Two exact algorithms for the generalized assignment problem

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 $\{b_1, b_2, \ldots, b_t\}$ be two sets of objects with $s + r = n$, the generalized assignment problem assigns each element $a_i \in A$ to at least $\alpha_i$ and at most $\alpha'_i$ elements in $B$, and each element $b_j \in B$ to at least $\beta_j$ and at most $\beta'_j$ elements 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 using the well known Hungarian algorithm. We also present an $O(n^3)$ algorithm for a special case of the generalized assignment, called the limited-capacity assignment problem, where $\alpha_i, \beta_j = 1$ for all $i, j$.

Keywords:
generalized assignment problem, limited-capacity assignment problem, Hungarian method, complete bipartite graph, objects with demands and capacities

1. Introduction

Given two sets $A$ and $B$, the assignment problem aims to optimally assign each object of one set to at least one object of the other set. The assignment problem has applications in various fields such as computational biology, pattern recognition, computer vision, music information retrieval, and computational music theory. Let $A$ and $B$ be two sets with $|A| + |B| = n$, Eiter and Mannila proposed an $O(n^3)$ algorithm for the assignment

*Corresponding author.
Email address: fatemehrajabialni@yahoo.com (Fatemeh Rajabi-Alni)
problem between $A$ and $B$ by reducing it to the minimum-weight perfect matching problem in a bipartite graph.

In this paper, we consider the generalized assignment between $A$ and $B$, called GA problem, where each point $a_i \in A$ is assigned to at least $\alpha_i$ and at most $\alpha'_i$ points in $B$, and each point $b_j \in B$ is assigned to at least $\beta_j$ and at most $\beta'_j$ points in $A$, such that sum of the matching costs is minimized. We also present an $O(n^3)$ time algorithm for a special case of the GA problem, where each object must be matched to at least one object. Schrijver \cite{schrijver1982} solved the GA problem in strongly polynomial time. We present a new algorithm which computes a generalized assignment between $A$ and $B$ in $O(n^4)$ time using $O(n)$ space. In section \ref{sec:preliminaries} we review the basic Hungarian algorithm and some preliminary definitions. In section \ref{sec:new_algorithm} we present our new algorithms.

2. Preliminaries

Given an undirected bipartite graph $G = (A \cup B, E)$, a matching in $G$ is a subset of the edges $M \subseteq E$, such that each vertex $v \in A \cup B$ is incident to at most one edge of $M$. Let $\text{Weight}(a,b)$ denote the weight of the edge $(a,b)$, the weight of the matching $M$ is the sum of the weights of all edges in $M$, hence

$$\text{Weight}(M) = \sum_{e \in M} \text{Weight}(e).$$

A max-weight matching $M$ is a matching that for any other matching $M'$, we have $\text{Weight}(M') \leq \text{Weight}(M)$.

A path with the edges alternating between $M$ and $E - M$ is called an alternating path. Each vertex $v$ that is incident to an 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.

A vertex labeling is a function $l : V \rightarrow \mathbb{R}$ that 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 $S \subseteq V$, 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 \( S \subseteq A \) 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 such that \( E_l \subseteq E_{l'} \).

Proof. Note that \( l \) is a feasible labeling, so we have

\[ l(a) + l(b) \geq \text{Weight}(a, b) \]

for each edge \((a, b)\) of \( E \).

After the update four cases arise:

- **a ∈ S and b ∈ 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 ∉ S and b ∉ T.** We have
  \[ l'(a) + l'(b) = l(a) + l(b) \geq \text{Weight}(a, b). \]

- **a ∉ S and b ∈ T.** We see that
  \[ l'(a) + l'(b) = l(a) + l(b) + \alpha_l > l(a) + l(b) \geq \text{Weight}(a, b). \]

- **a ∈ S and b ∉ 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, \( E_l \subseteq E_{l'} \).
\[-l(a) + l(b) - \text{Weight}(a, b) > \alpha_l. \text{ Obviously}
\]
\[l'(a) + l'(b) = l(a) - \alpha_l + l(b) > \text{Weight}(a, b).\]

\[\square\]

**Theorem 1.** If \(l\) is a 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 \(E_l\):

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

It is obvious that \(M\) is an optimal matching. \(\square\)

In the following, we briefly describe the basic Hungarian algorithm which computes a max-weight perfect matching in an undirected bipartite graph \(G = (A \cup B, E)\) with \(|A| = |B| = n\). It is obvious that for computing the minimum cost many to many matching using the Hungarian algorithm we must weight each edge \((a_i, b_j)\) by \(\frac{1}{\text{Weight}(a_i, b_j)}\).

In lines 2 and 3, we label all vertices of \(B\) with zero and each vertex \(a_i \in A\) with \(\max_{j=1}^n \text{Weight}(a_i, b_j)\) to get an initial feasible labeling. Note that \(M\) can be empty. In each iteration of the while loop of lines 5 - 21, two free nodes \(a_i\) and \(b_j\) are matched, so it iterates \(O(n)\) times. Using the array \(\text{slack}[1..n]\), we can run each iteration of this loop in \(O(n^2)\) time. The repeat loop runs at most \(O(n)\) times until finding a free node \(b_j\). In line 11, we can compute the value of \(\alpha_l\) by:

\[
\alpha_l = \min_{b_j \notin T} \text{slack}[j],
\]
Algorithm 1 The Basic Hungarian algorithm($A,B$)

1: Initial \quad \triangleright \text{Find an initial feasible labeling $l$ and a matching $M$ in $E_l$}
2: \quad \text{Let } l(b_j) = 0, \text{ for all } 1 \leq j \leq t
3: \quad l(a_i) = \max_{j=1}^{t} \text{Weight}(a_i,b_j) \text{ for all } 1 \leq i \leq s
4: \quad M = \emptyset
5: \textbf{while } M \text{ is not perfect } \textbf{do}
6: \quad \text{Select a free vertex } a_i \in A \text{ and set } S = \{a_i\}, T = \emptyset
7: \quad \textbf{for } j \leftarrow 1, n \textbf{ do}
8: \quad \quad \text{slack}[j] = l(a_i) + l(b_j) - \text{Weight}(a_i, b_j)
9: \quad \textbf{repeat}
10: \quad \quad \textbf{if } N_l(S) = T \textbf{ then}
11: \quad \quad \quad \alpha_l = \min_{b_j \notin T} \text{slack}[j]
12: \quad \quad \quad \textbf{Update}(l) \quad \triangleright \text{Update the labels according to Lemma 1}
13: \quad \quad \textbf{for all } b_j \notin T \textbf{ do}
14: \quad \quad \quad \text{slack}[j] = \text{slack}[j] - \alpha_l
15: \quad \textbf{Select } u \in N_l(S) - T
16: \quad \textbf{if } u \text{ is not free } \textbf{then} \quad \triangleright (u \text{ is matched to a vertex } z, \text{ extend the alternating tree})
17: \quad \quad S = S \cup \{z\}, T = T \cup \{u\}.
18: \quad \textbf{for } j \leftarrow 1, n \textbf{ do}
19: \quad \quad \text{slack}[j] = \min(l(z) + l(b_j) - \text{Weight}(z, b_j), \text{slack}[j])
20: \quad \textbf{until } u \text{ is free}
21: \quad \textbf{Augment}(M)
22: \quad \textbf{return } M
in $O(n)$ time. After computing $\alpha_t$ and updating the labels of the vertices, we must also update the values of the slacks. This can be done using:

$$\text{for all } b_j \notin T, \text{slack}[j] = \text{slack}[j] - \alpha_t.$$  

In line 12, we update the feasible labeling $l$ such that $N_i(S) \neq T$. In line 17 of Algorithm 1 when a vertex is moved from $\bar{S}$ to $S$ the values of $\text{slack}[1..n]$ must be updated. This is done in $O(n)$ time. $O(n)$ vertices are moved from $\bar{S}$ to $S$, so it takes the total time of $O(n^2)$.

The value of $\alpha_t$ may be computed at most $O(n)$ times in $O(n)$, so running each iteration takes at most $O(n^2)$ time. So, the time complexity of the basic Hungarian algorithm is $O(n^3)$.

3. The generalized assignment algorithm

In this section, we describe our new algorithm which is based on the well known Hungarian algorithm. Consider two 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 $\sum_{i=1}^{s} \alpha'_i \geq \sum_{j=1}^{t} \beta'_j$.

The input of our algorithm is the complete bipartite graph that is constructed as follows. Consider the complete bipartite graph $G = (X \cup Y, E)$ where $X = A \cup A'$ and $Y = B \cup B'$ (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 $(a_i, b_j)$.

3.1. The generalized assignment algorithm

**Theorem 2.** Let $A$ and $B$ be two sets with $|A| + |B| = n$, a generalized assignment between $A$ and $B$ can be computed in $O(n^4)$ time.
Figure 1: Our constructed complete bipartite graph with $\sum_{i=1}^{s} \alpha_i' \geq \sum_{j=1}^{t} \beta_j'$.

**Proof.** We apply our new algorithm, Algorithm 2, on our 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 $y$ when $x$ 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.

We save the current number of the vertices that are matched to the vertices of $A$, $B$, $A'$, and $B'$ 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 $u$ is free and $u$ is matched, respectively. Therefor, for example
in the augmenting path $a, b, c, d$, the vertex $a$ 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. In each iteration of our algorithm, $|M|$ increases by
1. So, our algorithm has $O(n^2)$ iterations with $O(n^2)$ time and runs in $O(n^4)$ time.

3.2. The limited-capacity assignment algorithm

Now we present an $O(n^3)$ algorithm for the limited-capacity assignment problem, where each object must be assigned to at least one point of the other set and the capacity of each object is limited.

**Theorem 3.** Let $A$ and $B$ be two sets with $|A| + |B| = n$, a limited-capacity assignment between $A$ and $B$ can be computed in $O(n^3)$ time.

**Proof.** We use the bipartite complete graph that is constructed for the generalized assignment problem. We modify the GA algorithm as following.
Algorithm 3 The limited-capacity assignment algorithm (Part I)

1: Initialize ▷ 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: $l(a_i) = \max_{j=1}^{t}(\max(\text{Weight}(a_i, b_j), \text{Weight}(a_i, b'_j)))$ for all $1 \leq i \leq s$
4: $l(a'_i) = \max_{j=1}^{t} \text{Weight}(a'_i, b_j)$ for all $1 \leq i \leq s$
5: Let $M = \emptyset$
6: while $\{u \in A, \text{ with } IsFree(u)\} \neq \emptyset$ do
7: Select $u \in A$ with $IsFree(u)$
8: Set $S = \{u\}, T = \emptyset$
9: for $j \leftarrow 1, t$ do
10: $\text{slack}[j] = \min(l(u) + l(b_j) - \text{Weight}(u, b_j))$
11: $\text{slack}[j + t] = \min(l(u) + l(b'_j) - \text{Weight}(u, b'_j))$
12: repeat
13: if $N_l(S) = T$ then
14: $\alpha_l = \min_{b_j \notin T} \min_{b'_j \notin T} \text{slack}[j], \min_{b'_j \notin T} \text{slack}[j + t]]$
15: Update($l$)
16: Select $y \in N_l(S) - T$
17: if IsMatched($y$) then
18: for all $\{z | (z, y) \in M$ and $z \notin S\}$ do
19: $S \cup \{z\}, T = T \cup \{y\}$
20: for $j \leftarrow 1, t$ do
21: $\text{slack}[j] = \min(l(z) + l(b_j) - \text{Weight}(z, b_j), \text{slack}[j])$
22: $\text{slack}[j + t] = \min(l(z) + l(b'_j) - \text{Weight}(z, b'_j), \text{slack}[j + t])$
23: until IsFree($y$)
24: Augment($M$)
Algorithm 4 The limited-capacity assignment algorithm (Part II)

25: while \( \{u \in B, \text{ with } \text{IsFree}(u)\} \neq \emptyset \) do
26: Select \( u \in B \) with \( \text{IsFree}(u) \)
27: Set \( S = \{u\}, T = \emptyset \)
28: for \( i \leftarrow 1, s \) do
29: \( \text{slack}[i] = \min(l(u) + l(a_i) - \text{Weight}(a_i, u)) \)
30: \( \text{slack}[i+s] = \min(l(u) + l(a'_i) - \text{Weight}(a'_i, u)) \)
31: repeat
32: if \( N_i(S) = T \) then
33: \( \alpha_i = \min(\min_{a_i \notin T} \text{slack}[i], \min_{a'_i \notin T} \text{slack}[i+s]) \)
34: Update(\( l \))
35: Select \( y \in N_i(S) - T \)
36: if \( \text{IsMatched}(y) \) then \( \triangleright (y \text{ is matched to some vertices } z) \)
37: for all \( \{z|(z,y) \in M \text{ and } z \notin S\} \) do
38: \( S \cup \{z\}, T = T \cup \{y\} \)
39: for \( i \leftarrow 1, s \) do
40: \( \text{slack}[i] = \min(l(z) + l(a_i) - \text{Weight}(a_i, z), \text{slack}[i]) \)
41: \( \text{slack}[i+s] = \min(l(z) + l(a'_i) - \text{Weight}(a'_i, u), \text{slack}[i+s]) \)
42: until \( \text{IsFree}(y) \)
43: \( \text{Augment}(M) \)

4. Conclusion

In this paper, we presented an \( O(n^4) \) time and \( O(n) \) space algorithm for computing a generalized assignment between \( A \) and \( B \) with total cardinality \( n \). In fact, we modified the basic Hungarian algorithm to get a new algorithm, called the generalized assignment algorithm. Then, we construct a bipartite
graph $G$ and apply our new algorithm on $G$. We also improved an $O(n^3)$ algorithm for the limited-capacity assignment problem.

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).