Communications in Combinatorics and Optimization
Vol. xx, No. x (xxxx), pp. 1-13
https://doi.org/10.22049/cco.2026.30709.2588

CCO
Commun. Comb. Optim.

Research Article

An $O(n^3)$ time algorithm for the maximum-weight limited-capacity many-to-many matching in bipartite graphs

Fatemeh Rajabi-Alni*, Behrouz Minaei-Bidgoli†

School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran
*rajabi_fatemeh@mail.iust.ac.ir
†b_minaei@iust.ac.ir

Received: 10 June 2025; Accepted: 30 June 2026
Published Online: 12 July 2026

Abstract: Given an undirected bipartite graph $G = (A \cup B, E)$, a *many-to-many matching* (MM) in G matches each vertex v in A (resp. B) to at least one vertex in B (resp. A). In this paper, we consider the *limited-capacity many-to-many matching* (LCMM) in G, where each vertex $v \in A \cup B$ is matched to at least one and at most $Cap(v)$ vertices; the function $Cap : A \cup B \to \mathbb{Z} > 0$ denotes the capacity of v (an upper bound on its degree in the LCMM). We give an $O(n^3)$ time algorithm for finding a maximum (respectively minimum) weight LCMM in G with non-positive real (respectively non-negative real) edge weights, where $|A| + |B| = n$.

Keywords: Hungarian algorithm, many-to-many matching, limited-capacity, bipartite graphs.

AMS Subject Classification: 90C27

1. Introduction

Given two sets of objects A and B, a *many-to-many matching* (MM) matches each object of A (respectively B) to at least one object of B (respectively A). The MM has many uses, including computational biology, pattern recognition, and wireless networks [9, 13–15]. We can represent the sets and their relations using a bipartite graph; for example, one part can represent mutated genes and the other part outlying genes [14]. Given an undirected bipartite graph $G = (A \cup B, E)$, a *matching* in G is a set of vertex-disjoint edges $M \subseteq E$. Denote by $W(e)$ the weight of the edge $e \in E$.

* *Corresponding Author*

The weight of M which is denoted by $W(M)$ is the sum of the weights of all edges in M, hence

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

A *maximum weight matching* (MWM) denoted by M' is a matching that for any other matching M'' we have $W(M'') \leq W(M')$. A *perfect matching* is a subset of edges $PM \subseteq E$ such that every vertex of G is adjacent to exactly one edge of PM. Assume that $|A| = |B| = n$ and $|E| = m$. The first polynomial-time algorithm for computing a *maximum weight perfect bipartite matching* (MWPBM), a maximum weight perfect matching in the bipartite graph G, is the basic Hungarian algorithm [8, 11]. Then, Fredman and Tarjan [3] solved the MWPBM problem in G in $O(mn + n^2 \log n)$ time by implementing the Hungarian algorithm using fibonacci heaps. Later, other algorithms were developed for bipartite graphs with integer edge weights [5, 12]. For more discussion, see [7].

In many applications, the capacities of objects are limited. For example, consider a set of base stations communicating with a set of wireless sensors. The aim is to send data gathered by the sensors to the base stations. The number of sensors that can communicate with each base station is limited by the finite battery storage capacity of the sensors and the limited capacity of radio links of the base stations. In this context, ensuring full utilization of the network is often a primary goal. A lower bound of 0 on vertex degree is insufficient, as it allows for base stations to be idle (a waste of infrastructure) and sensors to be unassigned (leading to data loss). Thus, imposing a uniform lower bound of 1 on all vertices elegantly prevents this, guaranteeing that every base station is active and every sensor's data is collected. While an arbitrary lower bound $l(v) \geq 1$ offers more generality, it might lead to a significantly higher time complexity. The case of a uniform lower bound of 1 is therefore particularly interesting: it enforces the critical requirement of full utilization while maintaining a favorable time complexity of $O(n^3)$.

The *capacity* of a vertex $v \in A \cup B$ denoted by a function $Cap : A \cup B \to \mathbb{Z} > 0$ is the number of vertices that can be matched to v. A *limited-capacity MM* (LCMM) in G is an edge set such that $1 \leq deg(v) \leq Cap(v)$ for all $v \in A \cup B$ (we use $deg(v)$ to refer to the degree of the vertex v in the LCMM). The maximum weight LCMM problem has been solved in the time complexity of $O(W'\sqrt{\beta'}m)$ for integer edge weights [6], where $W' = \max_{e \in E}(W(e))$ and $\beta' = \sum_{v \in A \cup B} Cap(v)$.

In this paper, we present an $O(n^3)$ time algorithm for the maximum weight LCMM problem in G when the edge weights are non-positive real numbers. Note that a maximum weight LCMM in G with non-positive real edge weights $W(e)$ is a minimum weight LCMM in G with non-negative real edge weights $F(e) = -W(e)$ for all $e \in E$. In a degree constrained subgraph (DCS) $H' = (V', E_{h'})$ of a general graph $H = (V', E_h)$ it holds that $l(v) \leq deg(v) \leq u(v)$ for each vertex $v \in V'$ with degree $deg(v)$, where $l(v)$ and $u(v)$ denote integer bounds. The maximum weight LCMM problem in G can be stated as a specific case of the maximum weight DCS problem in a general graph and solved in $O(n^2 \min(m \log n, n^2))$ time [4]. It is true that the maximum

weight LCMM problem can be formulated as a minimum cost flow problem. However, this general formulation does not capitalize on the problem's underlying bipartite structure, potentially leading to computational inefficiencies. Although recent theoretical breakthroughs in minimum cost flow offer superior asymptotic run-times [1], the high complexity of these algorithms casts doubt on their practical performance. We therefore introduce an approach based on the Hungarian algorithm, a classic solution prized for its simplicity and proven efficacy in bipartite matching problems. We first review the basic Hungarian algorithm and some preliminary definitions. Then, we present our new algorithm.

2. Preliminaries

Consider a weighted bipartite graph $G = (A \cup B, E)$ such that all edge weights are non-positive real values. Let $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. We denote by $W(a_i, b_j)$ the weight of the edge (a_i, b_j) for $a_i \in A$ and $b_j \in B$. If there exists no edge between two vertices $a_i \in A$ and $b_j \in B$, we assume that $W(a_i, b_j) = -\infty$. A path with the edges alternating between the edges of the matching 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*. All alternating paths originating from a free vertex $v \in A \cup B$ constitute an *alternating tree*. An alternating path with two free endpoints is called an *augmenting path*. Note that by finding the symmetric difference between M and an augmenting path, the *augmentation* of M which is denoted by $augment(M)$, we get a new matching M' with $|M'| = |M| + 1$ (the size of M increases by 1).

A *vertex labeling* is a function $l : V \to \mathbb{R}^- \cup \{0\}$ with $V = A \cup B$ that assigns a non-positive real value as a label to each vertex $v \in V$. A vertex labeling that in which $l(a_i) + l(b_j) \geq W(a_i, b_j)$ for all $a_i \in A$ and $b_j \in B$ is called a *feasible labeling*. The *equality graph* of a feasible labeling l is a graph $G_l = (V, E_l)$ such that $E_l = \{(a_i, b_j) \in E | l(a_i) + l(b_j) = W(a_i, b_j)\}$. The set of *neighbors* of a vertex $u \in V$ is defined as $N_l(u) = \{v \in V | (v, u) \in E_l\}$. Consider a set of vertices $S \subseteq V$, we define $N_l(S) = \bigcup_{u \in S} N_l(u)$ as the set of neighbors of S.

Lemma 1. *Consider a feasible labeling l of an undirected bipartite graph $G = (A \cup B, E)$. Let $S \subset A$ and $T \subset B$ with $N_l(S) = T$, where S and T represent the vertices of an alternating tree. Let*

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

If the labels of the vertices of G are updated such that:

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

then l' is a feasible labeling such that $E_l \subset E_{l'}$.

Proof. Obviously, in the cases $a_i \in S, b_j \in T$, or $a_i \notin S, b_j \notin T$, or $a_i \notin S, b_j \in T$, we have:

$$l'(a_i) + l'(b_j) \geq l(a_i) + l(b_j) \geq W(a_i, b_j).$$

And, for some vertices $a_i \in S, b_j \notin T$, we have $l'(a_i) + l'(b_j) = l(a_i) - \alpha_l + l(b_j) = W(a_i, b_j)$. □

Theorem 1. *Let l be a feasible labeling such that E_l covers all vertices. If M is a perfect matching in E_l, then M is a maximum 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:

$$W(M') = \sum_{(a_i,b_j)\in M'} W(a_i, b_j) \leq \sum_{v\in(A\cup B)} l(v).$$

Thus, $\sum_{v\in(A\cup B)} l(v)$ is an upper bound for each perfect matching. Now assume that M is a perfect matching in E_l:

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

□

Now, we review the basic Hungarian algorithm which computes an MWPBM in an undirected bipartite graph $G = (A \cup B, E)$ with $|A| = |B| = n$ (see Algorithm 1). It has been shown that the maximum weight matching problem in bipartite graphs can be reduced to the MWPBM problem and solved using the Hungarian algorithm in $O(n^3)$ time [2]. Note that, for all the free vertices $v \in A$, the Hungarian algorithm builds an alternating tree rooted at v to find an augmenting path.

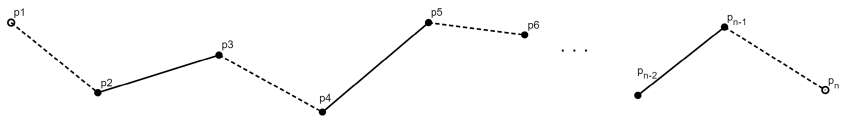

Figure 1. An example for illustration of Lemma 2; the hollow circles are free vertices and the filled circles are matched vertices.

In Lines 2 and 3 of Algorithm 1, the vertices of the input bipartite graph are labeled by a feasible labeling. An initial matching M, which can be empty, is defined in Line 4. In each iteration of the while loop of Lines 5–27, the size of M increases by exactly 1; therefore, the loop iterates at most n times.

Algorithm 1 The Basic Hungarian algorithm($G = (A \cup B, E)$)

1: **Initial** ▷ Find an initial feasible labeling l and a matching M in E_l
2: $l(b_j) = 0$ for all $1 \leq j \leq n$
3: $l(a_i) = \max_{j=1}^{n} W(a_i, b_j)$ for all $1 \leq i \leq n$
4: $M = \emptyset$
5: **while** M is not perfect **do**
6: Select a free vertex $a_i \in A$, and let $S = \{a_i\}$, $T = \emptyset$
7: **for** $j \leftarrow 1$ to n **do**
8: $slack[j] = l(a_i) + l(b_j) - W(a_i, b_j)$
9: **end for**
10: **repeat**
11: **if** $N_l(S) = T$ **then**
12: $\alpha_l = \min_{b_j \notin T} slack[j]$
13: $Update(l)$ ▷ Update the labels according to Lemma 1
14: **for all** $b_j \notin T$ **do**
15: $slack[j] = slack[j] - \alpha_l$
16: **end for**
17: **end if**
18: Select $u \in N_l(S) \setminus T$
19: **if** u is not free **then** ▷ (u is matched to a vertex z; extend the alternating tree)
20: $S = S \cup \{z\}, T = T \cup \{u\}$.
21: **for** $j \leftarrow 1, n$ **do**
22: $slack[j] = \min(l(z) + l(b_j) - W(z, b_j), slack[j])$
23: **end for**
24: **end if**
25: **until** u is free
26: Add (a_i, u) to M or $augment(M)$ so that all new adding edges to M are in E_l
27: **end while**
28: **return** M

The repeat loop of Lines 10–25 iterates at most $O(n)$ times until finding a free vertex u. In Line 12, the value of α_l is computed by:

$$\alpha_l = \min_{b_j \notin T} slack[j],$$

in $O(n)$ time. After computing α_l, the feasible labeling l is updated in Line 13 such that $N_l(S) \neq T$. The values of the slacks must also be updated by:

$$slack[j] = slack[j] - \alpha_l,$$

for all $b_j \notin T$ (Lines 14–15). A vertex $u \in N_l(S) \setminus T$ is selected in Line 18. Observe that if u is not a free vertex, the alternating tree must be extended (Lines 19–20). Note that in the repeat loop, an alternating tree is constructed to find an augmenting path. Once a vertex is moved from $\bar{S}$ to S, the values of $slack[j]$ for all $1 \leq j \leq n$ are updated in $O(n)$ time (Lines 21–22). At most $O(n)$ vertices are moved from $\bar{S}$ to S, so the repeat loop takes the total time of $O(n^2)$. Therefore, the time complexity of the basic Hungarian algorithm is $O(n^3)$.

Lemma 2. *After each augmentation of a matching (Line 26 of Algorithm 1), the weight of the matching does not increase.*

Proof. Given an augmenting path P, two cases arise:

- $P = (p_1, p_2)$. According to non-positive real edge weights, this condition is trivial.
- $P = (p_1, p_2, p_3, \ldots, p_n)$ (see Figure 1). Note that

$$W(p_i, p_{i+1}) = l(p_i) + l(p_{i+1})$$

for $i = 1, 2, \ldots, n-1$, since all edges of an augmenting path are in E_l. Assume for a contradiction that the lemma is false, and thus

$$W(p_1, p_2) + W(p_3, p_4) + \cdots + W(p_{n-1}, p_n) > W(p_2, p_3) + W(p_4, p_5) + \cdots + W(p_{n-2}, p_{n-1}).$$

So, it holds that:

$$l(p_1) + l(p_2) + \cdots + l(p_n) > l(p_2) + l(p_3) + \cdots + l(p_{n-1}),$$

and thus:

$$l(p_1) + l(p_n) > 0.$$

Note that both p_1 and p_n are free, and according to the above feasible labeling we have $l(p_n) \leq 0$ and $l(p_1) \leq 0$, a contradiction. □

The Hungarian algorithm gives the best run time for bipartite graphs with low range edge weights [10]. In the worst case, the repeat loop of the algorithm runs in $O(n^2)$ overall time; the function $Update(l)$ of the algorithm produces a new feasible labeling l' whose equality graph has only one more edge ($|E_{l'}| = |E_l|+1$). However, in bipartite graphs with low range edge weights and dense graphs, after updating the labels, many more edges are added to E_l, substantially decreasing the computational complexity. Low range matrices are used in problems with low precision data.

The maximum weight LCMM algorithm on bipartite graphs

Let $G = (A \cup B, E)$ be a bipartite graph with non-positive real edge weights, where $A = \{a_1, a_2, \ldots, a_s\}$ and $B = \{b_1, b_2, \ldots, b_t\}$ such that $s + t = n$. Let $C_A = \{\alpha_1, \alpha_2, \ldots, \alpha_s\}$ and $C_B = \{\beta_1, \beta_2, \ldots, \beta_t\}$ denote the capacities of A and B, respectively. Assume w.l.o.g that $t \leq s$. Also, assume that $s \leq \sum_{j=1}^{t} \beta_j$ (it is obvious that if $s > \sum_{j=1}^{t} \beta_j$, then there does not exist any LCMM in G). We present

an $O(n^3)$ time algorithm for computing a maximum weight LCMM in $G = (A \cup B, E)$, where each vertex $a_i \in A$ must be matched to at least one and at most α_i vertices in B, and each vertex $b_j \in B$ must be matched to at least one and at most β_j vertices in A for all $1 \leq i \leq s$ and $1 \leq j \leq t$.

Our algorithm is based on the basic Hungarian algorithm. Recall that the basic Hungarian algorithm computes an MWPBM in an undirected bipartite graph. Thus, we first construct a complete bipartite graph $G' = (X \cup Y, E')$ with $X = A \cup A'$ and $Y = B \cup B'$ (see Figure 2), and then run our algorithm on it.

In a *complete connection* between two sets, each element of one set is connected to all elements of the other set. We represent each set of vertices by a rectangle and the complete connection between two sets by a line connecting the two corresponding rectangles.

Given $A = \{a_1, a_2, \ldots, a_s\}$ and $B = \{b_1, b_2, \ldots, b_t\}$, we construct a complete connection between A and B, where the weight of the edge (a_i, b_j) in G' is equal to the weight of the edge (a_i, b_j) in G for all $1 \leq i \leq s$ and $1 \leq j \leq t$.

Let $B'_j = \{b'_{j1}, b'_{j2}, \ldots, b'_{j(\beta_j - 1)}\}$ for $1 \leq j \leq t$, and $B' = \{B'_1, B'_2, \ldots, B'_t\}$. Each vertex of A is connected to all vertices of B' such that

$$W(a_i, b'_{jk}) = W(a_i, b_j)$$

for all $1 \leq k \leq (\beta_j - 1)$.

Let $A'_i = \{a'_{i1}, a'_{i2}, \ldots, a'_{i(\alpha_i - 1)}\}$ for $1 \leq i \leq s$ and $A' = \{A'_1, A'_2, \ldots, A'_s\}$, we also construct a complete connection between the sets B and A' such that

$$W(a'_{ik}, b_j) = W(a_i, b_j)$$

for all $1 \leq k \leq (\alpha_i - 1)$. Also, there exists a complete connection between two sets A' and B' with zero weighted edges.

Now, we modify the basic Hungarian algorithm to get a new algorithm, called $ModifiedHungarianAlg$ (see Algorithm 2). In the modified Hungarian algorithm, Line 5 of Algorithm 1 is changed; the while loop iterates until matching the subset $A \subseteq X$. The initialization step is also removed.

Our new algorithm consists of two steps (see Algorithm 3). In the first step, all vertices of $A \subseteq X$ are matched, together with one or more vertices of B. And, in the second step, the remaining free vertices of $B \subseteq Y$ (the vertices of B that are not matched in the first step) are matched. We claim that by applying our algorithm on G, $Maxweight\ LCMM\ Algorithm(G = (A \cup B, E))$, we get a maximum weight LCMM in G.

Step I. Given an undirected complete bipartite graph $G' = (X \cup Y, E')$ with $X = A \cup A'$ and $Y = B \cup B'$, in this step, we call the function $ModifiedHungarianAlg(G', A, M, l)$ (Line 6 of Algorithm 3). It matches the vertices $a_i \in A$ for all $i = \{1, 2, \ldots, s\}$ until there does not exist any free vertex in $A \subseteq X$.

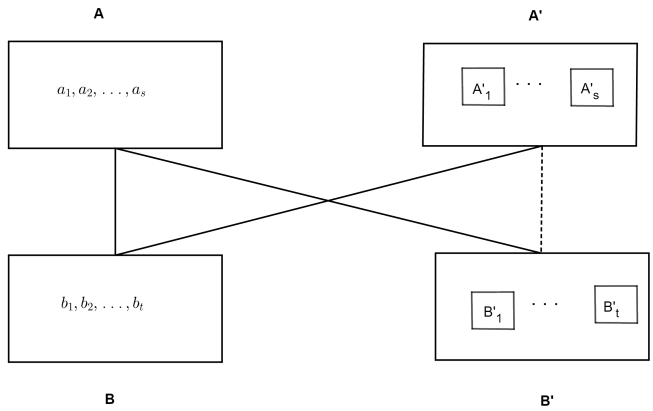

Figure 2. Our constructed complete bipartite graph; the dashed line represents a complete connection with zero weighted edges between A' and B'.

In $ModifiedHungarianAlg(G', A, M, l)$, the while loop of Lines 1–30, called the *main loop*, iterates until each vertex of A is matched to exactly one vertex of $B \cup B'$. Obviously, it iterates $O(n)$ times, since the number of vertices of A is $O(n)$. In the following, we show that each iteration of the main loop of $ModifiedHungarianAlg(G', A, M, l)$ takes $O(n^2)$ time.

Observation 1. The labels of all the free vertices $b'_{jk} \in B'_j$ are equal for all $1 \leq k \leq \beta_j - 1$.

Initially, we have $l(b'_{jk}) = 0$ for all $1 \leq k \leq \beta_j - 1$. The function $Update(l)$ updates the labels of all the vertices $b'_{jk} \in T$, i.e., all the vertices b'_{jk} that have been matched to a vertex in S. Hence, all the free vertices $b'_{jk} \in B'_j$ have equal labels for $1 \leq k \leq \beta_j - 1$.

Observation 2. The slacks of all the free vertices $b'_{jk} \in B'_j$ are equal for all $1 \leq k \leq \beta_j - 1$.

Note that the vertices of B'_j are copies of the vertex b_j. By Observations 1 and 2, all the free vertices of B'_j have equal labels and slacks. Therefore, in each iteration of the main loop, we consider only one of the free vertices $b'_{jk} \in B'_j$ for $1 \leq k \leq \beta_j - 1$, arbitrarily. Actually, in each iteration of the main loop, all the free vertices $b'_{jk} \in B'_j$ are considered a single vertex (Line 5 of Algorithm 2).

Let $B'' = b''_1, b''_2, \ldots, b''_t$, where b''_j is an arbitrary free vertex of B'_j, if exists (Line 6 of Algorithm 2). Let $Y' = B \cup C \cup B''$, where C is the set of the matched vertices of B' with respect to M (Lines 8–9 of Algorithm 2). In each iteration of the main loop, we first give initial values to the slacks of all the vertices $y_j \in Y'$ in $O(n)$ time (Lines 10–12). Then, the repeat loop of Lines 13–28 iterates until

we find a free vertex u and either add (x_i, u) to M or $augment(M)$. Note that if $N_l(S) = T$, we update the vertex labels of G' to get a new feasible labeling such that $N_l(S) \neq T$. The neighbor set of a vertex $u \in A$ is defined as $N_l(u) = \{v \in Y' | (u, v) \in E_l\}$.

Note that there exist at most $O(n)$ matched vertices, i.e. the vertices of A, so the number of the vertices in T and S is at most $O(n)$. Hence, in Line 15, we get the minimum value in $O(n)$ time. Also, updating the labels and slacks takes $O(n)$ time (Lines 16, 17–19 and 24–26).

Observe that in this step, the initial matching M is an empty set (Line 5 of Algorithm 3), and each iteration of the main loop starts from an arbitrary free vertex $a_i \in A$ for $1 \leq i \leq s$. Therefore, by Lemma 2, the output of this step, M_1, is an MWM covering A. Note that in M_1, all vertices of A are matched, but some vertices of B might be free. Let $Btemp = B$ (Line 7 of Algorithm 3). Note that $(a_i, b'_{jk}) \in M_1$ for $a_i \in A$ and $b'_{jk} \in B'_j$ implies that a_i is matched to b_j (since b'_{jk} is a copy of b_j). Thus, if a vertex a_i is matched to a vertex b'_{jk} in M_1, we remove the corresponding vertex b_j from $Btemp$ (Lines 8–12 of Algorithm 3).

Step II. In this step, we call the function $ModifiedHungarianAlg(G', Btemp, M_1, l_1)$ (Line 13 of Algorithm 3). We use the final labels of the vertices from Step I, so the labels of the vertices are feasible. We also use the output matching of Step I, M_1, as the initial matching for Step II. Once all vertices of $Btemp$ are matched, this step terminates. Recall that the initial matching of this step, M_1, is an MWM that covers A. Additionally, observe that each iteration of the main loop of $ModifiedHungarianAlg(G', Btemp, M_1, l_1)$ starts from an arbitrary free vertex $b_j \in B$ for $1 \leq j \leq t$. Therefore, the output of this step, M_2, is an MWM covering $A \cup B$ (if M_2 covers a vertex $b'_{jk} \in B'_j$, we assume that it also covers the vertex b_j). Note that, by Lemma 2, if we continue matching the vertices of G', the weight of M_2 does not increase. Similar to Step I, we observe that the slacks and labels of all the free vertices $a'_{ik} \in A'_i$ (for all $1 \leq k \leq \alpha_i - 1$) are equal in value. Thus, we can show that the time complexity of Step II is also $O(n^3)$.

Observation 3. M_2 does not contain both (a_i, b'_{jk}) and $(a'_{ik'}, b_j)$.

Proof. Assume for contradiction that $(a_i, b'_{jk}) \in M_2$ and $(a'_{ik'}, b_j) \in M_2$. Replace (a_i, b'_{jk}) and $(a'_{ik'}, b_j)$ in M_2 by the edges (a_i, b_j) and $(a'_{ik'}, b'_{jk})$. The result is still a matching that covers $A \cup B$ with a greater weight, this contradicts that M_2 is an MWM covering $A \cup B$. □

We claim that from the output matching of Step II, M_2, we get a maximum weight LCMM in $G = (A \cup B, E)$, denoted by L. In the following, we prove that the weight of M_2 is equal to the weight of L, i.e., $W(L) = W(M_2)$.

Lemma 3. $W(L) \leq W(M_2)$.

Proof. We construct from L a matching M' in G' covering $A \cup B$ such that $W(M') = W(L)$. We relate an edge of G' to each edge of L as follows.
For each $(a_i, b_j) \in L$, one of the following three cases arises:

- Both $a_i \in G'$ and $b_j \in G'$ are free. Then, we add (a_i, b_j) to M'.
- $a_i \in G'$ is free but $b_j \in G'$ is already matched. Then, we add (a_i, b'_{jk}) to M', where b'_{jk} is an arbitrary free vertex of $B'_j \in G'$.
- $a_i \in G'$ is already matched but $b_j \in G'$ is free. In this situation, we add (a'_{ik}, b_j) to M', where a'_{ik} is an arbitrary free vertex of $A'_i \in G'$.

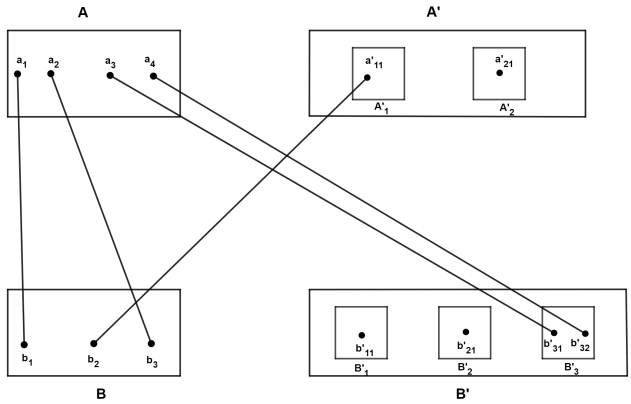

Figure 3. The edges of G' related to the edges of $L = \{(a_1, b_1), (a_1, b_2), (a_2, b_3), (a_3, b_3), (a_4, b_3)\}$.

Thus, for each edge $(a_i, b_j) \in L$, we add the edge (a_i, b_j) to M' if neither $a_i \in G'$ nor $b_j \in G'$ is incident with an edge in M'. Otherwise, we add to M' the edge incident to the free vertex of (a_i, b_j) and an arbitrary free vertex (see Figure 3 for an example). For each $(a_i, b_j) \in L$, we add an edge with equal weight in M', so $W(M') = W(L)$. Since M_2 is a maximum-weight matching in G' that covers $A \cup B$, we have $W(M') \leq W(M_2)$, and thus $W(L) \leq W(M_2)$. □

In the following, we get an LCMM L' in G from the output matching M_2 in G'. Note that for each $a_i \in A$ (where $1 \leq i \leq s$), there exists the set $\{a_i\} \cup A'_i$ in G' with α_i vertices. Similarly, for each $b_j \in B$ (where $1 \leq j \leq t$), there exist β_j copies in G' (i.e., $\{b_j\} \cup B'_j$). Therefore, the capacities of the vertices of A and B are satisfied in M_2. Note that, by Observation 3 and because M_2 consists of vertex-disjoint edges of G', at most one of the following can hold: $(a_i, b'_{jk}) \in M_2$, $(a'_{ik'}, b_j) \in M_2$, or $(a_i, b_j) \in M_2$ (we do not count the zero-weighted dummy edges $(a'_{ik'}, b'_{jk})$). For each such case, we add the edge (a_i, b_j) to L'. It is easy to see that $W(L) \geq W(L') = W(M_2)$.

Theorem 2. *Let $G = (A \cup B, E)$ be a non-positive real weighted bipartite graph with $|A| + |B| = n$, a maximum weight LCMM in G can be computed in $O(n^3)$ time.*

Algorithm 2 ModifiedHungarianAlg($G' = (X \cup Y, E')$,A,M,l)

while $\{u \in A | u \; is \; free\} \neq \emptyset$ **do**
  Select a free vertex $x_i \in A$, and let $S = \{x_i\}$, $T = \emptyset$
  $B'' = \emptyset$
  for $j \leftarrow 1$ to $|B|$ **do**
    Select a free vertex $v \in B'_j$
    $B'' = B'' \cup \{v\}$
  end for
  Let C be the set of the matched vertices of B'
  $Y' = B \cup C \cup B''$
  for all $y_j \in Y'$ **do**
    $slack[j] = l(x_i) + l(y_j) - W(x_i, y_j)$
  end for
  repeat
    if $N_l(S) = T$ **then** ▷ $N_l(u) = \{v \in Y' | (u, v) \in E_l\}$
      $\alpha_l = \min_{y_j \in Y' \setminus T} slack[j]$
      $Update(l)$ ▷ Update the labels according to Lemma 1
      for all $y_j \in Y' \setminus T$ **do**
        $slack[j] = slack[j] - \alpha_l$
      end for
    end if
    Select $u \in N_l(S) \setminus T$
    if u is not free **then** ▷ (u is matched to a vertex z, extend the alternating tree)
      $S = S \cup \{z\}, T = T \cup \{u\}$.
      for all $y_j \in Y'$ **do**
        $slack[j] = \min(l(z) + l(y_j) - W(z, y_j), slack[j])$
      end for
    end if
  until u is free
  Add (x_i, u) to M or $augment(M)$ so that all new adding edges to M are in E_l
end while
return M and l

Conflict of Interest

The authors declare that they have no conflict of interest.

Data Availability

Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

Algorithm 3 Maxweight LCMM Algorithm($G = (A \cup B, E)$)

Construct the bipartite graph $G' = (X \cup Y, E')$ from G with $X = A \cup A'$ and $Y = B \cup B'$
Initial ▷ Find an initial feasible labeling l and a matching M in E_l
    $p = |X|$, $q = |Y|$
    Let $l(y_j) = 0$ for all $1 \leq j \leq q$ and $l(x_i) = \max_{j=1}^{q} W(x_i, y_j)$ for all $1 \leq i \leq p$
    $M = \emptyset$
(M_1, l_1) =ModifiedHungarianAlg(G',A,M,l)
$Btemp = B$
for all $a_i \in A$ **do**
  if $\exists b'_{jk} \in B'_j$ s.t. $(a_i, b'_{jk}) \in M_1$ **then**
    $Btemp = Btemp - \{b_j\}$
  end if
end for
(M_2, l_2) =ModifiedHungarianAlg(G',$Btemp$,M_1,l_1)
return M_2

References

[1] L. Chen, R. Kyng, Y. Liu, R. Peng, M.P. Gutenberg, and S. Sachdeva, *Maximum flow and minimum-cost flow in almost-linear time*, J. ACM **72** (2025), no. 3, 1–103.
https://doi.org/10.1145/3728631.

[2] T.B. Eiter and H. Mannila, *Distance measures for point sets and their computation*, Acta Informatica **34** (1997), 109–133.
https://doi.org/10.1007/s002360050075.

[3] M.L. Fredman and R.E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM **34** (1987), no. 3, 596–615.
https://doi.org/10.1145/28869.28874.

[4] H.N. Gabow, *An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems*, Proceedings of the fifteenth annual ACM symposium on Theory of computing, 1983, pp. 448–456.

[5] H.N. Gabow and R.E. Tarjan, *Faster scaling algorithms for network problems*, SIAM J. Comput. **18** (1989), no. 5, 1013–1036.
https://doi.org/10.1137/0218069.

[6] C.C. Huang and T. Kavitha, *New algorithms for maximum weight matching and a decomposition theorem*, Math. Oper. Res. **42** (2017), no. 2, 411–426.
https://doi.org/10.1287/moor.2016.0806.

[7] M. Imanparast and S.N. Hashemi, *A linear time randomized approximation algorithm for Euclidean matching*, J. Supercomput. **75** (2019), 2648–2664.
https://doi.org/10.1007/s11227-018-2673-2.

[8] H.W. Kuhn, *The Hungarian method for the assignment problem*, Nav. Res. Logist. Q. **2** (1955), no. 1-2, 83–97.

[9] C. Lo, S. Kim, S. Zakov, and V. Bafna, *Evaluating genome architecture of a complex region via generalized bipartite matching*, BMC Bioinformatics **14** (2013), #S13.
https://doi.org/10.1186/1471-2105-14-S5-S13.

[10] P.A.C. Lopes, S.S. Yadav, A. Ilic, and S.K. Patra, *Fast block distributed CUDA implementation of the Hungarian algorithm*, J. Parallel Distrib. Comput. **130** (2019), 50–62.
https://doi.org/10.1016/j.jpdc.2019.03.014.

[11] J Munkres, *Algorithms for the assignment and transportation problems*, J. Soc. Indust. Appl. Math **5** (1957), no. 1, 32–38.

[12] J.B. Orlin and R.K. Ahuja, *New scaling algorithms for the assignment and minimum mean cycle problems*, Math. Program. **56** (1992), 41–56.
https://doi.org/10.1007/BF01586040.

[13] D. Rubert, E. Hoshino, M. Braga, J. Stoye, and F. Martinez, *Computing the family-free DCJ similarity*, BMC Bioinformatics **19** (2018), 152.
https://doi.org/10.1186/s12859-018-2130-5.

[14] J. Song, W. Peng, and F. Wang, *A random walk-based method to identify driver genes by integrating the subcellular localization and variation frequency into bipartite graph*, BMC Bioinformatics **20** (2019), 238.
https://doi.org/10.1186/s12859-019-2847-9.

[15] Q. Zhang, H. Wang, Z. Feng, and Z. Han, *Many-to-many matching-theory-based dynamic bandwidth allocation for UAVs*, IEEE Internet Things J. **8** (2021), no. 12, 9995–10009.
https://doi.org/10.1109/JIOT.2021.3049608.