Solving a Special Type of Optimal Transport Problem by a Modified Hungarian Algorithm

Yiling Xie
YXIE350@GATECH.EDU

Yiling Luo
YLUO373@GATECH.EDU

Xiaoming Huo
HUO@GATECH.EDU

Georgia Institute of Technology, United States

Abstract

We observe that computing empirical Wasserstein distance in the independence test is an optimal transport (OT) problem with a special structure. This observation inspires us to study a special type of OT problem and propose a modified Hungarian algorithm to solve it exactly. For an OT problem involving two marginals with \( m \) and \( n \) atoms \((m \geq n)\), respectively, the computational complexity of the proposed algorithm is \( O(m^2n) \). Computing the empirical Wasserstein distance in the independence test requires solving this special type of OT problem, where we have \( m = n^2 \). The associated computational complexity of our algorithm is \( O(n^5) \), while the order of applying the classic Hungarian algorithm is \( O(n^6) \). Numerical experiments validate our theoretical results. Broader applications of the proposed algorithm are discussed at the end.

1. Introduction

One appealing application of optimal transport (OT) and Wasserstein distance [16, 18] is the independence test. The Wasserstein distance between two distributions \( \mu_1, \mu_2 \in Z \) is defined as:

\[
W(\mu_1, \mu_2) := \inf \left\{ \int_{Z^2} d(z, z')d\gamma(z, z') : \gamma(\mu_1, \mu_2) \text{ is a distribution with marginals } \mu_1 \text{ and } \mu_2 \right\},
\]

where \((Z, d)\) is a metric space (w.l.o.g., we consider 1-Wasserstein distance in this paper). Wasserstein distance is a metric on probability measures [18]. To test the independence between variables \( Y \sim \nu_1 \) and \( Z \sim \nu_2 \), people utilize the Wasserstein distance between the joint distribution of \( Y, Z \) and the product distribution of \( Y, Z \), i.e., \( W_p(\gamma(\nu_1, \nu_2), \nu_1 \otimes \nu_2) \). While the statistical properties of this approach have been intensely investigated [11, 14, 19], there is no existing literature focusing on the computational aspect. In this paper, we will consider ‘How to calculate the empirical Wasserstein distance in the independence test?’

In practice, given \( n \) i.i.d. samples \( \{(y_1, z_1), ..., (y_n, z_n)\} \) generated from \( (Y, Z) \), we build the statistic—\( W_p(\gamma(\hat{\nu}_1, \hat{\nu}_2), \hat{\nu}_1 \otimes \hat{\nu}_2) \), where \( \hat{\nu} \) denotes the corresponding empirical distribution—to test the independence. We will show later that computing this statistic is equivalent to solving the
We further discuss other applications of the proposed algorithm, seeing Section 5.

Related work: Notice that our modified Hungarian algorithm is an exact OT solver. There are also a bunch of approximation algorithms [4, 5, 8, 20]. Note that some scenarios may require the precise solution of OT problems, and paper [6] demonstrates the favorable numerical performance of the exact solutions over the approximate solutions. Therefore, developing efficient exact algorithms is meaningful.

Our contributions: Firstly, we find out that computing the Wasserstein distance in the independence test (1) belongs to a special type of OT problem (2). Secondly, we propose a modified Hungarian algorithm to solve problem (2). The algorithm modifies the Hungarian algorithm to deal with the scenario where two marginals are of different dimensions and atoms in one of the marginals have multiple assignments. Thirdly, we adopt our algorithm to the independence test problem (1). We further discuss other applications of the proposed algorithm, seeing Section 5.

2. Modified Hungarian algorithm

In this section, we propose a modified Hungarian algorithm to solve the problem (2).
Figure 1: Examples when $n = 3$, $m = 9$, $m_1 = 2$, $m_2 = 3$, $m_3 = 4$. The solid line means that the edge belongs to the pseudo-matching. The dashed line indicates that the edge belongs to the equality graph but does not belong to the pseudo-matching.

We first review the basics of the Hungarian algorithm [12, 17]. Suppose that a bipartite graph $G = (V, E)$, where $V = V_1 \cup V_2$, $E \subseteq V_1 \times V_2$, has weight $w(\cdot)$ for each edge $e \in E$. $M \subseteq E$ is a matching if every node of $G$ coincides with at most one edge of $M$. Kuhn-Munkres theorem [12] illustrates that solving the assignment problem is equivalent to finding a perfect matching on the equality graph associated with some feasible labeling. (The feasible labeling is defined as a function $l : V \rightarrow \mathbb{R}$ satisfying $l(v_1) + l(v_2) \geq w(v_1, v_2)$, $\forall v_1 \in V_1, v_2 \in V_2$, and the equality graph w.r.t. labeling $l$ is $G' = (V, E_l)$, where $E_l = \{(v_1, v_2) : l(v_1) + l(v_2) = w(v_1, v_2)\}$.) Hence, Hungarian algorithm solves the assignment problem by looking for a perfect matching on some equality graph.

In problem (2), $X' \subseteq U'$ has one entry of 1 in each row, multiple entries of 1 in each column and 0’s elsewhere. Since a permutation matrix corresponds to a (perfect) matching in the bipartite graph, we define ‘pseudo-matching’ in the bipartite graph to describe $X'$: In the bipartite graph $G$, where $|V_1| = m$, $|V_2| = n$. $PM \subseteq E$ is a pseudo-matching if every node of $V_1$ coincides with at most one edge of $PM$ and $j$th node of $V_2$ coincides with at most $m_j$ edges of $PM$, where $\sum_{j=1}^n m_j = m$. Further, if every node of $V_1$ coincides with one edge of $PM$ and $j$th node of $V_2$ coincides with $m_j$ edges of $PM$, $PM$ is called a perfect pseudo-matching. See examples in Figure 1.

We develop a modified Kuhn-Munkres theorem (the theorem and its proof can be found in the Appendix). It demonstrates that we only need to find a perfect pseudo-matching on some equality graph to solve the problem (2). In this way, we design a modified Hungarian algorithm (Algorithm 1). It improves either the feasible labeling (adding edges to the associated equality graph) or the pseudo-matching until the pseudo-matching is perfect on some equality graph w.r.t. some feasible labeling. The algorithm improves the pseudo-matching by generating pseudo-augmenting paths and then exchanging the edge status along the paths. This process is called the pseudo-augmenting process. Also, we force the pseudo-augmenting paths emanating from $V_2$, which have a lower order of nodes. See the relevant definitions in Definition 1 and examples in Figure 1.

**Definition 1 (Free, matched, pseudo-matched, pseudo-alternating path, pseudo-augmenting path)**
Let $PM$ be a pseudo-matching of $G = (V, E)$. • If the node $v$ is in $V_1$, it is pseudo-matched if it is an endpoint of some edge in $PM$; if the node $v$ is the $j$th node in $V_2$, it is pseudo-matched if it is an endpoint of $m_j$ edges in $PM$. Otherwise, the node is free. • If the node $v \in V$, we say it is matched if it is an endpoint of some edge in $PM$. • A path is pseudo-alternating if its edge alternates between $PM$ and $E - PM$. A pseudo-alternating path is pseudo-augmenting if both its endpoints are free.

We now analyze the computational complexity of Algorithm 1. Similar to the Hungarian algorithm [17], we keep track of $slack_{v_1} = \min_{v_2 \in S} \{l(v_1) + l(v_2) - w(v_1, v_2)\}$, $\forall v_1 \not\in T$. The number of edges of the pseudo-matching increases by 1 after one loop, so we need $O(m)$ loops to
**Algorithm 1: Modified Hungarian Algorithm**

Generate an initial feasible labeling \( l: \forall v_2 \in V_2, l(v_2) = 0; \)
\( \forall v_1 \in V_1, l(v_1) = \max_{v_2 \in V_2} \{ w(v_1, v_2) \} \) and initialize a pseudo-matching \( M \) in \( E_l; \)

1. **if** \( M \) is a perfect pseudo-matching **then**
   1. **Stop**
   **else**
   1. Pick up a free node \( v_{\text{free}} \in V_2 \). Set \( S = \{ v_{\text{free}} \}, T = \emptyset; \)
   2. **for** \( v_1 \in V_1 \) is matched to \( v_{\text{free}} \) **do**
   3. \( T = T \cup v_1 \)
   **end**
   **end**

2. **if** \( N_l(S) - T = \emptyset \) **then**
   1. update labeling such that forcing \( N_l(S) - T \neq \emptyset : \alpha_l = \min_{v_1 \in T, v_2 \in S} \{ l(v_1) + l(v_2) - w(v_1, v_2) \}; l(v) = l(v) - \alpha, v \in S; l(v) = l(v) + \alpha, v \in T; l(v) = l(v), \) otherwise.
   **end**

3. **if** \( N_l(S) - T \neq \emptyset \) **then**
   1. pick \( v_1 \in N_l(S) - T; \)
   2. **if** \( v_1 \) is free **then**
   3. \( v_{\text{free}} \rightarrow v_1 \) is a pseudo-augmenting path. Pseudo-augment the pseudo-matching \( M \). Go to Step 1;
   **end**
   2. **if** \( v_1 \) is pseudo-matched to \( z \) **then**
   3. extend the pseudo-matching tree: \( S = S \cup \{ z \}, T = T \cup \{ v_1 \}; \)
   4. **for** \( v_1 \in V_1 \) is matched to \( z \) **do**
   5. \( T = T \cup v_1; \)
   **end**
   **end**
   **end**
   Go to Step 2.

form a perfect pseudo-matching. There are two subroutines in each loop: (1): updating feasible labeling (Step 2); (2): improving pseudo-matching (Step 3). In procedure (1): Since there are \( n \) nodes in \( V_2 \), improving the labeling occurs \( O(n) \) times to build a pseudo-alternating tree. Computing \( \alpha_l \), updating the slacks and calculating the labeling cost \( O(m) \) each time. In procedure (2): When a new node has been added to \( S \), it costs \( O(m) \) to update slacks, and \( O(n) \) nodes could be added. On the other hand, when a node has been added to \( T \), we just remove the corresponding slack \( v_1 \).

From the above we conclude that each loop costs \( O(mn) \), so the total computational complexity of Algorithm 1 to solve problem (2) is \( O(m^3n) \). Note that the adoption of the Hungarian algorithm has an order of \( O(m^3) \). Hence, the proposed modified Hungarian algorithm will outperform, especially when \( m \gg n \).
3. Independence test using the Wasserstein distance

In this section, we discuss the special case of the OT problem (2)—independence test using the Wasserstein distance, and apply the modified Hungarian algorithm to solve it.

Suppose that we have \( n \) i.i.d. samples \( \{(y_1, z_1), \ldots, (y_n, z_n)\} \), where \( (y_i, z_j) \sim (Y, Z) \). If we utilize the statistic \( W_p(\gamma(\tilde{P}_1, \tilde{P}_2), \tilde{P}_1 \otimes \tilde{P}_2) \) to test the independence between \( Y \) and \( Z \), we need to solve the following optimization problem:

\[
\min_{X \in \Pi} \sum_{i,j,k,l=1}^{n} d((y_i, z_j), (y_k, z_l)) X_{ij;kl},
\]

where

\[
\Pi = \left\{ X_{ij;kl} \geq 0 \left| \sum_{k,l=1}^{n} X_{ij;kl} = \frac{1}{n^2}, \sum_{i,j=1}^{n} X_{ij;kl} = \begin{cases} \frac{1}{n} & k = l, \\ 0 & k \neq l, \end{cases}, \forall i, j, k, l = 1, \ldots, n. \right\}.
\]

It is worth noting that \( X_{ij;kl} = 0, \forall k \neq l \). If we further let \( X_{ij;k}^p := \sum_{l=1}^{n} X_{ij;kl} \), problem (3) could be reformulated as problem (1) and is equivalent to the following optimization problem: (The proof and relevant discussions are relegated to the Appendix.)

\[
\min_{X' \in \Pi'} \sum_{i,j,k=1}^{n} \frac{d((y_i, z_j), (y_k, z_k))}{n^2} X'_{ij;k}, \quad \Pi' = \left\{ X'_{ij;k} \in \{0, 1\} \left| \sum_{k=1}^{n} X'_{ij;k} = 1, \sum_{i,j=1}^{n} X'_{ij;k} = n \right\}.
\]

Problem (4) belongs to the special type of OT problem where \( m_j = n, m = n^2 \). Adopting the Hungarian algorithm to problem (4) costs \( O(n^6) \) while adopting the proposed Hungarian algorithm directly costs \( O(n^5) \).

4. Numerical experiments

In this section, we carry out experiments on the independence test problem (1) on CIFAR10 \(^3\) \([10]\) and Wisconsin breast cancer \(^2\) \([7]\) datasets. The numerical results validate the computational complexity of the modified Hungarian algorithm and the favorability of applying the proposed algorithm over previous methods.

\(1\). https://www.cs.toronto.edu/~kriz/cifar.html
\(2\). https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
applying the Hungarian algorithm. The performance is evaluated by the number of numerical operations here. We relegate the results of the running time to the Appendix.

**Experiments details:** We create one dependent case and one independent case with different sample sizes: **CIFAR10:** Each image is a 3072-dim vector. Let $X \in \mathbb{R}^{3072}$ be generated uniformly from the first 5 classes; $Y \in \mathbb{R}^{3072}$ be generated uniformly from other classes. We calculate the Wasserstein distance (1) independent case: between $X$ and $Y_1$ (where $Y_1$ is the first 1536 coordinates of $Y$); (2) dependent case: between $X$ and $Z$ (where $Z = X_2/2 + Y_1/2$, $X_2$ is the last 1536 coordinates of $X$, $Y_1$ is the first 1536 coordinates of $Y$). **Breast cancer data:** Each instance is a 30-dim vector. Let $X \in \mathbb{R}^{30}$ be generated uniformly from the benign class, and $Y \in \mathbb{R}^{30}$ be generated uniformly from the malignant class. We calculate the Wasserstein distance (1) independent case: between $X_1$ and $Y_2$ (where $X_1$ is the first 5 coordinates of $X$, $Y_2$ the last 25 coordinates of $Y$); (2) dependent case: between $X$ and $Z$ (where $Z = X_1 \ast Y_1$, $X_1$ is the first 5 coordinates of $X$, $Y_1$ is the first 5 coordinates of $Y$, $\ast$ means the coordinate-wise product). We run the algorithms on each case 10 times and plot the worst, best and average number of numerical operations.

**Experiments results:** The results are presented in Figure 2, 3. The figures illustrate that the proposed algorithm gains a factor $n$ in computational complexity when solving problem (1). To be more specific, notice that the slope of $\ln(\text{number of numerical operations})$ over $\ln(\text{sample size})$ indicates the order of the associated algorithm, and the slope of our algorithm is around 5 while the slope of the Hungarian algorithm is around 6. This observation implies that the order of applying our algorithm is $O(n^5)$ while the order of applying the Hungarian algorithm is $O(n^6)$. It is consistent with our theoretical results.

5. Discussion

We propose a modified Hungarian algorithm to solve a special type of OT problem (2). The proposed algorithm could help solve one-to-many assignment [21] and many-to-many assignment [22] problems. In this way, our algorithm could be applied to related practical problems in service assignment problems [13], sensor networks [2], access control [1], etc. In addition, we may apply our algorithm to some modern machine learning models, including clustering [9] and representation learning [15].
References


Appendix A. Modified Kuhn-Munkres theorem

**Theorem 2 (Modified Kuhn-Munkres theorem)** If \( l \) is a feasible labeling on the weighted bipartite graph \( G = (V, E) \) and \( PM \subset E_l \) is a perfect pseudo-matching on the corresponding equality graph \( G' = (V, E_l) \), then \( PM \) is a maximum weighted pseudo-matching.

**Proof** Denote the edge \( e \in E \) by \( e = (e_{v_1}, e_{v_2}) \). Let \( PM' \) be any perfect pseudo-matching in \( G \) (not necessarily in the equality graph \( E_l \)). And \( v_i', i = 1, ..., m; \ v_j', j = 1, ..., n \) are nodes from \( V_1 \) and \( V_2 \), respectively. Since \( v_i' \in V_1 \) is covered exactly once by \( PM' \) and \( v_j' \in V_2 \) is covered exactly \( m_j \) times by \( PM' \), we have

\[
w(\text{PM}') = \sum_{e \in \text{PM}'} w(e) \leq \sum_{e \in \text{PM}'} (l(e_{v_1}) + l(e_{v_2})) = \sum_{i=1}^{m} l(v_i') + \sum_{j=1}^{n} m_j l(v_j'),
\]

where the first inequality comes from the definition of the feasible labeling.

Thus, \( \sum_{i=1}^{m} l(v_i') + \sum_{j=1}^{n} m_j l(v_j') \) is upper bound of the weight of any perfect pseudo-matching. Then let \( PM \) be a perfect pseudo-matching in the equality graph \( E_l \), we have

\[
w(\text{PM}) = \sum_{e \in \text{PM}} w(e) = \sum_{i=1}^{m} l(v_i') + \sum_{j=1}^{n} m_j l(v_j').
\]

Hence \( w(\text{PM}') \leq w(\text{PM}) \) and \( PM \) is the maximum weighted pseudo-matching. \( \blacksquare \)
A.1. Proof and relevant discussion for Section 3

Instead of direct proving (3) \(\iff\) (4), we prove the more general scenario first.

**Proposition 3** The optimization problem (2) is equivalent to the optimization problem (5):

\[
\min_{X \in \mathcal{U}} \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} C_{ij}, \quad \mathcal{U} = \left\{ X_{ij} \geq 0 \left| \sum_{j=1}^{n} X_{ij} = \frac{1}{m}, \sum_{i=1}^{m} X_{ij} = \frac{m_j}{m}, \sum_{j=1}^{n} m_j = m \right. \right\}. \tag{5}
\]

**Proof** First consider the following two optimization problems (6)-(7):

\[
\min_{X^{\dagger} \in \mathcal{U}^{\dagger}} \sum_{i=1}^{m} \sum_{j=1}^{m} \frac{1}{m} X^{\dagger}_{ij} C^{\dagger}_{ij}, \quad \mathcal{U}^{\dagger} = \left\{ X^{\dagger}_{ij} = \{0, 1\} \left| \sum_{j=1}^{m} X^{\dagger}_{ij} = 1, \sum_{i=1}^{m} X^{\dagger}_{ij} = 1 \right. \right\}, \tag{6}
\]

where \(C^{\dagger}\) is an \(m \times m\) matrix generated by duplicating the \(j\)th column of \(C\) \(m_j\) times:

\[
C^{\dagger}_{it} = C_{ij}, \quad 1 \leq t \leq m_j, j = 1; m_1 + \ldots + m_{j-1} + 1 \leq t \leq m_1 + \ldots + m_j, j \geq 2.
\]

\[
\min_{X^{\dagger} \in \mathcal{U}^{\dagger}} \sum_{i=1}^{m} \sum_{j=1}^{m} X^{\dagger}_{it} C^{\dagger}_{it}, \quad \mathcal{U}^{\dagger} = \left\{ X^{\dagger}_{it} \geq 0 \left| \sum_{j=1}^{m} X^{\dagger}_{it} = \frac{1}{m}, \sum_{i=1}^{m} X^{\dagger}_{it} = \frac{1}{m} \right. \right\}, \tag{7}
\]

where \(C^{\dagger}\) is an \(m \times m\) matrix generated by duplicating the \(j\)th column of \(C\) \(m_j\) times:

\[
C^{\dagger}_{it} = C_{ij}, \quad 1 \leq t \leq m_j, j = 1; m_1 + \ldots + m_{j-1} + 1 \leq t \leq m_1 + \ldots + m_j, j \geq 2.
\]

Then we denote the objective functions of the problems (2), (5), (6) and (7) by \(f(X), f^{\dagger}(X^{\dagger}), f'(X'),\) and \(f^1(X^1),\) respectively.

Firstly, we prove (5) \(\iff\) (7).

On the one hand, for any \(X^1 \in \mathcal{U}^1,\) if we let

\[
X_{ij} = \sum_{t=1}^{m_1} X^1_{it}, \quad j = 1,
\]

\[
X_{ij} = \sum_{t=m_1+\ldots+m_{j-1}+1}^{m_1+\ldots+m_j} X^1_{it}, \quad j \geq 2,
\]

then we have

\[
X_{ij} \geq 0,
\]

\[
\sum_{j=1}^{n} X_{ij} = \sum_{t=1}^{m_1} X^1_{it} + \sum_{t=m_1+\ldots+m_{j-1}+1}^{m_1+\ldots+m_j} X^1_{it} = \sum_{t=1}^{m} X^1_{it} = \frac{1}{m},
\]

\[
\sum_{i=1}^{m} X_{ij} = \sum_{t=1}^{m_1} \sum_{t=1}^{m_1} X^1_{it} = \frac{m_1}{m}, \quad j = 1,
\]

\[
\sum_{i=1}^{m} \sum_{t=m_1+\ldots+m_{j-1}+1}^{m_1+\ldots+m_j} X^1_{it} = \frac{m_j}{m}, \quad j \geq 2.
\]
Thus, $X \in \mathcal{U}$. For the objective functions, we have the following:

$$f(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} C_{ij} = \sum_{i=1}^{m} \left( \sum_{t=m_{1} + \ldots + m_{j-1} + 1}^{m_{1} + \ldots + m_{j}} X_{it}^{1} C_{it}^{1} \right) = \sum_{i=1}^{m} \sum_{t=1}^{m} X_{it}^{1} C_{it}^{1} = f^{1}(X^{1}).$$

On the other hand, for any $X \in \mathcal{U}$, if we let

$$X_{it}^{1} = \frac{X_{ij}}{m_{j}}, \quad 1 \leq t \leq m_{j}, j = 1; m_{1} + \ldots + m_{j-1} + 1 \leq t \leq m_{1} + \ldots + m_{j}, j \geq 2,$$

then we have

$$X_{it}^{1} \geq 0,$$

$$\sum_{t=1}^{m} X_{it}^{1} = \sum_{j=1}^{n} \frac{X_{ij}}{m_{j}} m_{j} = \sum_{j=1}^{n} X_{ij} = \frac{1}{m},$$

$$\sum_{i=1}^{n} X_{it}^{1} = \sum_{i=1}^{n} \frac{X_{ij}}{m_{j}} m_{j} = \frac{1}{m_{j}} \sum_{i=1}^{n} X_{ij} = \frac{1}{m}.$$
Proof Applying Proposition 3 to the optimization problem (1), the optimization problem (1) is equivalent to problem (8):

\[
\min_{X' \in \Pi'} \sum_{h=1}^{n} \sum_{k=1}^{n} \frac{1}{n^2} C_{h,k}^{d} X'_{h,k}, \quad \Pi' = \left\{ X'_{h,k} = \{0, 1\} \mid \sum_{k=1}^{n} X'_{h,k} = 1, \sum_{h=1}^{n^2} X'_{h,k} = n \right\}. \tag{8}
\]

If we express the index \( h \) by two indices \( i, j \) and express \( t \) by \( k, l \), one could check that (8) \( \iff \) (4). In conclusion, (1) \( \iff \) (4).

Appendix B. Experimental results w.r.t. running time

We run the Hungarian algorithm code from package ‘scipy’ in Python and implement the modified Hungarian algorithm. We record the running time. Figure 4, 5 are experiment results w.r.t. the running time. When the sample size is small, our algorithm may be slower than the Hungarian algorithm. However, as the sample size increases, our algorithm becomes more efficient than the Hungarian algorithm. In theory, our proposed algorithm will compare more favorably if both algorithms are implemented in a low-level language, such as C++.

Appendix C. Resources

All experiments in this paper are implemented using Python 3.8 on Macbook Pro 2019 with 16 GB memory.