A new algorithm for Many to Many Matching with Demands and Capacities

Fatemeh Rajabi-Alni

*Department of Computer Engineering, Islamic Azad University, North Tehran Branch, Tehran, Iran.

Abstract

Let $A = \{a_1, a_2, \ldots, a_s\}$ and $\{b_1, b_2, \ldots, b_t\}$ with $s + r = n$, the many to many point matching with demands and capacities matches each point $a_i \in A$ to at least $\alpha_i$ and at most $\alpha_i'$ points in $B$, and each point $b_j \in B$ to at least $\beta_j$ and at most $\beta_j'$ points in $A$ for all $1 \leq i \leq s$ and $1 \leq j \leq t$. In this paper, we present an $O(n^4)$ time and $O(n)$ space algorithm for this problem.

Keywords:
many to many matching, Hungarian method, bipartite graph, points with demands and capacities

1. Introduction

A matching between two sets defines a relationship between their elements. The matching is used in various fields such as computational biology [1], pattern recognition [2], computer vision [3], music information retrieval [4], and computational music theory [5]. A many-to-many matching between $A$ and $B$ assigns each point in $A$ to one or more points in $B$, and vice versa.

Let $A$ and $B$ be two sets with $|A| + |B| = n$, Eiter and Mannila [6] proposed an $O(n^3)$ algorithm for the minimum many-to-many matching problem between $A$ and $B$ by reducing the problem to the minimum-weight perfect matching problem in a bipartite graph.

The minimum many-to-many matching with demands and capacities, here called MMDC matching, is a matching in which each point $a_i \in A$ is matched...
to at least $\alpha_i$ and at most $\alpha'_i$ points in $B$, and each point $b_j \in B$ is matched to at least $\beta_j$ and at most $\beta'_j$ points in $A$, such that sum of the matching costs is minimized. Schrijver [7] solved the MMDC matching problem in strongly polynomial time. In this paper, we present a new algorithm that computes an MMDC matching between $A$ and $B$ in $O(n^4)$ time using $O(n)$ space. In section 2, we review the basic Hungarian algorithm and some preliminary definitions. In section 3, we present our new algorithm.

2. Preliminaries

Given an undirected bipartite graph $G = (A \cup B, E)$, a maximum matching $M$ is a matching that for any other matching $M'$, we have $\text{Weight}(M') < \text{Weight}(M)$. A path with the edges alternating between $M$ and $E - M$ is called an alternating path. Each vertex $v$ that is incident to one edge in $M$ is called a matched vertex; otherwise it is a free vertex. An alternating path that its both endpoints are free is called an augmenting path. Note that if the $M$ edges of an augmenting path is replaced with the $E - M$ ones, its size increases by 1. Let $V = A \cup B$, a vertex labeling function $l: V \rightarrow \mathbb{R}$ assigns a label to each vertex $v \in V$. A vertex labeling that in which $l(a) + l(b) \geq \text{Weight}(a, b)$ for all $a \in A$ and $b \in B$ is called a feasible labeling. The equality graph of a feasible labeling $l$ is a graph $G = (V, E_l)$ such that $E_l = \{(a, b)|l(a) + l(b) = \text{Weight}(a, b)\}$. The neighbors of a vertex $u \in V$ is defined as $N_l(u) = \{v|(v, u) \in E_l\}$. Consider a set of the vertices $S \subset V$, the neighbors of $S$ is $N_l(S) = \bigcup_{u \in S} N_l(u)$.

Lemma 1. Consider a feasible labeling $l$ of an undirected bipartite graph $G = (A \cup B, E)$ and $S \subset A$ with $T = N_l(S) \neq B$, let

$$\alpha_l = \min_{a_i \in S, b_j \notin T} \{l(a_i) + l(b_j) - \text{Weight}(a_i, b_j)\}.$$  

If the labels of the vertices of $G$ is updated such that:

$$l'(v) = \begin{cases} 
  l(v) - \alpha_l & \text{if } v \in S \\
  l(v) + \alpha_l & \text{if } v \in T \\
  l(v) & \text{Otherwise}
\end{cases}$$

then, $l'$ is also a feasible labeling.

Proof. Note that $l$ is a feasible labeling, so we have $l(a) + l(b) \geq \text{Weight}(a, b)$ for each edge $(a, b) \in E$. After the update four cases arise:
• $a \in S$ and $b \in T$. In this case
\[ l'(a) + l'(b) = l(a) - \alpha_l + l(b) + \alpha_l = l(a) + l(b) \geq \text{Weight}(a, b). \]

• $a \notin S$ and $b \notin T$. We have
\[ l'(a) + l'(b) = l(a) + l(b) \geq \text{Weight}(a, b). \]

• $a \notin S$ and $b \in T$. We see that
\[ l'(a) + l'(b) = l(a) + l(b) + \alpha_l > l(a) + l(b) \geq \text{Weight}(a, b). \]

• $a \in S$ and $b \notin T$. In this situation we have
\[ l'(a) + l'(b) = l(a) - \alpha_l + l(b). \]

Two cases arises:
- $l(a) + l(b) - \text{Weight}(a, b) = \alpha_l$. So
\[ l'(a) + l'(b) = l(a) - \alpha_l + l(b) = l(a) - l(a) - l(b) + \text{Weight}(a, b) + l(b) = \text{Weight}(a, b). \]

Hence, $E_l \subset E_{l'}$.
- $l(a) + l(b) - \text{Weight}(a, b) > \alpha_l$. Obviously
\[ l'(a) + l'(b) = l(a) - \alpha_l + l(b) > \text{Weight}(a, b). \]

\[ \square \]

**Theorem 1.** If $l$ is feasible labeling and $M$ is a Perfect matching in $E_l$, then $M$ is a max-weight matching \[8\].

**Proof.** Suppose that $M'$ is a perfect matching in $G$, since each vertex is incident to exactly one edge of $M'$ we have:

\[ \text{Weight}(M') = \sum_{(a, b) \in M'} \text{Weight}(a, b) \leq \sum_{v \in (A \cup B)} l(v). \]
So $\sum_{v \in (A \cup B)} l(v)$ is an upper bound for each perfect matching. Now assume that $M$ is a perfect matching in $E_l$:

$$Weight(M) = \sum_{e \in M} l(e) = \sum_{v \in (A \cup B)} l(v).$$

It is obvious that $M$ is an optimal matching. $\square$

In the following, we briefly describe the basic Hungarian algorithm which computes the maximum many to many matching between two sets. The input bipartite graph $G = (A \cup B, E)$ is a complete bipartite graph that in which $|A| = |B| = n$.

Algorithm 1 The Basic Hungarian algorithm$(A,B)$

1: Initial $\triangleright$ Find an initial feasible labeling $l$ and a matching $M$ in $E_l$
2: Let $l(b_j) = 0$, for all $1 \leq j \leq t$
3: $l(a_i) = \max_{j=1}^{t} Weight(a_i, b_j)$ for all $1 \leq i \leq s$
4: $M = \emptyset$
5: while $M$ is not perfect do
6: Select a free vertex $a_i \in A$ and set $S = \{a_i\}$, $T = \emptyset$
7: repeat
8: while $N_l(S) = T$ do
9: Update the labels according to Lemma 1
10: Select $b_j \in N_l(S) - T$
11: if $b_j$ is not free then $\triangleright (b_j$ is matched to the vertex $z$, extend the alternating tree) $\triangleright$
12: $S = S \cup z$, $T = T \cup b_j$
13: until $b_j$ is free
14: Augment $M$
return $M$

In line 1, we label all points of $B$ with zero and each point $a_i \in A$ with $\max_{j=1}^{n} Weight(a_i, b_j)$ to get an initial feasible labeling. Note that $M$ can be empty. It is obvious that for computing the minimum cost many to many matching using the Hungarian algorithm we must weight the edge $(a_i, b_j)$ by $1/Weight(a_i, b_j)$. 
Lemma 2. Each augmenting path is a 4-vertex path.

Proof. Suppose that the lemma is false. Let \( p = a_1, b_1, a_2, b_2, \ldots, b_k \) be an augmenting path with more than four vertices, that is \( k > 2 \). Note that \( a_1 \) and \( b_k \) are free nodes. It is obvious that the first edge is in \( E - M \), so the second, third, and fourth edges of \( p \) are in \( M, E - M, \) and \( M \), respectively. Since the third edge \((a_2, b_2)\) is in \( E - M \), the fourth edge \((b_2, a_3)\) must be in \( M \). Note that \( b_2 \) is a free node. A contradiction. \( \square \)

3. The algorithm

In this section, we describe our new algorithm which is based on the well known Hungarian algorithm. Consider two point sets \( A = \{a_1, a_2, \ldots, a_s\} \) and \( B = \{b_1, b_2, \ldots, b_t\} \) with \( s + t = n \). Let \( D_A = \{\alpha_1, \alpha_2, \ldots, \alpha_s\} \) and \( D_B = \{\beta_1, \beta_2, \ldots, \beta_t\} \) denote the demand sets of \( A \) and \( B \), respectively. Let \( C_A = \{\alpha'_1, \alpha'_2, \ldots, \alpha'_s\} \) and \( C_B = \{\beta'_1, \beta'_2, \ldots, \beta'_t\} \) be the capacity sets of \( A \) and \( B \), respectively. Without loss of generality, we assume that \( \sum_{i=1}^{s} \alpha'_i > \sum_{j=1}^{t} \beta'_j \).

Theorem 2. Let \( A \) and \( B \) be two sets with \( |A| + |B| = n \), an MMDC matching between \( A \) and \( B \) can be computed in \( O(n^4) \) time.

Proof.

We first construct a bipartite graph as follows. Consider the complete bipartite graph \( G = (X \cup Y, E) \) where \( X = A \cup A' \) and \( Y = B \cup B' \cup C \) (see Figure 1). A complete connection between two sets is a connection that in which each element of one set is connected to all elements of the other set. We show each set of the vertices by a rectangle and the complete connection between them by a line connecting the two corresponding rectangles.

Given \( A = \{a_1, a_2, \ldots, a_s\} \) and \( B = \{b_1, b_2, \ldots, b_t\} \), there exists a complete connection between \( A \) and \( B \) such that the weight of \((a_i, b_j)\) is equal to the cost of matching the point \( a_i \) to \( b_j \) for all \( 1 \leq i \leq s \) and \( 1 \leq j \leq t \). Let \( B' = \{b'_1, b'_2, \ldots, b'_t\} \) and \( A' = \{a'_1, a'_2, \ldots, a'_s\} \), each point of \( A \) is connected to the all points of \( B' \) such that the weight of \((a_i, b'_j)\) is equal to the weight of \((a_i, b_j)\). There exists also a complete connection between the sets \( B \) and \( A' \) such that the weight of \((a'_i, b_j)\) is equal to the weight of \((a_i, b_j)\). We have a set \( C = \{c_1, c_2, \ldots, c_h\} \) that in which \( h = \sum_{i=1}^{s} \alpha'_i - \sum_{j=1}^{t} \beta'_j \). In fact, we use \( C \) to get \( |X| = |Y| \). Each vertex of \( A' \) is connected to all vertices of \( C \) with zero weighted edges.
Figure 1: Our constructed complete bipartite graph with $h = \sum_{i=1}^{s} \alpha'_i - \sum_{j=1}^{t} \beta'_j$.

Now we apply our new algorithm, Algorithm 2, on above bipartite graph $G$. Let $Cap(u)$ and $Dem(u)$ denote the capacity and the demand of the vertex $u$; so for all $i, j$ we have $Dem(a_i) = \alpha_i$, $Dem(b_j) = \beta_j$, $Cap(a_i) = \alpha'_i$, and $Cap(b_j) = \beta'_j$.

In our algorithm, a vertex $x$ is free to another vertex $y$ if $x$ is not matched with $y$ in $M$ and has at least one empty capacity. So $a_i \in A$ and $a'_i \in A'$ are called free vertices to a vertex $b$ that are not matched with it in $M$, if $Num(a_i) < Dem(a_i)$ and $Num(a'_i) < Cap(a_i) - Dem(a_i)$, respectively. Also the vertices $b_j$ and $b'_j$ are free to another vertex that is not incident in $M$ to them, when $Num(b_j) < Dem(b_j)$ and $Num(b'_j) < Cap(b_j) - Dem(b_j)$, respectively.

In fact, we save the current number of the vertices that are matched to the vertices of $A, B, A', B'$ in the arrays $A[1 \ldots s]$, $B[1 \ldots t]$, $A'[1 \ldots s]$, and $B'[1 \ldots t]$, respectively; for example $A[i]$ shows the number of the nodes that are matched to $a_i$. The initial values of the arrays is 0; when a new point is matched to their representing node their values are increased by 1.

Assume that $Num(u)$ returns the number of the vertices that are matched to $u$ so far. So $Num(a_i) = A[i]$, $Num(a'_i) = A'[i]$, $Num(b_j) = B[j]$, and finally $Num(b'_j) = B'[j]$. Note that the procedures $IsFree(u)$ and $IsMatched(u)$ return True if $Num(u) < Cap(u)$ and $Num(u) = Cap(u)$, respectively. So in the augmenting path $a, b, c, d$, $a$ is free to $b$, $b$ is matched to $c$, and $d$ is free to $c$. Now we change the basic Hungarian algorithm as follows.

We first label the vertices of our bipartite graph $G$ using an initial feasible labeling in lines 2 - 4. Algorithm 2 has a while loop where $O(n^2)$ times iterates and $\sum_{i=1}^{s} \alpha_i + \sum_{j=1}^{t} \beta_j$ edges are selected. In each iteration of our algorithm $|M|$ increases by 1. Let

$$slack_y = \min_{x \in S} \{l(x) + l(y) - Weight(x, y)\}.$$
Algorithm 2 The MMDC Hungarian algorithm\((DA, CA, DB, CB)\)

1: Initialize \(\triangleright\) Find an initial feasible labeling \(l\) and a matching \(M\) in \(E_l\)
2: Let \(l(b_j), l(b'_j) = 0,\) for all \(1 \leq j \leq t\)
3: \(l(a_i) = \max_{j=1}^{t}(\max(Weight(a_i, b_j), Weight(a_i, b'_j)))\) for all \(1 \leq i \leq s\)
4: \(l(a'_i) = \max_{j=1}^{t}(Weight(a'_i, b_j))\) for all \(1 \leq i \leq s\)
5: Let \(M = \emptyset\)

6: while \(\{u \in A \cup A',\) with \(Is\)Free\((u)\}\) \(\neq \emptyset\) do
7: Select \(u \in A \cup A'\) with \(Is\)Free\((u)\)
8: Set \(S = \{u\}, T = \emptyset\)
9: repeat
10: while \(N_l(S) = T\) do
11: \(\triangleright\) Update the labels according to Lemma\([1]\)
12: Let \(\alpha_l = \min_{s_i \in S, t_j \in T}\{l(s_i) + l(t_j) - Weight(s_i, t_j)\}\)
13: Let \(l'(v) = \begin{cases} l(v) - \alpha_l & \text{if } v \in S \\ l(v) + \alpha_l & \text{if } v \in T \\ l(v) & \text{Otherwise} \end{cases}\)
14: Select \(y \in N_l(S) - T\)
15: if \(Is\)Matched\((y)\) then \(\triangleright (\text{Num}(y) = \text{Cap}(y))\)
16: \(\triangleright (y\ is\ matched\ to\ some\ vertices\ z)\)
17: \(S = S \cup \{z| (z, y) \in M\}, T = T \cup \{y\}\).
18: until \(Is\)Free\((y)\)
19: \(Augment(M)\)
In line 17 of Algorithm 2, the values of all slacks must be updated when a vertex is moved from \( \bar{S} \) to \( S \). This is done in \( O(n) \) time. During our algorithm \( s + t = n \) vertices are moved from \( \bar{S} \) to \( S \), so it takes the total time of \( O(n^2) \).

In lines 11, we can compute the value of \( \alpha_l \) by:

\[
\alpha_l = \min_{y \not\in T} \text{slack}_y,
\]

in \( O(n) \) time. After computing the value of \( \alpha_l \) and updating the labels of the vertices, we must also update the values of the slacks. This can be done using:

\[
\forall y \not\in T \text{slack}_y = \text{slack}_y - \alpha_l.
\]

In each iteration the value of \( \alpha_l \) may be computed at most \( O(n) \) times, that takes \( O(n) \) time each time, so running each iteration takes at most \( O(n^2) \) time. Our algorithm has \( O(n^2) \) iteration with \( O(n^2) \) time, so it runs in \( O(n^4) \) time.

\[\square\]

4. Conclusion

In this paper, we presented an \( O(n^4) \) time and \( O(n) \) space algorithm for computing an MMDC matching between \( A \) and \( B \) with total cardinality \( n \). In fact, we modified the basic Hungarian algorithm to get a new algorithm, called the MMDC matching algorithm. Then, we construct a bipartite graph \( G \) and apply our new algorithm on \( G \).

References

[1] A. Ben-Dor, R.M. Karp, B. Schwikowski, R. Shamir, The restriction scaffold problem, J. Comput. Biol. 10 (2003) 385-398.

[2] S.R. Buss, P.N. Yianilos, A bipartite matching approach to approximate string comparison and search, Technical report, NEC Research Institute, Princeton, New Jersey (1995).

[3] M.F. Demirci, A. Shokoufandeh, Y. Keselman, L. Bretzner, S. Dickinson, Object recognition as many-to-many feature matching, Int. J. Comput. Vis. 69 (2006) 203-222.
[4] G.T. Toussaint, A comparison of rhythmic similarity measures, 5th International Conference on Music Information Retrieval, (2004) 242-245.

[5] G.T. Toussaint, The geometry of musical rhythm, Japan Conference on Discrete and Computational Geometry, Berlin-Heidelberg, (2005) 198-212.

[6] T. Eiter, H. Mannila, Distance measures for point sets and their computation, Acta Inform. 34 (1997) 109-133.

[7] A. Schrijver, Combinatorial optimization. polyhedra and efficiency, vol. A, Algorithms and Combinatorics, no. 24, Springer-Verlag, Berlin (2003).

[8] L. R. Foulds, Combinatorial Optimization for Undergraduates, Springer-Verlag, Berlin (1984).