Improvements on Bottleneck Matching Using Geometry

Alon Efrat* Alon Itai†

September 3, 1995

Abstract

Let \( A \) and \( B \) be two sets of \( n \) points in the plane. We present an \( O(n^{1.5+\varepsilon}) \) time algorithm that matches the points of \( A \) to the points of \( B \) such that the distance between the farthest matched pair is minimal. Let the length of the matching denote this distance. We also show that our algorithm can be applied when \( A \) and \( B \) are point-sets in \( \mathbb{R}^d \), and the distance is measured using the \( L_\infty \) norm. This technique can be used to find in time \( O(n^{5+\varepsilon}) \) a translation of \( B \) for which the distance of its matching to \( A \) is smaller than a pre-determined parameter \( \rho \) (or deduce, that no such matching exist). This improves the previous result of Alt et al. [4] by a factor of nearly \( n \).

1 Introduction

A graph-matching of a bipartite graph \( G = (A \cup B, E) \) is a set of edges \( M \subseteq E(G) \) such that no vertex of \( G \) is incident to more than one edge of \( M \). A graph-matching \( M \) is perfect if every vertex of \( G \) is adjacent to an edge of \( M \). The problem of finding a perfect matching in a bipartite (or arbitrary) graph has been well studied. See for example [16, 17] for textbooks on this subject. The best known algorithm for finding a perfect matching in a bipartite graph runs in time \( O(m\sqrt{n}) \) (where \( n \) is the number of vertices and \( m \) is the number of edges) and is due to Micali and Vazirani [18]. When

*School of Mathematical Sciences, Tel Aviv University, Tel-Aviv 69982, Israel
alome@math.tau.ac.il

†Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel.
itai@cs.technion.ac.il
weights are associated to each edge, and we seek a perfect matching whose sum of distances is minimal, the best known algorithm runs in time \(O(n^3)\) using the so called \textit{Hungarian method}, and is due to Kuhn [15].

However, faster algorithms can be obtained for certain types of graphs. We are interested in “geometric graphs” when the vertices of the graph are sets \(A\) and \(B\) of \(n\) points in the Euclidean space \(\mathbb{R}^d\), and the matching we seek is a bijection between \(A\) and \(B\). Vaidya [20] presented an algorithm for finding the matching between \(A\) and \(B\) in the plane, whose sum of distances between the matched points is minimal (among all matchings between \(A\) and \(B\)). He explored the geometric features of the environment to deduce an \(O(n\log n)\) time algorithm for this problem and \(O(n^2\log n)\) time algorithm when the distance is measured in the \(L_\infty\) norm. The solution of the Euclidean case was recently improved by Agarwal et al. [2] to \(O(n^{2+\varepsilon})\) (for any \(\varepsilon > 0\)). See also [6] for a very fast algorithms for other types of graphs.

In this paper we address the (easier) \textit{min-max} or \textit{Bottleneck} problem of finding a minimal-length matching between \(A\) and \(B\) when the length of a matching is the Euclidean distance between the furthest matched pair. A matching is \textit{minimal} if its length is minimal (among all matchings between \(A\) and \(B\)); define \(\text{Match}(A,B)\) to be this length. We show that finding \(\text{Match}(A,B)\) is simpler to tackle than the sum-of-distances problem, by presenting an \(O(n^{1.5+\varepsilon})\) time algorithm for this problem, for any \(\varepsilon > 0\).

We show in Section 6 that when the distance is measured in the \(L_\infty\) norm, the solution is easily extended to any dimension \(d \geq 2\), and the running time is slightly improved to \(O(n^{1.5} \log^{d+1} n)\). We also show in Section 7 an application of our technique for the following problem: Let \(A\) and \(B\) be two \(n\)-point sets in the plane, and \(\rho\) a fixed number. The problem is to find a translation \(B'\) of \(B\) such that \(\text{Match}(A,B')\) is at most \(\rho\), or determine that no such translation exists. This problem often arises in the field of \textit{pattern matching}, where \(A\) and \(B\) represent shapes or pictures, and \(\text{Match}(A,B)\) measures the similarity (or as it is sometimes called, resemblance) between these pictures. Alt et al. [4] gave an \(O(n^6)\) time algorithm for this problem. We show how to use the technique described in Section 3 to improve the running time of the algorithm of [4] to \(O(n^{5+\varepsilon})\).

For computing \(\text{Match}(A,B)\) we use parametric search. We first introduce an oracle that determines for a parameter \(r\) whether \(r \leq \text{Match}(A,B)\). Later, this oracle will be used to find \(r^*\), the minimal \(r\) for which a perfect matching exists.

Let \(G[r]\) be the bipartite graph whose vertex set is \(A \cup B\), and whose edges consist of all pairs \((a,b)\) \(a \in A, b \in B\) for which the distance between \(a\) and \(b\) is at most \(r\). Note that \(\text{Match}(A,B) \leq r\) if and only if there exists a perfect graph-matching in \(G[r]\). We therefore focus on finding a maximum graph-matching in \(G[r]\)—a graph-
matching of largest cardinality.

2 Optimal Matchings in Bipartite graphs

Given a graph-matching $M$ of a bipartite graph $G = (A \cup B, E)$, the vertices incident to edges of $M$ are called matched and the remaining vertices are exposed. The path $\pi = (v_1, \ldots, v_2t)$ is an alternating path if $v_1$ is an exposed vertex of $A$, $v_{2i}$ is an exposed vertex of $B$, $(v_{2i}, v_{2i+1}) \in M$ and $(v_{2i-1}, v_{2i}) \in E \setminus M$ ($i = 1, \ldots, t$). Note that the odd vertices of $\pi$ belong to $A$, and the even ones to $B$. If $\pi$ is an alternating path then $M' = M \oplus \pi = (M \setminus \pi) \cup (\pi \setminus M)$ is a graph-matching too and $|M'| = 1 + |M|$.

A theorem of Berge [5] and of Norman and Rabin [19] states that a matching is maximum if and only if there are no augmenting paths. Thus one may start with the empty matching and augment it by alternating paths found in a greedy fashion.

Edmonds and Karp [9] found alternating paths by order of increasing length. Instead of finding the alternating paths one by one, Hopcroft and Karp [12] and Dinitz [8] find all shortest alternating paths together. We follow the terminology of Dinitz’s algorithm.

To find all shortest alternating paths, we conduct a breadth-first-search to get layers $L_1, \ldots, L_{2t}$. The first layer, $L_1$, consists of all exposed vertices of $A$; $L_{2i}$ contains all vertices of $B$ not appearing in $\bigcup_{j \leq 2i} L_j$ and connected (in $G$) to some vertex of $L_{2i-1}$. If $L_{2i}$ contains exposed vertices, then it is the last layer. Otherwise, we define $L_{2i+1}$ to contain all vertices connected (in the matching $M$) to vertices in $L_{2i}$. Note that the odd layers contain only vertices of $A$ and the even layers only vertices of $B$.

The layer graph $L$ consists of the vertex set $\bigcup_{i=1}^{2t} L_i$, and edges of $M$ that connect vertices of $L_{2j}$ to vertices of $L_{2j+1}$, and edges of $G$ that connect vertices of $L_{2j-1}$ to vertices of $L_{2j}$.

Dinitz found a maximal set of edge-disjoint alternating paths by conducting a depth-first-search of the layer graph. His algorithm requires $O(|E|)$ time to construct the layer graph and find the alternating paths. For sufficiently large values of $r$, $G[r]$ contains $\theta(r^2)$ edges, hence his algorithm requires $O(n^2)$ time per layer graph. We take advantage of the geometric features of $G[r]$ to improve the efficiency of Dinitz’s algorithm. We will represent the edges of $L$ implicitly, and thus our construction will enable us to find the alternating paths in $L$ in time $O(n^{1+\epsilon})$.

\footnote{A similar algorithm appeared also in [13]}
3 Constructing \( \mathcal{L} \) implicitly

Our goal is to find the set of vertices of each layer \( L_i \). For this, we maintain several dynamic Voronoi diagrams. Agarwal and Matoušek [3] introduced a dynamic Voronoi diagram for a point-set \( S \), denoted by \( \text{DVD}(S) \), that allows us to find the nearest neighbor of \( S \) to a query point \( q \) and delete or insert a point into \( S \) in time \( O(n^{1+\varepsilon}) \) per operation, for any fixed \( \varepsilon > 0 \). The first dynamic Voronoi diagram, \( U \), initially contains all the vertices of \( B \), but in the course of the algorithm some of its vertices will be deleted. Using this data structure, the layer graph is constructed by the following procedure:

\[
\begin{align*}
L_1 &\leftarrow \text{exposed vertices of } A ; \quad i \leftarrow 1 ; \quad U \leftarrow \text{DVD}(B) \\
\text{Repeat forever} & \\
L_{2i} &\leftarrow \emptyset \\
\text{For each } x \in L_{2i-1} & \text{ Do} \\
\quad \text{Repeat find } x's \text{ closest neighbor } y \text{ in } U, \text{ as long as } ||y-x|| \leq r & /\!\!\!\!\!\!\text{ /* We find the } y's \text{ connected in } G[r] \text{ to some } x \in L_{2i-1} */ \\
\quad \text{Add } y \text{ to } L_{2i} & \\
\quad \text{Delete } y \text{ from } U. & /\!\!\!\!\!\!\text{ /* To prevent finding } y \text{ again */} \\
\text{End} & \\
\text{If } L_{2i} \text{ is empty} & \text{Then no augmenting path exists. Stop.} \\
\text{Else If } L_{2i} \text{ contains exposed vertices,} & \text{Then the construction of } \mathcal{L} \text{ is complete.} \\
\text{Else let } L_{2i+1} &\subseteq A \text{ be all vertices adjacent to } L_{2i} \text{ via edges of } M. \\
& i \leftarrow i + 1 \\
\text{End} &
\end{align*}
\]

Each matched vertex of \( A \) is reached in \( O(1) \) time from its pair in \( M \). Also, each vertex of \( B \) is found at most once via a nearest neighbor query (using \( U \)), and deleted from \( U \) at most once. Since each such operation requires \( O(n^{1+\varepsilon}) \) time, the time for constructing \( \mathcal{L} \) is \( O(n^{1+\varepsilon}) \).

4 Finding alternating paths in \( \mathcal{L} \)

We now show that any maximal set of edge disjoint alternating paths are vertex disjoint.
Lemma 4.1 Let $M$ be a graph-matching of a bipartite graph $G = (A \cup B, E)$, $\Pi$ be a set of edge-disjoint alternating paths, and let $v$ be an intermediate vertex of some path of $\Pi$. Then $v$ cannot participate in any other alternating path of $\Pi$.

Proof: Since $v$ is neither the first nor the last vertex of the alternating path, $v$ is not exposed so it must be incident to exactly one edge $(v, v') \in M$. Suppose $v \in L_{2i}$. By our construction, $(v, v')$ connects $L_{2i}$ and $L_{2i+1}$. Hence, every alternating path that contains $v$ must also contain the edge $(v, v')$. Since the paths of $\Pi$ are edge disjoint, $v$ cannot belong to any other path of $\Pi$. 

We now construct a dynamic Voronoi diagram $U_{2i}$ for each of the even layers $L_{2i} \subseteq B$. Next we look for an alternating path from an exposed vertex of $L_1$ to an exposed vertex of $L_{2i}$ (the last layer). We do it in a DFS fashion: When scanning a vertex $v \in L_{2i-1}$ we need to find a neighbor of this vertex in $L_{2i}$. To this end we perform a query in $U_{2i}$ to find $u$, the nearest-neighbor of $v$ in $L_{2i}$, and we check if $||u - v|| \leq r$; that is, is $(v, u)$ an edge of $\mathcal{L}$. If such $u$ has found, we advance to $v$. To advance from a vertex $b \in L_{2j}$, we use the graph-matching $M$: If $b$ is not exposed, then there exists an edge $(b, a) \in M$, $a \in L_{2j+1}$.

The path we have found is stored in a stack. On finding a vertex $u \in L_{2i-1}$ from which we cannot advance to $L_{2i}$, we return to the vertex $w$ that led us to $u$ by popping it from the stack, delete $w$ from $L_{2j-2}$ and from $U_{2j-2}$, (since $w$ is useless for any future path in $\mathcal{L}$), and continue the search.

If we reach an exposed vertex of $L_{2i}$, then an alternating path is found. We improve the matching, and delete all the vertices of the path from the layer graph, and from the Voronoi diagrams. (This is justified by Lemma 4.1.)

The process terminates when there are no more exposed vertices in $L_1$. Recall that a vertex is deleted from the graph if either it is matched, or it doesn’t lead to any exposed vertex of $L_{2i}$. In the latter case there exists no alternating path from it. This implies that this vertex cannot be matched in $G[r]$, and the oracle should state that $r^* > r$.

Note that the time spent on finding all alternating paths in a single layer graph is again $O(n^{1+\varepsilon})$. To bound the total time required for the algorithm, we need the following lemma, which is due to Hopcroft and Karp [12].

Lemma 4.2 Only $O(\sqrt{n})$ phases of the algorithm are performed until the size of $M$ reaches $n - \sqrt{n}$.

Since the maximal size of the matching is $n$, and in each layer graph we improve the matching (see [16]), at most $\sqrt{n}$ phases are needed after size of $|M|$ reaches $n - \sqrt{n}$. 

Bottleneck-Matching September 3, 1995
By Lemma 4.2, the entire algorithm performs only $O(\sqrt{n})$ phases. As shown, each such phase required $O(n^{1+\varepsilon})$ time. Hence the total time required for the oracle is $O(n^{1.5+\varepsilon})$.

5 Finding $r^*$

In this section we show how we use the oracle to find $r^*$ — the minimal $r$ for which a perfect matching exists.

Let $S$ be a set of points in the plane and $1 \leq k \leq \binom{n}{2}$ an integer. Consider the sequence of $\binom{n}{2}$ distances between pairs of points in $S$. M. Katz [14] showed that the $k$'th largest distance in this sequence can be found in time $O(n^{4/3}\log^2 n)$. A simpler and slightly less efficient algorithm for this problem (but still sufficient for our purpose) appeared in [1]. Note that $r^*$ is a distance between a pair of points in $A \cup B$, and applying this algorithm, for the points of $A \cup B$, we perform a binary search among the $\binom{2n}{2}$ distances, using the oracle to determine if a given distance is larger or smaller than $r^*$, and increase or decrease $k$ accordingly. Thus each iteration consists of finding a value for $r$ ($O(n^{4/3}\log^2 n)$ time) and consulting the oracle ($O(n^{1.5+\varepsilon})$ time). Thus each iteration requires $O(n^{1.5+\varepsilon})$ time. Since there are $O(\log n)$ iterations, we have shown:

**Theorem 5.1** Let $A$ and $B$ be two sets of $n$ points in the plane. $\text{Match}(A, B)$ can be found in time $O(n^{1.5+\varepsilon})$, for every $\varepsilon > 0$.

6 Finding $\text{Match}(A, B)$ in higher dimensions

The oracle can be easily extended to the case where $A$ and $B$ are two point-sets in $\mathbb{R}^d$, for arbitrary (constant) $d$, and the distance is measured using the $L_\infty$ norm. In [20] Vaidya showed how to use orthogonal range query data structure for maintaining nearest-neighbor questions (in the $L_1$ or $L_\infty$ norm) under insertion and deletion of points, such that each operation take $O(\log^d n)$. Replacing this data structure for the Dynamic Voronoi diagram of Agarwal and Matoušek [3] yields an oracle that finds in $O(n^{1.5}\log^d n)$ time whether $\text{Match}(A, B) \leq r$.

Performing the binary search is more involved, since Katz’s algorithm [14] and the one of Agarwal et al. [1] do not extend directly to higher dimensions. For this we need a different approach, first described by Chew and Kedem [7]. Note that when $L_\infty$ is the underlying norm, $r^*$ is the distance between the projection of some $a \in A$...
and $b \in B$ on one of the axes $X_i$. We use the oracle to perform a binary search among all such projection on each of the axes. Consider the $i$-axis. Let $a_1, \ldots, a_n$ (resp. $b_1, \ldots, b_n$) be the projection of $A$ (resp. $B$) on these axes in increasing order. Consider the matrix $D = (d_{ij})$ where $d_{ij} \equiv a_i - b_j$. Note that all rows and all columns of $D$ are sorted. Frederickson and Johnson [11, 10] show how to find a critical value in such an (implicitly stored) matrix using $O(\log n)$ calls to the oracle. Repeating this process for all $d$-axis we have shown

**Theorem 6.1** Let $A$ and $B$ be two sets of $n$ points in $\mathbb{R}^d$. $\text{Match}(A, B)$ can be found in time $O(n^{1.5} \log^{d+1} n)$.

**Remark 6.2:** Note that for $d = 2$ the solution for the $L_\infty$ norm is asymptotically faster than that of Theorem 5.1.

### 7 Perfect Matching Under Translation

Let $A$ and $B$ be two sets of $n$ points in $\mathbb{R}^2$, and $\rho$ a fixed parameter. For a translation $\tau \in \mathbb{R}^2$ let $\tau + B$ denote the set $B$ translated by $\tau$. The problem we discuss in this section is to find a translation $\tau \in \mathbb{R}^2$ for which $\text{Match}(A, \tau + B) \leq \rho$, or determine that no such $\tau$ exists. Alt et al. [4] showed how to solve this problem in time $O(n^6)$. We use our technique to improve the running time of their algorithm to $O(n^{5+\varepsilon})$.

Let us briefly describe their algorithm, and refer the reader to their paper for details: If for a translation $\tau$, $\text{match}(A, \tau + B) \leq \rho$ then there also exists a translation $\tau'$ and a pair of points $a \in A$, $b \in B$ such that $\text{match}(A, \tau' + B) = \rho$ and the distance from $a$ to $\tau + b$ is exactly $\rho$. Hence we can concentrate on translations $\tau$ which bring some point of $A$ to distance $\rho$ (exactly) of some point $b \in B$. Let $A^\rho$ denote the set of disks of radius $\rho$ centered at all points of $A$. Let $a \in A$, $b \in B$ and let $C_{a \bar{b}}$ denote all translations that bring $a$ to distance $\rho$ from $b$. The algorithm checks for each pair $a \in A$, $b \in B$ if $\text{Match}(A, \tau + B) \leq \rho$ for some translation $\tau \in C_{a \bar{b}}$. That is, if there exists a perfect matching in the graph $G_{\tau} [\rho]$ determined by $A$ and $\tau + B$. Let $\tau_0$ to be a fixed translation on $C_{a \bar{b}}$. We first evaluated $\text{Match}(A, \tau_0 + B)$. If its value is less than or equal to $\rho$ then we are done. Otherwise, we translate $B$ rigidly by all translations of $C_{a \bar{b}}$. During this process points of $B$ are moved into, or out of disks of $A^\rho$, implying that edges are inserted into or deleted from the graph $G_{\tau} [\rho]$. After each such event, referred to as a critical event, we might need to re-compute $\text{Match}(A, \tau + B)$. This might force us to find an augmenting path in $G_{\tau} [\rho]$, and update the matching.
Alt et al. [4] use a standard graph-technique to find such a path (and increase the matching accordingly), and hence spend $O(n^2)$ time for each critical event. As easily seen, the number of critical events for a pair $a, b$ is $O(n^2)$, and summed over all pairs $a \in A, b \in B$, the total number of events encountered in the course of the algorithm is $O(n^4)$, so the total time spent by the algorithm of Alt et al. is $O(n^4) \times O(n^2) = O(n^6)$.

The improvement we suggest is achieved by replacing the procedure for finding an augmented path by the procedure described in Section 3. This procedure required only $O(n^{1+\varepsilon})$ time for such an operation, and hence, taken over all $O(n^4)$ critical events, the total time sums to $O(n^{1+\varepsilon})$. Hence we have shown:

**Theorem 7.1** Let $A$ and $B$ be two sets of $n$ points in $\mathbb{R}^2$, and $\rho$ a fixed parameter. We can find in time $O(n^{5+\varepsilon})$, for every $\varepsilon > 0$, a translation $\tau$ for which $\text{Match}(A, \tau + B) \leq \rho$, or determine that no such $\tau$ exists.

**Acknowledgment.**

Discussions with Pankaj K. Agarwal, Esther M. Arkin and Yefim Dinitz contributed significantly to this paper, and we are grateful to them.

**References**


*Bottleneck-Matching*  

*September 3, 1995*


Bottleneck-Matching September 3, 1995