Geometric Matching and Bottleneck Problems

Sergio Cabello
University of Ljubljana, Ljubljana, Slovenia
Institute of Mathematics, Physics and Mechanics, Ljubljana, Slovenia

Siu-Wing Cheng
HKUST, Hong Kong, China

Otfried Cheong
SCALGO, Aarhus, Denmark

Christian Knauer
University of Bayreuth, Germany

Abstract

Let P be a set of at most n points and let R be a set of at most n geometric ranges, such as disks and rectangles, where each $p \in P$ has an associated *supply* $s_p > 0$, and each $r \in R$ has an associated *demand* $d_r > 0$. A (many-to-many) *matching* is a set $\mathcal{A}$ of ordered triples $(p, r, a_{pr}) \in P \times R \times \mathbb{R}_{>0}$ such that $p \in r$ and the a_{pr}'s satisfy the constraints given by the supplies and demands. We show how to compute a maximum matching, that is, a matching maximizing $\sum_{(p,r,a_{pr})\in\mathcal{A}} a_{pr}$.

Using our techniques, we can also solve minimum bottleneck problems, such as computing a perfect matching between a set of n red points P and a set of n blue points Q that minimizes the length of the longest edge. For the L_∞-metric, we can do this in time $O(n^{1+\varepsilon})$ in any fixed dimension, for the L_2-metric in the plane in time $O(n^{4/3+\varepsilon})$, for any $\varepsilon > 0$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Many-to-many matching, bipartite, planar, geometric, approximation

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.31

Related Version *Full Version*: https://arxiv.org/abs/2310.02637

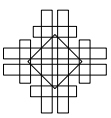
Funding This research was funded in part by the Slovenian Research and Innovation Agency (P1-0297, J1-2452, N1-0218, N1-0285), in part by the Research Grants Council, Hong Kong, China (project no. 16207419), and in part by the European Union (ERC, KARST, 101071836). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements The authors thank Sang Won Bae for helpful discussions.

1 Introduction

The classic *assignment* problem is to assign n items (such as jobs) to n other items (such as machines). It asks for a bijection between the two sets that maximizes some quality or minimizes some cost function. Computing a perfect matching in a bipartite graph is one special case of the assignment problem. The assignment problem stood at the cradle of the field of combinatorial optimization in the 1950's, and influenced the development of integer and linear programming and the theory of network flows. The book by Burkard et al. [5] presents the history of the problem and results for different cost functions.

A natural generalization is to relax the one-to-one requirement. Problems of this nature arise commonly in practice, for instance when assigning mobile clients to base stations, patients to hospitals, dinner guests to restaurants, etc. Such problems are commonly solved using maximum-flow, see for instance Exercises 7, 8, 9, 16, 19, 26 in Chapter 7 of Kleinberg and Tardos' book [23].



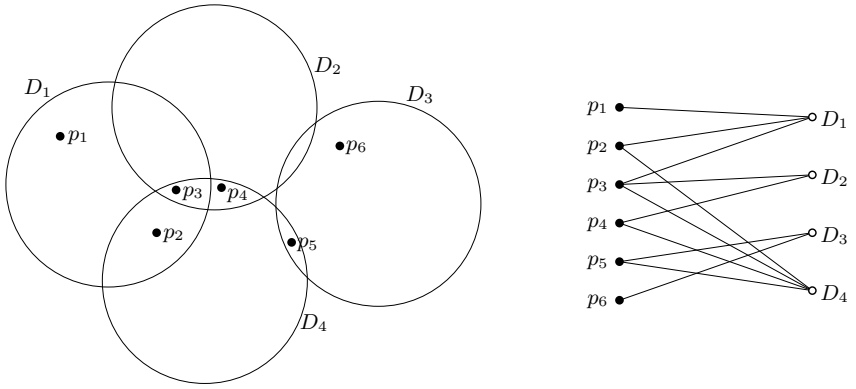


Figure 1 A set of six points and a set of four congruent disks in the plane (left) and their incidence graph (right).

Geometric matching problems. In several settings one encounters assignment or matching problems between geometric objects. Let us define these geometric matching problems more precisely. Let P be a set of points and let R be a set of geometric ranges, such as disks or rectangles. We use n to denote $|P| + |R|$. Their *incidence graph*, denoted by $I(P, R)$, is the bipartite graph (see Figure 1 for an example)

$$I(P, R) \;=\; \big(P \sqcup R, \{pr \mid p \in P, r \in R, p \in r\}\big).$$

Each point $p \in P$ has an associated *supply* $s_p > 0$ and each range $r \in R$ has an associated *demand* $d_r > 0$. We consider the elements of P as *providers* and the elements of R as *consumers* of a common commodity that is infinitely divisible. The commodity can be sent only along the edges of $I(P, R)$, and the goal is to maximize the amount of commodity that is sent while respecting the supplies and the demands. The *target value* of the instance is

$$\mu \;=\; \min\left\{\sum_{p\in P} s_p,\; \sum_{r\in R} d_r\right\},$$

which is an obvious upper bound on the amount of the commodity that can be sent.

Formally, a (many-to-many) *matching* is a set $\mathcal{A}$ of ordered triples $(p, r, a_{pr}) \in P \times R \times \mathbb{R}_{>0}$ such that

- for every $(p, r, a_{pr}) \in \mathcal{A}$, pr is an edge of $I(P, R)$;
- for every $(p, r) \in P \times R$, there is at most one triple $(p, r, \cdot)$ in $\mathcal{A}$;
- for every point $p \in P$, $\sum_{r\in R:\ (p,r,a_{pr})\in\mathcal{A}} a_{pr} \leqslant s_p$;
- for every range $r \in R$, $\sum_{p\in P:\ (p,r,a_{pr})\in\mathcal{A}} a_{pr} \leqslant d_r$.

The *value* of the matching $\mathcal{A}$ is $\eta = \eta(\mathcal{A}) = \sum_{(p,r,a_{pr})\in\mathcal{A}} a_{pr}$, and the objective is to maximize this value. A matching is *satisfying* when its value is the target value, that is, when $\eta = \mu$. The *size* of a matching is $|\mathcal{A}|$, that is, the number of edges of $I(P, R)$ that carry part of the commodity.

A geometric matching problem can be solved using a standard transformation to a maximum-flow instance built around $I(P, R)$ (with directed edges). However, any approach that explicitly constructs $I(P, R)$ is going to spend time at least proportional to its size, which may be $\Theta(n^2)$. Thus, even when using the recent breakthrough near-linear time algorithm of Chen et al. [9] for maximum-flow, we will spend quadratic time.

A key question is whether one can use the geometry of the input to solve the matching problem without explicitly constructing $I(P, R)$. Our contribution is to note that a compact representation of the incidences of geometric objects can indeed be used to solve the geometric

matching problem without constructing $I(P, R)$. More precisely, we use a union of complete bipartite graphs as a compact representation of $I(P, R)$. We denote by σ the size of this compact representation, which will be defined in Section 2. We provide two algorithms for the maximum matching problem depending on what we want to assume about the supplies and the demands.

Integral supplies and demands. Our first algorithm uses the recent algorithm of Chen et al. [9] to solve maximum-flow instances in near-linear time and assumes that the supplies and the demands are integers. Recall that $\mu = \min\{\sum_{p\in P} s_p, \sum_{r\in R} d_r\}$. The running time is $O(\sigma^{1+\varepsilon} \log \mu)$ with high probability (more specifically, the probability is at least $1 - \sigma^{-O(1)}$), for any constant $\varepsilon > 0$. Retrospectively, this result is a simple combination of the recent breakthrough in computing maximum flow and compact representations of incidences in geometric settings, a common tool in computational geometry.

This first algorithm has interesting consequences for several fundamental problems concerning one-to-one matchings in geometric settings. However, not every solution to the matching problem with unit supplies and demands is a one-to-one matching, because the solution may use fractional flow values on some edges. For example, on a regular bipartite graph solving the matching problem with unit supplies and demands is trivial using fractional flows, but it is still a challenge to determine an integral maximum flow. We solve this problem using the approach in [15, 26].

Here is a list of consequences of our new algorithm; in all cases, n is the number of points and ranges, the results hold with high probability, and ε is an arbitrary positive constant that affects the constants hidden in the O-notation:

- When R is a set of axis parallel rectangles in the plane, we can compute a maximum one-to-one matching in $I(P, R)$ in $O(n^{1+\varepsilon})$ time.
- When R is a set of congruent disks in the plane, we can compute a maximum one-to-one matching in $I(P, R)$ in $O(n^{4/3+\varepsilon})$ time.
- For two given point sets P and Q in the plane, and a real value $\lambda \geqslant 0$, we can compute a maximum one-to-one matching in the bipartite graph on $P \sqcup Q$ that connects points at distance at most λ. In the L_∞-metric, the running time is $O(n^{1+\varepsilon})$, while in the L_2-metric the running time is $O(n^{4/3+\varepsilon})$.
- We can decide in $O(n^{1+\varepsilon})$ time if the bottleneck distance of two persistence diagrams with n points outside the diagonal is smaller or equal to a given value $\lambda \geqslant 0$.

For all problems, the previous best algorithm had a running time of roughly $O(n^{3/2})$ and was due to Efrat, Itai and Katz [14]. That result was an efficient version of the maximum-matching algorithm by Hopcroft and Karp [17], where one makes $O(n^{1/2})$ rounds, and in each round one finds several augmenting paths of the same length. Using range searching data structures, each round is implemented in near-linear time.

If one is willing to use as a parameter the so-called density ρ of the geometric objects, then Bonnet, Cabello, and Mulzer [4] provided an algorithm to compute a maximum (one-to-one) matching of $I(P, R)$ in $O(\rho^{3\omega/2} n^{\omega/2})$ time with high probability, where $\omega > 2$ is a constant such that any two $n \times n$ matrices can be multiplied in time $O(n^\omega)$. With the current bounds on ω, the running time is $O(\rho^{3.56} n^{1.19})$. Note that in the worst case $\rho = \Theta(n)$ and then the algorithm takes $\Omega(n^4)$ time, even for ω arbitrarily close to 2. For congruent disks and the current value of ω, our new bound is better when $\rho = \Omega(n^{0.041})$; if ω can be made arbitrarily close to 2, our new bound for congruent disks is still better when $\rho = \Omega(n^{1/9+\varepsilon})$ for some $\varepsilon > 0$. For axis-parallel rectangles and the current value of ω, our new bound is

better for any density ρ; if ω can be made arbitrarily close to 2, our new bound for rectangles is better as soon as $\rho = \Omega(n^{\varepsilon})$ for some $\varepsilon > 0$. They give other results that are independent of the density ρ, but those do not apply to the bipartite setting that we consider here.

Har-Peled and Yang [16] considered obtaining $(1+\varepsilon)$-approximations to the maximum matching in near-linear time.

Arbitrary supplies and demands. In our second algorithm we do not make any assumption about the supplies and the demands. In computational geometry it is common to allow arbitrary real numbers in the input, because the input may be coming from a previous computation and may be, say, a Euclidean distance or the area of a disk.

Our algorithm is a variant of Dinitz' algorithm for maximum-flow as implemented by Sleator and Tarjan [32]. We prune the current matching after each phase of Dinitz' algorithm, ensuring that the current flow is non-zero on a linear number of edges only. We also store the edges from providers to consumers in each residual graph implicitly using a clique cover of the graph. Assuming a compact representation of the incidence graph $I(P,R)$ of size σ we compute a maximum matching in $O(n\sigma \log \sigma)$ time.

Our time bound is better than using the usual transformation into a maximum flow and then using the fastest known combinatorial algorithms for maximum flow that work for arbitrary edge capacities: For maximum-flow instances on graphs with $|V|$ vertices and $|E|$ edges, the best known running time is $O(|V||E|)$ [22, 29, 30]. In our setting, this leads to a worst-case running time of $O(n^3)$ (for the decision problem), because $I(P,R)$ could be dense.

Our time bound is also better than using the construction used for the integral case; see Figure 3. In this construction, we have a graph with $O(\sigma)$ edges and vertices, which means that a maximum flow can be computed in $O(\sigma^2)$ time. This time bound is strictly better than $O(n\sigma \log \sigma)$ only when $n = \omega(\sigma/\log \sigma)$, but this never happens for any of the geometric scenarios we consider. Even for points and intervals on the line we have $\sigma = \Theta(n \log n)$.

Our algorithm has the following consequences for the geometric matching problem with arbitrary supplies and demands:

- When R is a set of axis parallel rectangles in the plane, we can compute a maximum matching in $I(P,R)$ in $O(n^2 \log^3 n)$ time.
- When R is a set of congruent disks in the plane, we can compute a maximum matching in $I(P,R)$ in $O(n^{7/3} \log^2 n)$ time.
- For two given point sets P and Q in the plane, and a real value $\lambda \geqslant 0$, we can compute a maximum many-to-many matching in the bipartite graph on $P \sqcup Q$ that connects points at distance at most λ. In the L_∞-metric, the running time is $O(n^2 \log^3 n)$, while in the L_2-metric the running time is $O(n^{7/3} \log^2 n)$.

Bottleneck instances. Consider the scenario where we have two sets of n points each in the plane and we want to compute a perfect matching between them that minimizes the length of the longest edge. That is called a *bottleneck matching*, and its computation has been considered before by Efrat, Itai and Katz [14]. A precise definition is given in Section 3.2.

A similar concept of using a perfect matching to define a bottleneck distance has been introduced in the context of persistence diagrams; see [10, 12, 13, 21] or the definition in Section 3.3. This concept is closely related to the L_∞-metric in the plane.

Similarly, for non-unit demands and supplies we can consider a matching problem between two sets of points where we want to compute a satisfying matching that minimizes the length of the longest edge being used. This is the bottleneck value for the geometric matching problem.

Long and Wong [24] introduced this *spatial matching problem*, which they denoted SPM-MM. Closely related problems have been considered in the field of spatial databases, for example when assigning mobile clients to base stations with limited capacity [25, 33]. In this context, one uses the L_2-metric.

In all these problems, we can solve the *decision problem* using the algorithms described above: For a given λ, is there a perfect matching or a satisfying matching where all edges have length at most λ? We can then perform a binary search in a set of (so-called *critical*) values that is known to contain the optimal value, without substantially increasing the running time. We obtain the following results, where, as before, the running times hold with high probability and ε is an arbitrarily small positive constant:

- The bottleneck matching for two sets of n points in the plane under the L_2-metric can be computed in in $O(n^{4/3+\varepsilon})$ time. The previous time bound was $O(n^{3/2}\log n)$ [14].
- The bottleneck matching for two sets of n points in the plane under the L_1- or L_∞-metric can be computed in $O(n^{1+\varepsilon})$ time. The previous bound was $O(n^{3/2}\log n)$ [14].
- The bottleneck distance of two persistence diagrams with n points outside the diagonal can be computed in $O(n^{1+\varepsilon})$ time. The previous bound was $O(n^{3/2}\log n)$.
- The bottleneck value for a geometric matching problem can be computed in $O(n^{4/3+\varepsilon})$ time, if the supplies and the demands are small integers, and in $O(n^{7/3}\log^3 n)$ time for arbitrary supplies and demands.

Note that the recent result of Katz and Sharir [20] is for the bottleneck matching problem in the *non-bipartite* case; the techniques do not apply to the bipartite case we consider here.

It is perhaps surprising that a minimum bottleneck matching is not necessarily planar. Finding a minimum *planar* bottleneck matching is hard [7, 18].

Due to space constraints, missing proofs and details can be found in the full version [6].

2 Compact representation of incidences

For any two disjoint sets U and V, let $K(U,V)$ denote the complete bipartite graph with vertex set $U \sqcup V$ and edge set $\{uv \mid u \in U, v \in V\}$.

Let $G = (U, V, E)$ be a bipartite graph with vertex set $U \sqcup V$. Let I be some index set. A *compact representation* of G is a collection $\{(U_i, V_i) \mid i \in I\}$ such that

- for each $i \in I$, $U_i \subseteq U$ and $V_i \subseteq V$, and
- G is the union of the complete bipartite graphs $K(U_i, V_i)$ over $i \in I$.

The compact representation is edge-disjoint if the graphs $K(U_i, V_i)$, $i \in I$, are pairwise edge-disjoint. The *size* of the compact representation is defined as $\sum_{i\in I}(|U_i| + |V_i|)$. The size represents the length of the description of the representation, assuming that vertices can be represented in constant space (the bit complexity would be slightly larger). See Figure 2 for an example.

Each bipartite graph $G = (U, V, E)$ has a trivial compact representation of size $\Theta(|U| + |V| + |E|)$, namely $\{(\{u\}, \{v\}) \mid uv \in E\}$. However, some bipartite graphs have compact representations of size $o(|E|)$. We next describe a few such cases arising in geometry.

▶ **Theorem 1** (Implicit in [2, Section 3] or [11, Chapter 5]). *Let P be a set of at most n points and let R be a set of at most n axis-parallel boxes in $\mathbb{R}^d$, for constant d. We can compute an edge-disjoint compact representation of the incidence graph $I(P, R)$ of size $O(n\log^d n)$ in time $O(n\log^d n)$.*

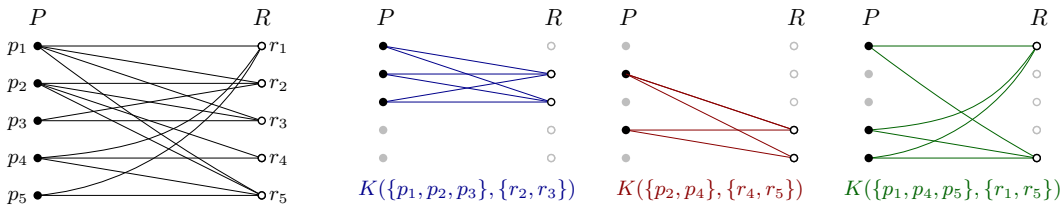


Figure 2 An example of a bipartite graph (left) represented as the union of three complete bipartite graphs (right). In this example, the compact representation has size $(3+2)+(2+2)+(3+2)=14$. Note that this compact representation is not edge-disjoint because of the edge p_4r_5.

▶ **Theorem 2** ([19]). *Let P be a set of at most n points and let D be a set of at most n congruent disks in the plane. We can compute an edge-disjoint compact representation of the incidence graph $I(P,D)$ of size $O(n^{4/3}\log n)$ in time $O(n^{4/3}\log n)$.*

▶ **Theorem 3** (Implicit in [27]; see also [2, Section 4.3]). *Let P be a set of at most n points and let H be a set of at most n halfspaces in $\mathbb{R}^d$, for constant d. We can compute an edge-disjoint compact representation of the incidence graph $I(P,H)$ of size $O(n^{\frac{2d}{d+1}}\operatorname{polylog} n)$ in time $O(n^{\frac{2d}{d+1}}\operatorname{polylog} n)$.*

▶ Remark. Points and balls of different sizes in $\mathbb{R}^d$ can be treated as points and halfspaces in $\mathbb{R}^{d+1}$ using standard techniques. However, for $d=2$ one can get stronger results for any semi-algebraic regions, and in particular for disks; see [3] for a recent development.

3 Geometric matching problems with integral supplies and demands

Consider a geometric matching problem for a set P of points and a set R of geometric ranges, with n objects in total. Each point $p \in P$ has a supply $s_p > 0$ and each range $r \in R$ has a demand $d_r > 0$.

Assume that we have a compact representation $\{(P_i, R_i) \mid i \in I\}$ of size σ for the incidence graph $I(P,R)$. We are going to reduce the problem of deciding whether there is a satisfying matching to a max-flow problem in a graph whose size is roughly the size of the compact representation.

The max-flow instance is constructed as follows (see Figure 3 for an example): Initially we have the vertices $P \sqcup R$ and no edges. We then add the following:

- We add a new vertex s and, for each $p \in P$, we add the directed edge (s,p) with capacity s_p.
- We add a new vertex t and, for each range $r \in R$, we add the directed edge (r,t) with capacity d_r.
- For each $i \in I$, we add a new vertex v_i, directed edges (p,v_i) for all $p \in P_i$, and directed edges (v_i,r) for all $r \in R_i$. These edges have capacity $\mu = \min\{\sum_{p\in P} s_p, \sum_{r\in R} d_r\}$. We refer to the set of edges added in this step as E_i.

This finishes the description of the s-t max-flow instance, which we denote by G. This instance depends on P, R, the demands, the supplies, and the compact representation of $I(P,R)$ being used.

The instance we constructed has $2+|P|+|R|+|I| = 2+n+|I| = O(\sigma)$ vertices and $|P|+|R|+\sum_{i\in I}(|P_i|+|R_i|) = O(\sigma)$ edges. The time to construct the instance is also $O(\sigma)$, assuming that the compact representation is already available. The maximum capacity in the instance is μ.

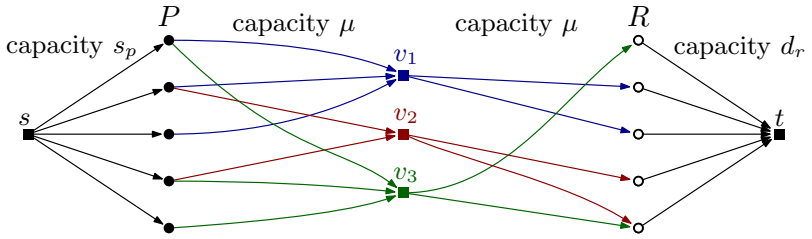


Figure 3 The max-flow instance G for the compact representation of Figure 2.

▶ **Lemma 4.** *There is an s-t flow of value η in G if and only if there exists a matching of value η. Moreover, from a flow of value η we can recover a matching of value η and size $O(\sigma)$ in $O(\sigma)$ time.*

▶ **Theorem 5.** *Consider a geometric matching problem for a set P of at most n points and a set R of at most n ranges such that the demands and the supplies are integral. Let μ be the target value of the instance. Assume that we have a compact representation of the incidence graph $I(P, R)$ of size σ. Then we can compute a maximum (many-to-many) matching of size $O(\sigma)$ in $O(\sigma^{1+\varepsilon} \log \mu)$ time with high probability, for any constant $\varepsilon > 0$.*

Proof. We build the max-flow instance associated with the compact representation in $O(\sigma)$ time. Note that the resulting instance has $O(\sigma)$ vertices, $O(\sigma)$ edges, and maximum capacity μ. We solve the s-t max-flow problem in $O(\sigma^{1+o(1)} \log \mu)$ time with high probability using the algorithm of Chen et al. [9, Corollary 1.3]. This running time can be upper bounded by $O(\sigma^{1+\varepsilon} \log \mu)$ for any constant $\varepsilon > 0$. ◀

3.1 Maximum and perfect matchings in intersection graphs

We next turn our attention to the setting where we want to compute a maximum one-to-one matching in a bipartite graph that has a compact representation. A particular case of this is deciding whether the graph has a perfect matching (and constructing one).

This is essentially a matching problem with unit supplies and demands. Standard integrality results imply that the value of the matching is the size of the maximum one-to-one matching. However, we have to be careful because the algorithm in Theorem 5 could return a *non-integral* matching. A maximum matching can be converted into a maximum one-to-one matching using an adaptation of the method by Mądry [26, Section 8 of the full version]. We only state the main result here.

▶ **Theorem 6.** *Let P be a set of at most n points and let R be a set of at most n ranges. Assume that we have a compact representation of the incidence graph $I(P, R)$ of size σ. We can compute a maximum one-to-one matching in $I(P, R)$ in $O(\sigma^{1+\varepsilon})$ time with high probability, for any constant $\varepsilon > 0$.*

3.2 Bottleneck distances

Let P and Q be two sets of n points in the plane and consider the bipartite graph obtained by connecting points whose L_∞-distance is at most λ:

$$G_\infty(P, Q, \lambda) \;=\; \big(P \sqcup Q, \{pq \mid \max\{|x(p) - x(q)|, |y(p) - y(q)|\} \leqslant \lambda\}\big).$$

We are interested in computing the L_∞-*bottleneck distance* between P and Q, defined as

$$\lambda^*_\infty(P,Q) \;=\; \min\{\lambda \mid G_\infty(P,Q,\lambda) \text{ has a perfect matching}\}.$$

The same concepts can be defined for the L_1- and L_2-distances

$$\begin{aligned} G_1(P,Q,\lambda) &= \big(P \sqcup Q, \{pq \mid |x(p)-x(q)| + |y(p)-y(q)| \leqslant \lambda\}\big),\\ \lambda^*_1(P,Q) &= \min\{\lambda \mid G_1(P,Q,\lambda) \text{ has a perfect matching}\},\\ G_2(P,Q,\lambda) &= \big(P \sqcup Q, \{pq \mid (x(p)-x(q))^2 + (y(p)-y(q))^2 \leqslant \lambda^2\}\big),\\ \lambda^*_2(P,Q) &= \min\{\lambda \mid G_2(P,Q,\lambda) \text{ has a perfect matching}\}. \end{aligned}$$

As noted by Efrat, Itai and Katz [14, Section 6.2.2] we can use an algorithm for the decision problem (where λ is given) to perform a binary search to find $\lambda^*(P,Q)$.

3.3 Bottleneck distance of persistence diagrams

Our techniques can be applied to compute the bottleneck distance of *persistence diagrams*, one of the most important concepts in computational topology; see for example the book by Edelsbrunner and Harer [13, Part C] or the book by Dey and Wang [12, Chapter 3].

▶ **Corollary 7.** *For persistence diagrams X and Y, each with at most n points outside the diagonal, we can compute their bottleneck distance $W_\infty(X,Y)$ in $O(n^{1+\varepsilon})$ time with high probability, for any $\varepsilon > 0$*

4 Geometric matching problems with arbitrary supplies and demands

In the previous section we made use of a fast algorithm for max-flow that works for integer capacities. In this section we consider the scenario where supplies and demands are arbitrary real numbers, and we insist on an algorithm that solves the problem exactly on the real RAM, as is customary in computational geometry.

Clearly, we can restate Theorem 5 by making use of a combinatorial max-flow algorithm. The fastest combinatorial algorithm that works for arbitrary capacities runs in time $O(nm)$, where n is the number of vertices, m the number of edges of the network. Applying this to the graph of Figure 3 results in a running time of $O(\sigma^2)$, where σ is the size of the compact representation of the incidence graph $I(P,R)$.

We now show how this can be improved when $\sigma > n$. Instead of explicitly computing the five-layer graph of Figure 3, we run Dinitz' algorithm on the classic four-layer graph based on $I(P,R)$, but we represent that graph and all intermediate graphs implicitly, making use of compact representations.

4.1 Dinitz' algorithm with pruning

Given a set of at most n points P with supplies s_p, a set of at most n ranges R with demands d_r, and an incidence graph $I(P,R)$ given through a compact representation of size σ, we consider the flow network consisting of a source s, the vertices P, the vertices R, and a sink t: For each $p \in P$, we have the edge (s,p) with capacity s_p, for each $r \in R$, we have the edge (r,t) with capacity d_r, and for each $(p,r) \in I(P,R)$ we have the edge (p,r) with infinite capacity. Let G denote this flow network. Since G can have quadratic complexity, we will not actually construct it explicitly.

Let f be a flow in G. Naturally, f induces a matching, so we call f a flow or matching interchangeably. Let $H(f)$ be the *undirected* graph obtained from G by removing s and t, and keeping the undirected edges $\{p, r\}$ if and only if $f(p, r) > 0$.

▶ **Lemma 8.** *For every flow f in G, there exists a flow f_0 in G with the same value such that $H(f_0)$ is a forest and a subgraph of $H(f)$.*

Proof. Suppose that $H(f)$ contains a cycle. This cycle zig-zags back and forth between P and R. We color the edges of the cycle alternatingly red and blue. Both red and blue edges carry positive flows out of their endpoints in P. We can add a value Δ to the flow on all red edges and subtract Δ from the flow on all blue edges without changing the outgoing and incoming flow at any node. Let e be the blue edge that has the minimum flow value among the blue edges. By choosing Δ to be the flow value on e, we can reduce the flow value on e to zero. This gives a new flow f' in G such that $H(f')$ is the subgraph of $H(f)$ with e (and possibly more edges) deleted. We repeat this procedure until there is no cycle left in $H(f_0)$. ◀

We will use Dinitz's algorithm to compute a maximum flow in G. Dinitz' algorithm starts with the zero flow, and augments the flow in phases. In each phase, it computes the *level graph* $L(f)$ of the residual graph $G(f)$ of G with respect to the current flow f. The level graph $L(f)$ is defined as follows: For each node v of $G(f)$, we define the *level* $\ell(v)$ of v as the length of the shortest path (in terms of the number of edges) from s to v in $G(f)$. The level graph contains exactly those edges (u, v) of $G(f)$ where $\ell(v) = \ell(u) + 1$. In other words, all edges of the level graph are on a shortest path starting in s.

Clearly, s is the only node at level zero in $L(f)$. In our application, every odd level consists of elements of P, every even level other than zero consists of elements of R. The sink t resides in an odd level. We ignore all other nodes on level $\ell(t)$ and above.

Next, Dinitz' algorithm computes a *blocking flow* g in $L(f)$, and augments f to $f + g$ before proceeding to the next phase. A flow g is blocking iff for every path γ from s to t in $L(f)$, there is an edge e of γ that is saturated by g, that is, the flow sent by g on e is equal to the capacity on e. Dinitz' algorithm terminates when there is no longer a path from s to t in the level graph.

Our algorithm differs from Dinitz' algorithm in only one detail: After augmenting the current flow, we apply Lemma 8 to prune the current matching. It follows that at the end of each phase, the graph $H(f)$ has at most $2n - 1$ edges (since it is a forest).

The following lemma shows that at most n phases of this algorithm suffice to obtain a maximum flow in G. This is essentially Theorem 8.4 in [32], but some extra effort is needed due to the pruning step (by Lemma 8) at the end of each phase.

We first establish some notation. In the residual graph $G(f)$ and the level graph $L(f)$, we call the edges from s to P the *feeder edges*, the edges from elements of P to elements of R the *forward edges*, the edges from R to t the *draining edges*, and all other edges *backward edges*. Note that backward edges do not exist in G, rather they are the reverse of an edge with positive flow in G. In the residual graph, backward edges can be directed from R to P, from P to s, or from t to R. Only backward edges from R to P will appear in the level graph, as $\ell(s) = 0$ and since we ignore levels after $\ell(t)$.

By the definition of the residual graph, the capacity of every feeder edge (s, p) is the unused supply left at p, that is $s_p - \sum_{r \in R} f(p, r)$. The capacities of forward edges are ∞. The capacity of every drain edge (r, t) is equal to the demand at r that has not been satisfied yet, that is $d_r - \sum_{p \in P} f(p, r)$. Finally, the capacity of a backward edge (u, v) is equal to $f(v, u)$.

▶ **Lemma 9.** *Starting with the zero flow, at most n successive augmentations by blocking flows in the corresponding level graphs will yield a maximum flow in G.*

Proof. We will show that the level of t in the level graph increases after each augmentation. Since the level of t is always odd, starts with 3, and can be at most $2n+1$, there can be at most n augmentations.

Let f be the current flow in G. Let $Z = G(f)$ denote the residual graph of G with respect to f. Let g be a blocking flow in $L(f)$. We use $\ell(v)$ to denote the level of the node v in $L(f)$, which is also the distance of v from s in Z. Define $h = f + g$. Let f' be obtained from h by Lemma 8. Let $Z' = G(f')$ be the residual graph of G with respect to f'.

▷ Claim 10. If (v, w) is a feeder, forward, or draining edge of Z', then (v, w) is an edge of Z.

Proof of Claim 10. If (v, w) is a forward edge in Z', then (v, w) is an edge of G. Since the capacity of (v, w) is infinite, it is also an edge of Z.

The feeder and draining edges are never involved in a cycle in the proof of Lemma 8. So the flows on them are the same with respect to f and f', which means that their residual capacities in Z and Z' are the same. Therefore, if (v, w) is in Z', it is also in Z. ◁

▷ Claim 11. For every edge (v, w) in Z', we have $\ell(w) \leqslant \ell(v) + 1$.

Proof of Claim 11. We first observe that if (v, w) is an edge of Z, then $\ell(w) \leqslant \ell(v)+1$ follows from the definition of $\ell(w)$. For feeder, forward, and draining edges the claim therefore follows immediately from Claim 10.

If (v, w) is a backward edge, then $f'(w, v) > 0$. By Lemma 8, $H(f')$ is a subgraph of $H(h)$ and so $h(w, v) > 0$. If $f(w, v) > 0$ as well, then the edge (v, w) appeared in Z, so $\ell(w) \leqslant \ell(v) + 1$.

If $f(w, v) = 0$, then $g(w, v) = h(w, v) > 0$. The positive value of the blocking flow $g(w, v)$ means that (w, v) was an edge in $L(f)$, which by definition of $L(f)$ means that $\ell(v) = \ell(w)+1$ and therefore $\ell(w) = \ell(v) - 1 < \ell(v) + 1$. ◁

Let now $\ell'(v)$ be the level of a node v in Z', that is, the level of v in $L(f')$. We claim that $\ell'(v) \geqslant \ell(v)$. Indeed, consider any path from s to v in Z'. By Claim 11, $\ell(\cdot)$ increases by at most one along each edge of this path, so $\ell(v)$ is at most the length of this path.

In particular, we have $\ell'(t) \geqslant \ell(t)$. We show that in fact $\ell'(t) > \ell(t)$. For the sake of contradiction, assume that $\ell'(t) = \ell(t)$. This means that for any shortest path from s to t in Z', $\ell(\cdot)$ increases by exactly one on each edge of this path. This means that for each edge (v, w) on this path, v and w are on consecutive levels of $L(f)$. We claim that any shortest path of length $\ell(t)$ from s to t in $L(f')$ is also a path in $L(f)$. The reason is as follows. By Claim 10, if (v, w) is a feeder, forward, or draining edge, then (v, w) is an edge of Z and therefore also an edge of $L(f)$. In the remaining case, $v \in R$, $w \in P$, and $\ell(w) = \ell(v) + 1$. If (v, w) is an edge in $L(f)$, we are done. If (v, w) is not an edge of $L(f)$, then $f(w, v) = 0$. Since (v, w) is an edge of Z', we have $f'(w, v) > 0$, and by Lemma 8 $h(w, v) = g(w, v) > 0$. This implies again that (w, v) must have been a forward edge in $L(f)$, and so $\ell(v) = \ell(w)+1$, a contradiction. This establishes our claim.

It follows that the entire shortest path of length $\ell(t)$ from s to t was already a path in $L(f)$. But by definition of a blocking flow, that means that g must have saturated some edge along this path. If this was a feeder or draining edge, those edges are no longer in Z', a contradiction. A forward edge cannot be saturated (it has infinite capacity), so it must be a backward edge, say (r, p). Saturating a backward edge (r, p) means that $g(r, p) = f(p, r)$. It follows that $h(p, r) = 0$, which implies $f'(p, r) = 0$ by Lemma 8. Therefore, (r, p) is not an edge of Z', again a contradiction. ◀

We cannot afford to store the residual graph $G(f)$ or the level graph $L(f)$ explicitly, as they may have a quadratic number of edges. We will therefore store these graphs implicitly, using the given compact representation for $I(P, R)$. Inbetween phases of the algorithm, we only need to store the current flow f. Since it is non-zero on $O(n)$ edges only, we store this flow explicitly. In the following sections we describe how to implement a phase of Dinitz' algorithm efficiently.

4.2 Constructing the level graph

Given the current flow f, we will represent the level graph $L(f)$ by explicitly constructing the elements of each level of $L(f)$, all feeder edges, all draining edges, and all backward edges. There are $O(n)$ such edges, so we can indeed afford to store them explicitly. The forward edges are represented using compact representations.

Recall that we are given a compact representation $\{(U_i, V_i) \mid i \in I\}$ of $I(P, R)$ for some index set I. We start by building an index for this representation, which, for every $p \in P$, gives us a list of all $i \in I$ where $p \in U_i$. We also set $\tilde{P} := P$, $\tilde{R} := R$, and $\tilde{I} := I$.

There are at most $|P| \leqslant n$ feeder edges from s to P in $L(f)$. We construct these edges and their capacities by brute-force in $O(n)$ time. This also yields the set $P_1 \subseteq P$ of elements of level 1 of $L(f)$.

We now query our index to retrieve the set of indices $I_1 \subseteq \tilde{I}$ such that $U_i \cap P_1 \neq \emptyset$ for $i \in I_1$. We set $R_2 = \bigcup_{i \in I_1} V_i$ to be the set of vertices on level 2, and build the following compact representation for the forward edges from level 1 to level 2:

$$\{(U_i \cap P_1, V_i) \mid i \in I_1\},$$

We update $\tilde{P} := \tilde{P} \setminus P_1$, $\tilde{R} := \tilde{R} \setminus R_2$, and $\tilde{I} := I \setminus I_1$.

If there is any $r \in R_2$ that has an unmet demand, then we place t in level 3 and produce the draining edges (r, t) with capacity equal to the unmet demand of each $r \in R_2$. The level graph is then complete.

If there is no $r \in R_2$ with unmet demand, we proceed to construct level 3. For each $r \in R_2$, we determine all $p \in \tilde{P}$ with flow into r, that is, with $f(p, r) > 0$. The set P_3 of all these vertices p constitutes level 3. We record the edges (r, p) as backward edges from level 2 to level 3.

We now again use our index to find the set of indices $I_3 \subseteq \tilde{I}$ such that $U_i \cap P_3 \neq \emptyset$ for $i \in I_3$. We set $R_4 = \bigcup_{i \in I_3} V_i \cap \tilde{R}$ to be the set of vertices on level 4, and build the following compact representation of the forward edges from level 3 to level 4:

$$\{(U_i \cap P_3, V_i \cap \tilde{R}) \mid i \in I_3\},$$

Again we update $\tilde{P} := \tilde{P} \setminus P_3$, $\tilde{R} := \tilde{R} \setminus R_4$, and $\tilde{I} := \tilde{I} \setminus I_3$, and repeat this process until either we place the sink t or an even level becomes empty. In the latter case there is no path from s to t in the level graph, and the current flow is a maximum flow.

▶ **Lemma 12.** *The elements of each level of $L(f)$, the feeder edges, the backward edges, the draining edges, and compact representations of all forward edges of $L(f)$ can be constructed in $O(n + \sigma)$ time.*

Proof. Constructing the feeder and draining edges explicitly takes $O(n)$ time. Finding the backward edges takes $O(n)$ total time as $H(f)$ has $O(n)$ edges. Each pair (U_i, V_i) of the compact representation of $I(P, R)$, is handled only once, and this takes time $O(|U_i| + |V_i|)$ using our index and suitable datastructures for $\tilde{P}$, $\tilde{R}$, and $\tilde{I}$. ◀

4.3 Constructing a blocking flow

We use Sleator and Tarjan's algorithm [31] for finding a blocking flow in the level graph, see for instance the presentation in Section 8.2 of Tarjan's book [32]. The algorithm computes a blocking flow on a level graph with n vertices and m edges in time $O((n+m)\log(n+m))$. To run the algorithm, we construct a concrete level graph L' from our implicitly represented level graph $L(f)$. We start by setting L' to contain all the vertices, feeder, backwards, and draining edges of $L(f)$. For each pair of layers P_j and R_{j+1}, for $j = 1, 3, 5, \ldots$, we then proceed as in Section 3. Namely, we insert an intermediate layer C_j as follows: Recall that we have a compact representation $\{(U_i, V_i) \mid i \in I_j\}$ of the forward edges from P_j to R_{j+1}. For each $i \in I_j$, we add a new vertex v_i^j, directed edges (p, v_i^j) for all $p \in U_i$, and directed edges (v_i^j, r) for all $r \in V_i$. These edges have infinite capacity.

The graph L' has $O(\sigma)$ vertices and edges. It is easy to see that L' is the level graph of a flow network (namely the flow network L' itself), so Sleator and Tarjan's algorithm computes a blocking flow g' for L' in time $O(\sigma \log \sigma)$.

We convert the flow g' on L' into a flow g on the graph $L(f)$ as we have done in Lemma 4. It remains to argue that g is indeed a blocking flow on $L(f)$. To this end, we observe that the flows g and g' are identical on all feeder, backwards, and draining edges. The forward edges, on the other hand, have infinite capacity, so they cannot be saturated by any flow. For any node $v \in L(f)$ and any path from s to v in $L(f)$, there is also a path from s to v in L' that uses the same finite-capacity edges. Since g' saturates an edge on this path, g also saturates an edge on the path from s to v in $L(f)$.

4.4 Pruning the matching

After computing a blocking flow, we augment the previous flow. Since the previous flow is non-zero on $O(n)$ edges and the blocking flow is non-zero on $O(\sigma)$ edges, this can be done in time $O(n+\sigma) = O(\sigma)$. It remains to describe how to prune this new flow f efficiently, that is, how to compute the flow f_0 guaranteed by Lemma 8. It is important to start each iteration with a flow that uses $O(n)$ edges. This turns out to be surprisingly hard. We show in the full version [6] that the flow f can be pruned in $O(\sigma \log n)$ time using a red-blue link-cut tree, which is a variant of link-cut trees.

4.5 Putting everything together

▶ **Theorem 13.** *Consider a geometric matching problem for a set P of at most n points and a set R of at most n ranges. Assume that we have a compact representation of the incidence graph $I(P, R)$ of size σ. Then we can compute a maximum matching in $O(n\sigma \log \sigma)$ time.*

We can combine this with the compact representations presented in Theorems 1, 2 and 3 to obtain the following:

- The geometric matching problem for a set P of at most n points and a set R of at most n axis-parallel boxes in $\mathbb{R}^d$, for constant d, can be solved in $O(n^2 \log^{d+1} n)$ time.
- The geometric matching problem for a set P of at most n points and a set D of at most n congruent disks in the plane can be solved in $O(n^{7/3} \log^2 n)$ time.
- The geometric matching problem for a set P of at most n points and a set H of at most n halfspaces in $\mathbb{R}^d$, for constant d, can be solved in $O(n^{\frac{3d+1}{d+1}} \operatorname{polylog} n)$ time.

For the bottleneck versions in the plane, we can solve the decision problem for the L_∞-metric using the first item, while the second item can be used for the L_2-metric. One can then perform a binary search using the techniques described in Section 3.2, which only add an extra logarithmic factor to the running time needed to solve the decision problem.

5 Conclusions

Using Theorems 1 and 6 and a binary search scheme noted by Efrat, Itai and Katz [14], one can compute the L_∞- and L_1-bottleneck distance in $O(n^{1+\varepsilon})$ time with high probability for any $\varepsilon > 0$. Using Theorem 2 and 6 and distance selection algorithms [19, 34], we can compute the L_2-bottleneck distance in $O(n^{4/3+\varepsilon})$ time with high probability for any $\varepsilon > 0$. We conjecture that computing the L_2-bottleneck distance has a lower bound of $\Omega(n^{4/3-\varepsilon})$ for any $\varepsilon > 0$.

Our discussion has centered on incidence graphs of points and ranges. However, the same approach can be exploited for *any* bipartite graph that has a compact representation. For example, let B be a set of *blue* pairwise disjoint segments in the plane and let R be a set of *red* pairwise disjoint segments in the plane. Consider the red-blue intersection graph

$$X(R, B) = (R \sqcup B,\ \{rb \mid r \in R,\ b \in B,\ \text{segments } r \text{ and } b \text{ intersect}\}).$$

It follows from the results of Chazelle et al. [8] that $X(R, B)$ has a compact representation of size $O(n \log^2 n)$ that can be obtained in $O(n \log^2 n)$ time; see for example [28, Section 4] for an explicit formulation. It follows that we can solve a maximum matching in $X(R, B)$ in $O(n^{1+\varepsilon})$ time with high probability, for any $\varepsilon > 0$.

As another example, consider a simple polygon, and split it with a diagonal d. Let P be the set of vertices on one side of d, Q the set of vertices on the other side of Q. A maximum matching between P and Q such that $p \in P$ and $q \in Q$ can be matched only if they are mutually visible inside the polygon can be computed in $O(n^{1+\varepsilon})$ time, since the visibility edges admit a near-linear-size compact representation [1].

References

1 Pankaj K. Agarwal, Noga Alon, Boris Aronov, and Subhash Suri. Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12:347–365, 1994. doi:10.1007/BF02574385.

2 Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In B. Chazelle, J. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. AMS, 1998.

3 Pankaj K. Agarwal, Esther Ezra, and Micha Sharir. Semi-algebraic off-line range searching and biclique partitions in the plane. In *40th International Symposium on Computational Geometry, SoCG 2024*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

4 Édouard Bonnet, Sergio Cabello, and Wolfgang Mulzer. Maximum matchings in geometric intersection graphs. *Discrete & Computational Geometry*, 70:550–579, 2023. doi:10.1007/s00454-023-00564-3.

5 Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2012. doi:10.1137/1.9781611972238.

6 Sergio Cabello, Siu-Wing Cheng, Otfried Cheong, and Christian Knauer. Geometric matching and bottleneck problems, 2023. arXiv:2310.02637.

7 John Gunnar Carlsson, Benjamin Armbruster, Saladi Rahul, and Haritha Bellam. A bottleneck matching problem with edge-crossing constraints. *International Journal of Computational Geometry & Applications*, 25:245–261, 2015. doi:10.1142/S0218195915500144.

8 Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. Algorithms for bichromatic line-segment problems and polyhedral terrains. *Algorithmica*, 11(2):116–132, 1994. doi:10.1007/BF01182771.

9 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 *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022. Full version at https://arxiv.org/abs/2203.00671. doi:10.1109/FOCS54457.2022.00064.

10 David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discret. Comput. Geom.*, 37(1):103–120, 2007. doi:10.1007/s00454-006-1276-5.

11 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd ed. edition, 2008. doi:10.1007/978-3-540-77974-2.

12 Tamal Krishna Dey and Yusu Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2022. doi:10.1017/9781009099950.

13 Herbert Edelsbrunner and John Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010. URL: http://www.ams.org/bookstore-getitem/item=MBK-69.

14 Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31:1–28, 2001.

15 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Perfect matchings in $O(n \log n)$ time in regular bipartite graphs. *SIAM J. Comput.*, 42(3):1392–1404, 2013. doi:10.1137/100812513.

16 Sariel Har-Peled and Everett Yang. Approximation algorithms for maximum matchings in geometric intersection graphs. In *38th International Symposium on Computational Geometry, SoCG 2022*, volume 224 of *LIPIcs*, pages 47:1–47:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SoCG.2022.47.

17 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

18 A. Karim Abu-Affash, Paz Carmi, Matthew J. Katz, and Yohai Trabelsi. Bottleneck non-crossing matching in the plane. *Computational Geometry*, 47:447–457, 2014. doi:10.1016/j.comgeo.2013.10.005.

19 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26:1384–1408, 1997. doi:10.1137/S0097539794268649.

20 Matthew J. Katz and Micha Sharir. Bottleneck matching in the plane. *Comput. Geom.*, 112:101986, 2023. doi:10.1016/j.comgeo.2023.101986.

21 Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. Geometry helps to compare persistence diagrams. *ACM J. Exp. Algorithmics*, 22, 2017. doi:10.1145/3064175.

22 Valerie King, S. Rao, and Robert Endre Tarjan. A faster deterministic maximum flow algorithm. *J. Algorithms*, 17(3):447–474, 1994. doi:10.1006/jagm.1994.1044.

23 Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2005.

24 Cheng Long and Raymond Chi-Wing Wong. Optimal worst-case matching. In Shashi Shekhar, Hui Xiong, and Xun Zhou, editors, *Encyclopedia of GIS*, pages 1511–1522. Springer, 2017. doi:10.1007/978-3-319-17885-1_1516.

25 Cheng Long, Raymond Chi-Wing Wong, Philip S. Yu, and Minhao Jiang. On optimal worst-case matching. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 845–856. ACM, 2013.

26 Aleksander Mądry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proceedings of the 54th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2013*, pages 253–262, 2013. Full version at https://arxiv.org/abs/1307.2205. doi:10.1109/FOCS.2013.35.

27 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discret. Comput. Geom.*, 10:157–182, 1993. doi:10.1007/BF02573972.

28 Guillaume Moroz and Boris Aronov. Computing the distance between piecewise-linear bivariate functions. *ACM Trans. Algorithms*, 12(1):3:1–3:13, 2016. doi:10.1145/2847257.

29 James B. Orlin. Max flows in $O(nm)$ time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013. doi:10.1145/2488608.2488705.

30 James B. Orlin and Xiao-Yue Gong. A fast maximum flow algorithm. *Networks*, 77:287–321, 2021.

31 Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26:362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

32 Robert E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.

33 Leong Hou U, Man Lung Yiu, Kyriakos Mouratidis, and Nikos Mamoulis. Capacity constrained assignment in spatial databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 15–28, 2008. doi:10.1145/1376616.1376621.

34 Haitao Wang and Yiming Zhao. Improved algorithms for distance selection and related problems. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 101:1–101:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.101.