges of $\mathcal{P}_i$. We refer to each $\mathcal{P}_i$ in G' as a *piece* of the original graph G. Let $r = \max_{\mathcal{P}_i \in \mathbb{P}} |\mathcal{V}_i|$, i.e., the size of the piece of G with the largest number of vertices.

Setting weights on the edges of the graph G and its residual graph G_M: For any edge $(u, v) \in E$, we assign it a *weight* $w(u, v)$. For any separator edge $(u, v) \in E_S$, we set $w(u, v)$ to 1. For any other edge $(u', v') \in E \setminus E_S$, we set $w(u', v')$ to 0. Every edge (u, v) in the residual graph inherits the weight of the corresponding edge (u, v) in $G(V, E)$. All edges incident on the source s and the sink t in G_M receive a weight of 0.

Preprocessing: For each piece, the HK algorithm is used to find the maximum cardinality matching on each piece. Let M be the union of the matchings computed for every piece and let M^* be a maximum cardinality matching in G. At the end of this step, the difference $|M^*| - |M|$ is $O(\omega)$, i.e., the number of augmenting paths the algorithm needs to find is $O(\omega)$.

The remaining $O(\omega)$ unmatched vertices are subsequently matched by executing *phases*. Like the HK algorithm, each phase consists of two *stages*. These stages somewhat resemble the stages of the HK algorithm. We highlight the differences in the description below.

Stage 1: In the first stage, the algorithm finds, for any vertex $v \in V$, the minimum cost path from s to v in the residual graph G_M for the weights $w(\cdot, \cdot)$. Note that, since all edge weights are either 0 or 1, a standard BFS implementation can be modified to support such a minimum-weight search algorithm in $O(m)$ time by simply prioritizing edges of weight 0 over edges of weight 1. We call this modified version of BFS, *0/1 BFS*. For any vertex $v \in V$, let ℓ_v be the distance from s to v in G_M as computed by the $0/1$ BFS and let $\ell = \ell_t$. Any edge (u, v) of G_M is *admissible* if $\ell_u, \ell_v \leq \ell$ and $\ell_v = \ell_u + w(u, v)$. The *admissible* graph $\mathcal{A}$ is identical to G_M, except it contains only admissible edges. Similar to HK algorithm, it can be shown that the admissible graph $\mathcal{A}$ captures every minimum-cost augmenting path.

Stage 2: The second stage of the LR algorithm, much like the second stage of the HK algorithm, finds a set of shortest augmenting paths (by weight). It does so by iteratively conducting partial-DFS from s until no augmenting path is found. Each execution of the partial-DFS terminates when an augmenting path P is found and the matching M is immediately augmented along P. Let $\mathcal{K}$ be the set of *affected pieces* that contain at least one edge from P. Before the start of the next partial-DFS, the algorithm deletes all edges of the admissible graph that were visited by the previous partial-DFS except for those edges that belong to an affected piece. In other words, edges in an affected piece can be visited multiple times within the same phase.

Differences with HK algorithm: The main differences between the LR and HK algorithms are:

(1) The LR algorithm assigns weights of 0 and 1 to the edges. No weights are assigned in the HK algorithm.

(2) The LR has a preprocessing step that computes the maximum cardinality matching for each piece.

(3) In Stage 1, the LR algorithm executes a $0/1$-BFS instead of the BFS executed by HK algorithm.

(4) In Stage 2, the partial-DFS reuses edges from affected pieces. As a result, in each phase, the LR algorithm finds augmenting paths that are not necessarily vertex-disjoint.

Differences (1) – (3) between HK algorithm and the LR algorithm do not impact the execution time of the algorithm by any more than a small constant factor. See Section F of the appendix for a discussion on this. The critical difference between the two algorithms is (4). Unlike the HK algorithm, Stage 2 of the LR algorithm reuses edges from affected pieces and computes a set of augmenting paths that are not necessarily vertex-disjoint. This allows for computing many more augmenting paths within each phase. As shown in [24], after each phase, the weight of the shortest augmenting path increases by at least one and after $\sqrt{\omega}$ phases, there are no more than $O(\sqrt{\omega})$ free vertices remaining. Each of the remaining free vertices can be matched in at most $O(\sqrt{\omega})$ phases. Thus the total number of phases can be bounded by $O(\sqrt{\omega})$.

On the other hand, allowing for edge revisits within each phase increases the execution time of each phase. Note that an edge can be revisited only if it was inside an affected piece when it was previously visited. Therefore, one can upper-bound the total number of edge revisits across the entire algorithm by the total number of edges present in the affected pieces (edges of a piece that is affected k times will be counted k times in this sum). Lahn and Raghvendra show that the total cost of all the augmenting paths computed by the algorithm is $O(\omega \log \omega)$. Using this they upper-bound the total number of affected pieces by $O(\omega \log \omega)$. For graphs that admit recursive separators (such as planar and graphs with excluded minors), they show that the number of edges for any piece can be bounded by $O(mr/n)$ leading to an $O(\frac{mr\omega}{n} \log n)$ bound on the total number of revisits.

Theorem 1. *Consider a graph G and a set of separator edges. Suppose each piece has at most $O(mr/n)$ edges. A maximum cardinality matching can be computed in $O(m\sqrt{r} + m\sqrt{\omega} + \frac{mr\omega}{n} \log n)$ time.*

The assumption on an upper bound on the number of edges within each piece can be eliminated when graphs, such as those considered in this paper, support a dynamic data structure $\mathcal{D}$ of the following form: $\mathcal{D}$ can store any subset $A' \subseteq A$ of vertices and, given any query vertex $b \in B$, it can return a vertex $a \in A'$ that minimizes the cost of the edge (b, a). If no edge exists between b and any vertex of A', then the data structure returns NULL. Suppose that $\mathcal{D}$ supports arbitrary insertions and deletions from A', as well as queries, each in $\Phi(n)$ time. Then, one can use the data structure, maintaining A' as the set of unvisited nodes of A during the BFS and DFS. Consequently, one can execute 0/1 BFS and DFS in time $O(n\Phi(n))$. As a result, the execution time of the algorithm can be improved to $O(n\Phi(n)\sqrt{r} + n\Phi(n)\sqrt{\omega} + r\omega\Phi(n)\log n)$.

Theorem 2. *Given a graph G that supports a dynamic nearest neighbor data structure with query and update time of $\Phi(n)$, a maximum cardinality matching can be computed by the* LR *algorithm in $O(\Phi(n)(n\sqrt{r} + n\sqrt{\omega} + r\omega\log n))$. The HK algorithm computes a maximum cardinality matching in $O(n^{3/2}\Phi(n))$ time.*

Equivalency to original LR algorithm: The original algorithm maintains a dual weight $y(v)$ for every vertex $v \in A \cup B$ at any point during the algorithm. For any edge $(a, b) \in (A \times B) \cap E$, the dual weights satisfy the following:

$$y(b) - y(a) \leq w(a, b) \qquad \text{if } (a, b) \notin M, \tag{2}$$

$$y(a) - y(b) = w(a, b) \qquad \text{if } (a, b) \in M. \tag{3}$$

Additionally, their algorithm maintains the invariants that all free vertices of B_F have the same dual weight of $y_{\max} = \max_{v \in A \cup B} y(v)$ and all free vertices of A have the same dual weight of 0. The *slack* $s(a, b)$ of any edge $(a, b) \in (A \times B) \cap E$ is defined as follows: if $(a, b) \notin M$, then $s(a, b) = w(a, b) - y(b) + y(a)$; otherwise, $(a, b) \in M$ and $s(a, b) = 0$.

In the original LR algorithm, the first Stage adjusts the dual weights so that there is a zero-slack augmenting path in G_M. The second stage repeatedly executes DFS from s to t on the set of zero-slack edges in order to find augmenting paths. After augmenting along a path, all edges visited during the previous DFS are deleted, unless they were in an affected piece.

Note that the only fundamental difference between the original version of the LR algorithm and our simplified version is the fact that we compute minimum-weight augmenting paths while they compute zero-slack augmenting paths. The following lemma, whose proof appears in Section A of the appendix, shows that the a zero slack path computed in the LR algorithm is also a minimum weight path. It follows that the two versions of the algorithm are equivalent.

Lemma 1. *During Stage 2 of* LR *algorithm, an augmenting path has zero slack if and only if it has minimum weight.*

3 Applications

In this section, we show how the LR algorithm can be applied to efficiently compute a maximum cardinality matching on a δ-disc graph, an optimal bottleneck matching, as well as the Lévy-Prokhorov distance between distributions. All applications considered are with regards to point sets $A, B \subset \mathbb{R}^2$. For all applications, one can construct a dynamic data structure that supports insertion, deletion and nearest neighbor queries in $\Phi(n) = \log^{O(1)} n$ time, which can be used in the application of Theorem 2.

3.1 δ-disc graph matching

Let $P = A \cup B$. Let $\mathbb{B}(p)$ be a ball centered at p with radius δ. Consider $k = \max_{p \in \mathbb{R}^2} |\mathbb{B}(p) \cap P|$, i.e., k is the largest number of points of $A \cup B$ inside any ball of radius δ. We refer to k as the *δ-density* of the point set P. We show that a maximum cardinality matching in a δ-disc graph can be computed using the LR algorithm in $\tilde{O}(n^{4/3}k^{1/3})$ time. Thus, when the δ-density $k = o(\sqrt{n})$, the LR algorithm outperforms the HK algorithm.

Theorem 3. *For any point set $P = A \cup B$ and a parameter $\delta > 0$, a maximum cardinality matching in the δ-disc graph defined on P can be computed in $\tilde{O}(n^{4/3}k^{1/3})$ time, where k is the δ-density of P.*

This result also extends to the case where the points of A and B are independently and identically distributed random variables drawn from distributions $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively. We say that a distribution $\mathcal{P}$ has a δ-density of k if, for any ball $\mathbb{B}(p)$ of radius δ, the probability that a point drawn from $\mathcal{P}$ lies inside the ball is at most k/n.

Theorem 4. *Let A, B be drawn iid from distributions $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively. For a parameter $\delta > 0$, a maximum cardinality matching in the δ-disc graph defined on $A \cup B$ can be computed, with high probability, in $\tilde{O}(n^{4/3}k^{1/3})$ time, where k is the maximum of the δ-density of $\mathcal{P}_A$ and $\mathcal{P}_B$.*

Proof of Theorem 3: We show how a set of separator edges can be generated so that $\omega = O(n^{2/3}k^{2/3})$ and $r = O(n^{2/3}/k^{1/3})$. From Theorem 2 and since $n\sqrt{r} = O(n^{4/3}/k^{1/6})$, $n\sqrt{\omega} = O(n^{4/3}k^{1/3})$, and, $r\omega = O(n^{4/3}k^{1/3})$, the execution time of the LR algorithm can be bounded by $\tilde{O}(n^{4/3}k^{1/3})$. Next, we describe how to generate the separator edges E_S.

We use a *grid* $\mathbb{G}$ to generate the separator edges. Any grid consists of a set of equispaced horizontal and vertical lines that partition $\mathbb{R}^2$ into *cells*. Each cell is a square and any grid can be seen as a set of these squares. Let $\mathcal{L}(\mathbb{G})$ denote the side-length of any cell $C \in \mathbb{G}$. We say that a cell C is *non-empty* if $C \cap P \neq \emptyset$. Let $\theta = \lceil n^{1/3}/k^{2/3} \rceil$. To generate our pieces, we choose a grid $\mathbb{G}$ where the side-length of each cell is set to $\mathcal{L}(\mathbb{G}) = \theta\delta$. The separator edge set E_S consists of all edges of the δ-disc graph that have their endpoints in different cells. All such edges are assigned a weight of 1. Any edges whose endpoints are contained within the same cell of $\mathbb{G}$ are in $E \setminus E_S$ and are assigned a weight of 0. Any point that has at least one separator edge incident on it becomes a boundary vertex. To generate $\mathbb{G}$, we check $O(\theta)$ possible vertical and horizontal shifts and pick the one that minimizes the number of boundary vertices. We provide the details of generating $\mathbb{G}$ in Section B.1 of the appendix. Our choice of $\mathbb{G}$ guarantees that $\omega = O(n^{2/3}k^{2/3})$ for any point set, independent of its δ-density. We show this in Section B.2 of the appendix.

Bounding r: By this definition, the number of vertices of any piece is bounded by the maximum number of points that can lie inside any cell of $\mathbb{G}$, i.e., $\max_{C \in \mathbb{G}} |C \cap P|$. Note that we can cover any cell of $\mathbb{G}$ with $\Theta(\theta^2)$ balls of radius δ each. Due to the δ-density of P being k, each of these balls can contain at most k points and the total number of points inside any cell can be bounded by $O(\theta^2 k) = O(n^{2/3}/k^{1/3})$ as desired. In other words, $r = O(n^{2/3}/k^{1/3})$.

Proof of Theorem 4: The construction of the grid here will be identical to the one in the proof of Theorem 3. Note also that the bound on ω provided in that proof depends only on the construction of $\mathbb{G}$ and not on the δ-density of the point set. Therefore, the same bound continues to hold here as well. In Section C of the appendix, we use the δ-density of $\mathcal{P}_A$ and $\mathcal{P}_B$ along with Chernoff's bound to prove that $r = O(n^{2/3}/k^{1/3})$ points with high probability.

3.2 Bottleneck distance

In this section, we assume that A and B are points drawn uniformly at random from a unit square. From the work of Shor [25], we know that, for appropriate constants $c_{\min}$ and $c_{\max}$, the optimal bottleneck distance is at least $\delta_{\min} = \frac{c_{\min}\log^{3/4}(n)}{\sqrt{n}}$ and at most $\delta_{\max} = \frac{c_{\max}\log^{3/4}(n)}{\sqrt{n}}$ with very high probability (probability exceeding $1 - 1/n^{\alpha}$ for some $\alpha = \Omega(\sqrt{\log n})$). Observe that the optimal bottleneck distance will be equal to the length of some edge of $A \times B$. As in the work of Efrat *et al*. [11], our algorithm will use a selection algorithm of Katz and Sharir [19] to find the jth smallest edge, $1 \leq j \leq n^2$ in $O(n^{4/3}\log^2 n)$ time. Let $d(j)$ be the length of the jth smallest edge returned by their algorithm. This allows us to execute a binary search over the edges of $A \times B$, ordered by their length. Let $g_{\min} = 1$ and $g_{\max} = n^2$. We repeat the following process until $g_{\max} = g_{\min} + 1$: We choose $j = \lfloor (g_{\max} + g_{\min})/2 \rfloor$ and find the jth smallest edge whose length is denoted by $d(j)$. If $d(j) \geq \delta_{\max}$, we set $g_{\max} \leftarrow j$. If $d(k) \leq \delta_{\min}$, we set $g_{\min} \leftarrow j$. Otherwise, $\delta_{\min} \leq d(j) \leq \delta_{\max}$, and we find the maximum cardinality matching in a δ-disc graph where δ is set to $d(j)$. If we obtain a perfect matching, we set $g_{\max} = j$. Otherwise, the maximum matching is not perfect, and we set $g_{\min} = j$. When the algorithm terminates, $d(g_{\max})$ is the optimal bottleneck distance.

Analysis: The algorithm makes $O(\log n)$ many guesses. These guesses are found by a selection algorithm that runs in $O(n^{4/3}\log^2 n)$ time. For each guess j where $\delta_{\min} \leq d(j) \leq \delta_{\max}$, we must compute a maximum-cardinality matching on a δ-disc graph, which takes $O(n^{4/3}k^{1/3})$ time using

the LR algorithm. Since, $\mathcal{P}(A)$ and $\mathcal{P}(B)$ are the uniform distribution, their δ-density increases as δ increases. Therefore, the $\delta_{\max}$-density of $\mathcal{P}(A)$ and $\mathcal{P}(B)$ will be an upperbound on the δ-density for any execution of the LR algorithm. The probability that any random point lies within any ball of radius $\delta_{\max}$ is at most $(\delta_{\max})^2$, which is $\Theta(\log^{3/2}(n)/n)$. Therefore, the $\delta_{\max}$-density of $\mathcal{P}(A)$ and $\mathcal{P}(B)$ is at most $k = O(\log^{3/2}(n))$. Applying Theorem 4 gives the following:

Theorem 5. *Let A, B be drawn uniformly at random from a unit square. An optimal bottleneck matching can be computed between A and B, with high probability, in $\tilde{O}(n^{4/3})$ time.*

Practical considerations: The algorithm described uses two black-boxes that are impractical and have hidden high constants in the Big-O notation. (a) the algorithm relies on a dynamic nearest neighbor data structure, and, (b) the algorithm uses the selection algorithm of Katz and Sharir [19]. In Section D of the appendix, we address both (a) and (b) by presenting more practical alternatives.

3.3 Lévy-Prokhorov distance

We show that δ-disc graph matching can be applied to the computation of Lévy-Prokhorov distance.

We describe a simple algorithm to decide if $\pi(\mathcal{P}_A, \mathcal{P}_B)$ greater than or at most ε. We compute a maximum cardinality matching M in an ε-disc graph. Let A_F and B_F be the free vertices with respect to M. Then, we say that the distance is greater than ε if $|A_F| > \varepsilon n$. Otherwise, we say that the distance is at most ε. Using a binary search similar to the one described in Section 3.2, we can determine the distance in $\tilde{O}(n^{4/3}k^{1/3})$ time.

Proof via Hall's theorem: Given any bipartite graph $G(A \cup B, E)$, for any set $X \subseteq A$, the neighborhood $\mathcal{N}(A)$ is the set of all vertices of B that share an edge with at least one vertex of A. Thus, X^ε is the neighborhood of X in an ε-disc graph. The deficiency of a graph with respect to A is $\mu(A) = \max_{X \subseteq A} |X| - \mathcal{N}(X)$. Hall's theorem says that a bipartite graph has a perfect matching if and only if the deficiency of the graph with respect to A is non-positive. Hall's theorem can be used to prove the following general claim

Lemma 2. *For any bipartite graph $G(A \cup B, E)$, where $|A| = |B| = n$, and for any integer $k > 0$, the deficiency with respect to A or B is k if and only if the maximum cardinality matching is of size $n - k$.*

The proof of Lemma 2 follows in a straight-forward way from the Hall's theorem. For the sake of completion, we provide this proof in Section E of the appendix. Recollect that, our algorithm returns the distance to be greater than ε if $|A_F| > \varepsilon n$. By Lemma 2, we conclude that the deficiency of the graph is greater than εn, i.e., there is a set $X \subseteq A$ such that $|X| - |X^\varepsilon| > \varepsilon n$. Thus, Equation 1 does not hold and the distance is greater than ε.

Our algorithm returns a distance at most ε if $|A_F| \leq \varepsilon n$. In this case, from Lemma 2, the deficiency of the graph with respect to A is less than εn, i.e., for every subset $X \subseteq A$, $|X| - |X^\varepsilon| \leq \varepsilon n$. Note that $|A_F| = |B_F| \leq \varepsilon n$ and so an identical argument applies for B as well. Thus, Equation 1 holds and the distance is at most ε. We conclude that the algorithm terminates with the correct ε.

4 Experimental results

In this section, we compare the performance of the HK and LR algorithms when applied to computing an exact bottleneck matching between equal-sized point sets $A, B \subset \mathbb{R}^2$ drawn uniformly at random from a unit square, where $n = |A| + |B|$.

Experimental setup: For each value of n in $\{100, 1000, 5000, 10000, 50000, 100000, 500000, 1000000, 1500000\}$, we execute 10 runs. For each run, we uniformly sample points from a unit square to obtain the point sets A and B. Next, we compute a bottleneck matching between A and B separately, using both the HK and LR algorithms, and record performance metrics for both algorithms. We execute our experiments on a server running CentOS Linux 7, with 12 Intel E5-2683v4 cores and 128GB of RAM.

When guessing the bottleneck distance for each run, instead of enforcing that the number of guesses is $O(\log n)$ it is sufficient in practice to continue the binary search on δ until the relative error becomes less than a sufficiently small value ε (see Section D of the appendix). Both the HK-based

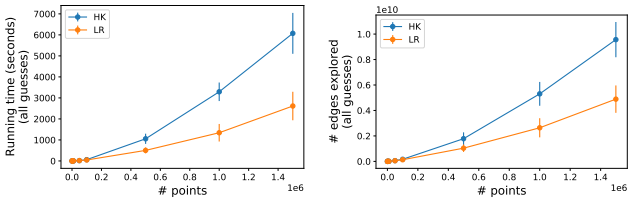

Figure 1: A running time comparison between the HK and LR algorithms. Left: Comparison of actual running time. Right: Comparison of total number of edge visits.

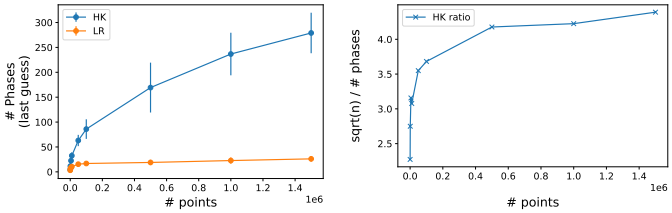

Figure 2: Data for number of phases for the final bottleneck guess. Left: A comparison of the number of phases for the HK and LR algorithms. Right: $\sqrt{n}$ divided by the number of phases executed by the HK algorithm.

algorithm and the LR-based algorithm use the same strategy for guessing the bottleneck distance in the experiments.

Experimental results: For each figure, the data presented for each value of n is averaged over all 10 runs. Error bars represent a single standard deviation. Figure 1 presents the actual running time of both algorithms, summed over all guesses of the bottleneck distance. For datasets with more than 10^6 points, the LR algorithm takes roughly half as much time as the HK algorithm and the gap seems to grow as the input size increases. However, the actual running times can be affected by several factors including the exact implementation details and execution environment. Therefore, we focus on comparing metrics that are accurate independent of the exact implementation details. Recall that both algorithms combine variants of BFS and DFS to compute augmenting paths. As a result, the total number times edges are visited during each algorithm acts as an implementation-independent proxy of the running time. Figure 1 shows the total number of edge visits for both algorithms. Note that this data seems to follow a similar trend to the actual running times of the algorithms.

Next, we summarize our observations that help account for this difference in performance of the two algorithms. For more details, see Section G of the appendix. Recall that there are four main differences (1) – (4) between the HK and LR algorithms. As discussed in Section F of the appendix, differences (1) – (3) do not have any direct significant impact on the relative running times of the two algorithms; the most significant difference is (4) – Stage 2 of the LR algorithm reuses edges from affected pieces. This reuse of edges has two main effects on the efficiency of the LR algorithm. First, we find that the LR algorithm executes significantly fewer phases than the HK algorithm. Specifically, as the guess of the bottleneck distance approaches the actual bottleneck distance, our results suggest that the number of phases executed by the HK algorithm seems to grow at a rate of $\Theta(\sqrt{n})$ – exhibiting its worst-case analysis. In contrast, the number of phases executed by the LR algorithm grows at a much slower rate (see Figure 2). This explains why the LR algorithm runs faster than the HK algorithm. The second impact of allowing for edge revisits is that a single edge can be revisited, perhaps many times, during a single phase. Despite this, the total number of edges visited by the LR algorithm is still significantly less than the total number of edges visited by the HK algorithm (see Figure 1).

References

[1] A. Adcock, D. Rubin, and G. Carlsson, Classification of hepatic lesions using the matching metric, *Computer vision and image understanding*, 121 (2014), 36–42.

[2] J. Altschuler, J. Weed, and P. Rigollet, Near-linear time approximation algorithms for optimal transport via sinkhorn iteration, *Neural Information Processing Systems*, 2017, pp. 1961–1971.

[3] M. Arjovsky, S. Chintala, and L. Bottou, Wasserstein GAN, *arXiv:1701.07875v3 [stat.ML]*, (2017).

[4] M. K. Asathulla, S. Khanna, N. Lahn, and S. Raghvendra, A faster algorithm for minimum-cost bipartite perfect matching in planar graphs, *ACM Trans. Algorithms*, 16 (2020), 2:1–2:30.

[5] J.-D. Benamou, G. Carlier, M. Cuturi, L. Nenna, and G. Peyré, Iterative bregman projections for regularized transportation problems, *SIAM Journal on Scientific Computing*, 37 (2015), A1111–A1138.

[6] J. Bigot, R. Gouet, T. Klein, A. López, et al., Geodesic PCA in the wasserstein space by convex PCA, *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, Vol. 53, Institut Henri Poincaré, 2017, pp. 1–26.

[7] J. Blanchet, A. Jambulapati, C. Kent, and A. Sidford, Towards optimal running times for optimal transport, *arXiv:1810.07717 [cs.DS]*, (2018).

[8] M. Cuturi and A. Doucet, Fast computation of wasserstein barycenters, *International Conference on Machine Learning*, 2014, pp. 685–693.

[9] R. M. Dudley, Distances of probability measures and random variables, *The Annals of Mathematical Statistics*, 39 (1968), 1563–1572.

[10] P. Dvurechensky, A. Gasnikov, and A. Kroshnin, Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm, *International Conference on Machine Learning*, 2018, pp. 1366–1375.

[11] A. Efrat, A. Itai, and M. J. Katz, Geometry helps in bottleneck matching and related problems, *Algorithmica*, 31 (2001), 1–28.

[12] R. Flamary, M. Cuturi, N. Courty, and A. Rakotomamonjy, Wasserstein discriminant analysis, *Machine Learning*, 107 (2018), 1923–1945.

[13] H. N. Gabow and R. E. Tarjan, Faster scaling algorithms for network problems, *SIAM Journal on Computing*, 18 (1989), 1013–1036.

[14] A. L. Gibbs and F. E. Su, On choosing and bounding probability metrics, *International statistical review*, 70 (2002), 419–435.

[15] F. Godi, Bottleneck distance, in: *GUDHI User and Reference Manual*, GUDHI Editorial Board, 3.4.1 edition, 2021.

[16] J. E. Hopcroft and R. M. Karp., An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs., *SIAM Journal on Computing*, 2 (1905), 1973–231.

[17] P. J. Huber, *Robust Statistics*, New York: John Wiley and Sons, 1981.

[18] M. L. Huson and A. Sen, Broadcast scheduling algorithms for radio networks, *Proceedings of MILCOM'95*, Vol. 2, IEEE, 1995, pp. 647–651.

[19] M. J. Katz and M. Sharir, An expander-based approach to geometric optimization, *SIAM Journal on Computing*, 26 (1997), 1384–1408.

[20] M. Kerber, D. Morozov, and A. Nigmetov, Geometry helps to compare persistence diagrams, *Journal of Experimental Algorithmics (JEA)*, 22 (2017), 1–20.

[21] H. W. Kuhn, The hungarian method for the assignment problem, *Naval Research Logistics Quarterly*, 2 (1955), 83–97.

[22] N. Lahn, D. Mulchandani, and S. Raghvendra, A graph theoretic additive approximation of optimal transport, *Advances in Neural Information Processing Systems 32*, 2019, pp. 13813–13823.

[23] N. Lahn and S. Raghvendra, A faster algorithm for minimum-cost bipartite matching in minor-free graphs, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2019, pp. 569–588.

[24] N. Lahn and S. Raghvendra, A weighted approach to the maximum cardinality bipartite matching problem with applications in geometric settings, *Journal of Computational Geometry*, 11 (2021). Special Issue of Selected Papers from SoCG 2019.

[25] F. T. Leighton and P. W. Shor, Tight bounds for minimax grid matching, with applications to the average case analysis of algorithms, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, ACM, 1986, pp. 91–103.

[26] T. Lin, N. Ho, and M. I. Jordan, On efficient optimal transport: An analysis of greedy and accelerated mirror descent algorithms, *arXiv:1901.06482 [cs.DS]*, (2019).

[27] A. Madry, Navigating central path with electrical flows: From flows to matchings, and back, *54th Annual IEEE Symposium on Foundations of Computer Science*, 2013, pp. 253–262.

[28] R. Motwani, Average-case analysis of algorithms for matchings and related problems, *J. ACM*, 41 (1994), 1329–1356.

[29] M. Mucha and P. Sankowski, Maximum matchings via gaussian elimination, *45th Annual IEEE Symposium on Foundations of Computer Science*, 2004, pp. 248–255.

[30] Y. V. Prokhorov, Convergence of random processes and limit theorems in probability theory, *Teor. Veroyatnost. i Primenen.*, 1 (1956), 177–238. [English translation: Theory Probab. Appl. 1 (1956), 157-214].

[31] K. Quanrud, Approximating optimal transport with linear programs, *Symposium on Simplicity of Algorithms*, Vol. 69, 2019, pp. 6:1–6:9.

[32] R. Sandler and M. Lindenbaum, Nonnegative matrix factorization with earth mover's distance metric for image analysis, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33 (2011), 1590–1602.

[33] P. W. Shor and J. E. Yukich, Minimax Grid Matching and Empirical Measures, *The Annals of Probability*, 19 (1991), 1338 – 1348.

[34] V. Strassen, The existence of probability measures with given marginals, *The Annals of Mathematical Statistics*, 36 (1965), 423–439.

[35] J. van den Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang, Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Any limitations of the work should be apparent from the descriptions of the relevant problem statements that have been included.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We are not aware of any potential negative societal impacts of this work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Our paper does not involve human subjects or sensitive data, and we are not aware of any relevant ethical concerns.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Some proofs are to be included in the appendix portion of the supplemental results.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Relevant code for our experiments will be included in the supplemental materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A] Our experiments are focused on analyzing running times of algorithms, and are not concerned with training on data sets.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] For applicable charts, we included error bars that represent a single standard deviation.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A] We do not use existing assets.
   (b) Did you mention the license of the assets? [N/A] We do not use existing assets.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include code in the supplemental materials.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We did not use existing data.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Our only input data consists of randomly generated geometric points.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]