Two Algorithms for Maximum and Minimum Weighted Bipartite Matching

Advisor: Prof. Yuh-Dauh Lyuu

Hung-Pin Shih

Department of Computer Science and Information Engineering
National Taiwan University

Abstract

This thesis applies two algorithms to the maximum and minimum weighted bipartite matching problems. In such matching problems, the maximization and minimization problems are essentially same in that one can be transformed into the other by replacing the weight on each edge with an inverse of the weight. Depending on the algorithms we used, we will choose the maximization or minimization problems for illustrations. We apply the ant colony optimization (ACO) algorithm on a minimum weighted bipartite matching problem by transforming the problem to a traveling salesman problem (TSP). It may seem that makes the problem more complicated, but in reality it does not. We call this algorithm "ant-matching," which can solve any weighted bipartite matching problems with or without a perfect matching. Besides, we also apply the Metropolis algorithm to solve the maximum weighted bipartite matching problem. To analyze the performance on these two algorithms, we compare the algorithms with the exact Hungarian algorithm, a well-known combinatorial optimization method for solving the weighted bipartite matching problem.

Keywords: Metropolis algorithm, Ant colony optimization, Maximum weighted bipartite matching, Minimum weighted bipartite matching

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The maximum/minimum weighted bipartite matching problem is a combinatorial optimization problem, and there exist many optimization algorithms to solve the matching problem. But here we try some heuristic or sampling methods and expect them to generate reasonable sub-optimal solutions within reasonable time bounds. Specifically we use the ant colony optimization (ACO) and the Metropolis algorithm. In the ACO algorithm, we transform this matching problem to a traveling salesman problem (TSP) and use the concepts of ACO algorithm as a basis for a new method to solve the matching problem. We call the new method "ant-matching." In the Metropolis algorithm, we randomly select a matching edge in each iteration to generate an approximate solution. Then we compare the performances of these two algorithms with the exact Hungarian algorithm [2], which is a optimization algorithm used to solve the weighted matching problems.

The structure of this thesis is organized as follows: In Chapter 1 and Chapter 2, we define the problem of maximum/minimum weighted bipartite matching and introduce the background of the two above-mentioned algorithms. In Chapter 3, we transform the weighted bipartite matching problem to a traveling salesman problem (TSP) and apply the concepts of ant colony optimization (ACO) algorithm as a basis for a new matching algorithm called "ant-matching." In the latter part of Chapter 3, we apply the Metropolis algorithm to solve the maximum weighted bipartite matching problem. Finally, Chapter 4 compares these algorithms with other methods in solving the weighted matching problems, and we conclude in Chapter 5.

Chapter 2

Preliminaries

2.1 Maximum/Minimum Weighted Bipartite Matching

2.1.1 Weighted Bipartite Graph

A bipartite graph $G = (U, V, E)$ is a graph whose vertices can be divided into two disjoint sets U and V such that each edge $(u_i, v_j) \in E$ connects a vertex $u_i \in U$ and one $v_j \in V$. If each edge in graph G has an associated weight w_{ij}, the graph G is called a weighted bipartite graph.

2.1.2 Maximum/Minimum Weighted Bipartite Matching

In a bipartite graph $G = (U, V, E)$, a matching M of graph G is a subset of E such that no two edges in M share a common vertex. If the graph G is a weighted bipartite graph, the maximum/minimum weighted bipartite matching is a matching whose sum of the weights of the edges is maximum/minimum. The maximum/minimum weighted bipartite matching can be formulated as follows:

$$\max / \min \sum_{(u_i,v_j)\in E} w_{ij}x_{ij}$$

$$\sum_{i=1}^{|U|} x_{ij} = 1, \forall j = 1, ..., |V|$$

$$\sum_{j=1}^{|V|} x_{ij} = 1, \forall i = 1, ..., |U|$$

$$x_{ij} \in \{0, 1\}$$

where x_{ij} is 1 denotes edge (u_i, v_j) is an edge in the maximum/minimum weighted bipartite matching.

2.2 Ant Colony Optimization

In the real world, ants are social insects and communicate with some interesting behavior. Each ant lays down pheromone on the path. When ants find foods, they will go back to its nest. If some other ants encounter the path, they will follow it based on the density of pheromone deposited. The pheromone on the path that ants found initially will be reinforcing. Eventually, ants will find the foods.

Over time, the pheromone on the path evaporates, thus reducing its attracting strength. The longer path an ant travels down and back, the more time the pheromone has to evaporate. On a shorter path, an ant travels down the path and back faster, and the pheromone density remains high as the pheromone is laid on as fast as it can evaporate. Assume the ant colony converges to a path. If a shorter path is found by some ants, because the path is shorter than the original path that the ant colony converged to, the ants travel down the path and back faster. The density of pheromone on the shorter path will increase gradually and the density on the original path will decrease. After a period of time, the ant colony will converge to the shorter path. It will prevent the ant colony from converging to a local optimal solution.

The ant colony optimization (ACO) algorithm comes from observing the behavior of the ants. It simulates the ants and the pheromone evaporation on the paths. In each iteration, the ants select the paths according to the density of pheromone deposited. If the path has higher density, it will be selected with higher probability. After finding a path, the ants lay down pheromone on the path. In the ACO algorithm, each path represents a solution. Besides, the pheromone evaporates gradually. The ACO algorithm is illustrated in Algorithm 1.

Algorithm 1 ACO Algorithm for TSP

```
while termination condition not met do
    generate solutions
    pheromone update
end while
```

The ACO algorithm has been used in solving the traveling salesman problem (TSP) to find a nearly optimal solution [4]. In the TSP problem, there are n cities. Each city has to be visited exactly once, and the tour ends at the starting city. The problem is to find a shortest tour to visit these n cities. Let d_{ij} be the distance between the city i and the city j and τ_{ij} be the pheromone on the edge connects i and j.

Each of the m ants decides independently on the city to be visited next. They base their decision on the density of pheromone τ_{ij} and a heuristic function η_{ij}. The probability of choosing the next city j from current city i at the tth iteration is

$$p_{ij}^k = \frac{[\tau_{ij}(t)]^\alpha (\eta_{ij})^\beta}{\sum_{l \in N_l^k} [\tau_{il}(t)]^\alpha (\eta_{ij})^\beta} \quad , \forall j \in N_i^k \tag{2.1}$$

where $\eta_{ij} = \frac{1}{d_{ij}}$ is a commonly used heuristic function, α and β are two parameters to determine the relative influences of the pheromone and the distance, and N_i^k is a list of cities that the kth ant has yet to visit.

After all ants have constructed their complete tours, we update the pheromone on each edge thus:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{2.2}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate, $\Delta\tau_{ij}^k(t)$ is the pheromone that the kth ant deposited at the tth iteration, which is defined as follows:

$$\Delta\tau_{ij}^k(t) = \begin{cases} 1/L^k(t) & \text{if}(i,j) \in T^k(t) \\ 0 & \text{if}(i,j) \notin T^k(t) \end{cases} \tag{2.3}$$

where $L^k(t)$ is the length of the kth ant's tour at the tth iteration, $T^k(t)$ is the kth ant's tour at the tth iteration. The shorter the tour the ants found, the more pheromone is deposited. After a number of iterations, the ant colony will converge to a nearly shortest tour. Finally, select any starting city and follow the edge with the highest density of pheromone to visit as the next city. After completing the tour, the tour is a solution of the TSP problem.

2.3 Metropolis Algorithm

The Metropolis algorithm is an algorithm to generate a sequence of samples from a probability distribution that is hard to sample directly. It uses the Markov chain Monte Carlo simulation to approximate the distribution [1].

The Metropolis algorithm can generate samples from any unknown probability distribution $P(x)$ and requires only a function proportional to the density of $P(x)$ that can be calculated at x. The algorithm generates a Markov chain in which each state x^{t+1} depends only on the previous state x^t. Let $Q(x^t, x')$ denotes the candidate-generating density, where $\int Q(x^t, x')dx' = 1$. The algorithm depends on the current state x^t to generate a new sample x' from $Q(x^t, \cdot)$[1]. The sample x' is accepted as the next state x^{t+1} if $\alpha \in U(0, 1)$ satisfies Eq. (2.4):

$$\alpha < \frac{P(x')Q(x', x^t)}{P(x^t)Q(x^t, x')} \tag{2.4}$$

If α satisfies Eq. (2.4), $x^{t+1} = x'$. Otherwise, the current state x^t is retained, i.e., $x^{t+1} = x^t$. The algorithm is illustrated in Algorithm 2.

Algorithm 2 Metropolis Algorithm

for $t = 1, 2, ..., N$ **do**
  sample x' from $Q(x^t, \cdot)$ and α from $U(0, 1)$ $\{U(0, 1)$ is the uniform distribution on$(0, 1)\}$
  if $\alpha < \frac{P(x')Q(x', x^t)}{P(x^t)Q(x^t, x')}$ **then**
    $x^{t+1} = x'$
  else
    $x^{t+1} = x^t$
  end if
end for
return $\{x^{(1)}, x^{(2)}, ..., x^{(N)}\}$

[1]Lyuu: but if you do not know what Q is how can you generate x'????
Hung-Pin: We don't know Q at first. But we can observe the possible transitions between the current state and the next state to find a possible Q. The possible Q may be correct or wrong, but it will describe the transitions. This is reasonable, if we don't know the probability distribution $P(x)$, we observe the transitions between the current state and the next state and infer a possible Q. We can use Q to sample the other next states and find the unknown probability distribution $P(x)$.

Chapter 3

Maximum/Minimum Weighted Bipartite Matching Problems

In a weighted bipartite graph $G = (U, V, E)$, we want to find a maximum/minimum weighted bipartite matching (maximum/minimum-weighted BM) whose sum of the weights of the edges is maximum/minimum. To solve this matching problem, we present two methods. For illustrations, we will introduce each method on a different matching problem. Specifically we apply the ACO algorithm to solve the minimum-weighted BM problems and the Metropolis algorithm to solve the maximum-weighted BM problems.

3.1 ACO Algorithm for Minimum-Weighted Bipartite Matching

In a weighted bipartite graph $G = (U, V, E)$, we want to find a matching whose sum of the weights of the edges is minimum. Here we apply the concepts of ACO algorithm as a basis for a new method to solve the matching problem. First, we transform the matching problem to a TSP problem. It may seem that makes the matching problem more complicated, but in reality it does not. Second, we apply the ACO algorithm on the TSP problem and perform some simplifications to be explained later on solving the TSP problem. The result is a new method for solving the minimum-weighted BM problem. In this chapter, the TSP refers to the problem of finding a minimum weighted complete tour on the complete graph with multiple edges between vertices, and the number of edges between vertices will change as the different tour selection.

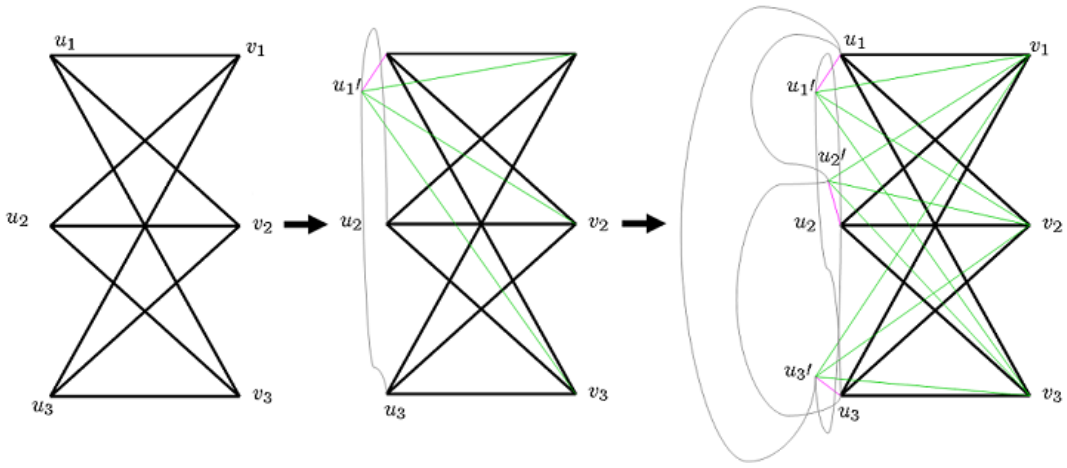

Figure 3.1: Transforming a bipartite matching problem to a TSP: (1) The original bipartite graph G. (2) Add u_1' and its corresponding edges. (3) The final graph G'.

3.1.1 Transforming Minimum-Weighted Bipartite Matching to Traveling Salesman Problem

In a weighted bipartite graph $G = (U, V, E)$, we choose either U or V as the base vertices and transform the minimum-weighted BM problem to a TSP problem. We choose the vertices set U as our base vertices for illustrations. First, for each $u_i \in U$, we create a corresponding virtual vertex $u_i' \in U'$, where U' is the set of vertices correspond to U and $|U'| = |U|$. Then, for each $(u_i, v_j) \in E$ where $u_i \in U, v_j \in V$, we create a corresponding back edge $(u_i', v_j) \in E'$ with an associated weight 0, where E' contains the edges we create. Furthermore, we create another back edge $(u_i, u_i') \in E'$ with a large associated weight $w_{\max}$. Here we define this large associated weight $w_{\max}$ as $20 * \max_{i \in |U|, j \in |V|} w_{ij}$. Finally, for each virtual vertex $u_i' \in U'$, we create the edges $(u_i', u_k) \in E'$ to connect each vertex $u_k \in U, u_k \neq u_i$, with an associated weight $w_{ik} = 0$. Thus, we have a new graph $G' = (U, V, E, U', E')$. The above steps are illustrated in Algorithm 3 and Figure 3.1. In Figure 3.1, each back edge $(u_i', v_j) \in E'$ is in green and $(u_i, u_i') \in E'$ is in red. Besides, the edges (u_i', u_k) connect each vertex u_i' to all the other vertices in U are in grey.

In the new graph $G' = (U, V, E, U', E')$, on each base vertex $u_i \in U$, we define that if we choose the edge $(u_i, v_j) \in E$, then we will follow the edge $(v_j, u_i') \in E'$ back to vertex $u_i' \in U'$. Thus, for each base vertex $u_i \in U$, there are some paths $u_i - v_j - u_i'$ where $(u_i, v_j) \in E$ and $(v_j, u_i') \in E'$, and the path

Algorithm 3 Transform Graph G to G'

Input: Weighted bipartite graph $G = (U, V, E)$
and each edge $(u_i, v_j) \in E$ has a weight w_{ij}.
$E' = \emptyset$
for $u_i \in U$ **do**
  create vertex $u'_i \in U'$
  for $(u_i, v_j) \in E$ **do**
    add back edge (u'_i, v_j) to E' with weight 0
  end for
  add back edge (u_i, u'_i) to E' with weight $w_{\max}$
  for $u_k \in U$ **do**
    add edge (u'_i, u_k) to E' with weight 0
  end for
end for
return graph $G' = \{U, V, E, U', E'\}$

$u_i - u'_i$ where $(u_i, u'_i) \in E'$. So there exist a edge from u_i to its corresponding virtual vertex $u'_i \in U'$. Furthermore, each virtual vertex u'_i is connected to all the other base vertices $u_k \in U$. Thus, if we start from any base vertex $u_i \in U$ on the graph G', we can visit each base vertex in U exactly once and end at u_i (a Hamiltonian cycle). If we want to find a matching M in the bipartite graph G, we can find it with a Hamiltonian cycle in the graph G' as follows.

Let $\texttt{Avail}_i(V) \subseteq V$ be the vertex set adjacent to the vertex $u_i \in U$ such that for each vertex $v_j \in \texttt{Avail}_i(V)$, each edge $(u_l, v_j) \in E$ is not yet an matching edge in the matching M. On each vertex $u_i \in U$, the ant may select[1][2] the edge $(u_i, v_j) \in E$ followed by $(v_j, u'_i) \in E'$ where $v_j \in \texttt{Avail}_i(V)$, or select the edge $(u_i, u'_i) \in E'$. If the ant select the edge $(u_i, v_j) \in E$ followed by $(v_j, u'_i) \in E'$, we add a new matching edge (u_i, v_j) to M. If the ant select the edge $(u_i, u'_i) \in E'$, we do not change M. After visiting all the base vertex $u_i \in U$ exactly once, we will find the matching M.

[1]Lyuu: what do you mean by select? what if both are possible, which one do you select?
Hung-Pin: We choose edge based on the pheromone. It will be explained later. Here we just explain the reduction.

[2]Lyuu: this is strange. if you are only describing a reduction, why bring up "select"? Also if you use "select", it is better to write "the ant may select"?
Hung-Pin: I replace reduction with transformation. Moreover, because there are multi-paths between u_i and u'_i, the different selection has different means in the matching M. I use the "select" to explain the differences and the relations between the matching problem and the TSP problem.

Thus, if we find a tour to visit each base vertex $u_i \in U$ exactly once and then return to the starting base vertex, and the total weight of this tour is minimum, we have a minimum weighted bipartite matching M on the weighted bipartite graph G. Now, we have successfully transformed the minimum-weighted BM problem to the TSP problem.

3.1.2 Using ACO Algorithm to Solve Minimum-Weighted Bipartite Matching

After transforming the weighted bipartite graph $G = (U, V, E)$ to a new graph $G' = (U, V, E, U', E')$, we can use the ACO algorithm to solve the TSP problem on G' and find the solutions of the original matching problem. But here we can perform some simplifications on solving the TSP problem. In the graph G', the algorithm needs to visit all vertices in the set U and corresponding set U' of U to do matching. With the simplification, because we define that if the ant select the base vertex $u_i \in U$, then the ant will follow the paths $u_i - v_j - u'_i$ or the path $u_i - u'_i$ back to the vertex u'_i, we can see U and U' same because if the algorithm visits the base vertex $u_i \in U$, it will visit its corresponding virtual vertex $u'_i \in U'$ next.[3] Therefore, the algorithm can just visit each $u_i \in U$ and select a matching edge $(u_i, v_j) \in E$. It will help this method more efficient and applicable for the minimum-weighted BM problems. We call this new method "ant-matching."

In the ACO algorithm, the ants choose the next vertex independently based their decision on the density of pheromone and a heuristic function. In the ant-matching algorithm, the ants choose the next vertex based on the same mechanisms. In each iteration t, the ants choose the next base vertex $u_i \in \text{Avail}^k(U)(t)$ based on the density of pheromone $\tau_{p(i)i}(t)$ as Eq. (3.1), where $\text{Avail}^k(U)(t)$ is the set of available base vertices that the kth ant has yet to visit at the tth iteration and $u_{p(i)} \in U$ is the predecessor of the vertex u_i. Let $N_i^k(t) \subseteq V$ be the set of the kth ant's available adjacent vertices of u_i at the tth iteration. If $|N_i^k(t)| > 0$, each of the m ants chooses one vertex $v_j \in N_i^k(t)$ to do matching based on the density of pheromone $\tau_{ij}(t)$ and a heuristic function $\eta_{ij}(t)$ as Eq. (2.1), then adds the weight w_{ij} to $L^k(t)$, where $L^k(t)$ is the length of the kth ant's tour in iteration t. Otherwise, the

[3]Lyuu: I do not think this is true, as we discussed last Thursday. You want it to be true, but it may not be true for a general TSP solver. you need to say YOUR TSP solver will impose this condition

Hung-Pin: I add a sentence before this sentence to explain the condition. Besides, at the end of the section 3.1, I redefine the TSP problem in the transformation. It is not a general TSP. It likes a related TSP problem which generalized TSP deals with "states". Thus, I redefine the TSP problem.

ant does nothing but adds a large associated weight $w_{\max}$ to $L^k(t)$. After each of m ants visiting all the vertices in the set U, update the pheromone on each edge and start the next iteration. The pheromone deposited on each edge $(u_i, u_k) \in E'$ represents an order relation such that if one ant chooses the vertex u_i, it will choose the vertex u_k next time. And a large amount of pheromone on the edge (u_i, v_j) represents that the edge (u_i, v_j) is a preferred matching edge. Thus, we have the order relations to do matching on the vertex set U and the preferred matching edges on each vertex $u_i \in U$. We can find the solutions of the matching problems according to the density of pheromone. The algorithm is listed in Algorithm 4, and Algorithm 5 is the detail.

$$p^k_{p(i)i} = \frac{[\tau_{p(i)i}(t)]^\alpha}{\sum_{l \in \text{Avail}^k(U)(t)} [\tau_{p(i)l}(t)]^\alpha} \quad \forall i \in \text{Avail}^k(U)(t) \tag{3.1}$$

Algorithm 4 Ant-Matching Algorithm

```
for t = 1 to t_max do
  for k = 1 to NumberOfAnts do
    choose u_i ∈ Avail^k(U)(t) based on τ_{p(i)i}(t)
    choose vertex v_j ∈ N_i^k(t) based on τ_ij(t) and η_ij(t)
  end for
  update pheromone
end for
```

3.1.3 The Complexity of Ant-Matching Algorithm

In the ant-matching algorithm, each ant visits each vertex u_i in the vertex set U and searchs each edge adjacent to u_i exactly once. Thus, generating a solution costs $O(U + E)$. Moreover, it costs $O(U^2 + E)$ to update the density of pheromone on each edge (u_i, u_k) where $u_i, u_k \in U$ and on each edge $(u_i, v_j) \in E$. If there are m ants and $t_{\max}$ iterations. The total complexity of the ant-matching algorithm is $O(t_{\max}(m(U + E) + U^2 + E))$.

3.2 Metropolis Algorithm for Maximum-Weighted Bipartite Matching

In a weighted bipartite graph $G = (U, V, E)$, we can choose the vertex set U or V arbitrarily to do matching. We choose the vertex set U for illustrations. We

Algorithm 5 Ant-Matching Algorithm in Detail

Input: Weighted bipartite graph $G = (U, V, E)$
and each edge $(u_i, v_j) \in E$ has a weight w_{ij}.
$\texttt{MatchingList}_{Best} = []$
$\texttt{Weight}_{\min} = \infty$
for $t = 1$ to $t_{\max}$ **do**
{Generate Solution}
for $k = 1$ to $\texttt{NumberOfAnts}$ **do**
Avail$^k(U)(t) = U$;
$\texttt{MatchingList}_k(t) = []$
$\texttt{Weight}_k(t) = 0$
while $|\text{Avail}^k(U)(t)| > 0$ **do**
choose available $u_i \in \text{Avail}^k(U)(t)$ based on $\tau_{ij}(t)$
remove u_i in Avail$^k(U)(t)$
if u_i has available adjacent vertices $N_i^k(t) \subseteq V$ **then**
choose $v_j \in N_i^k(t)$ based on $\tau_{ij}(t)$ and η_{ij}
add (u_i, v_j) into $\texttt{MatchingList}_k(t)$
$\texttt{Weight}_k(t) = \texttt{Weight}_k(t) + w_{ij}$
else
add (u_i, ϕ) into $\texttt{MatchingList}_k(t)$
$\texttt{Weight}_k(t) = \texttt{Weight}_k(t) + w_{\max}$
end if
end while
if $\texttt{Weight}_k(t) \leq \texttt{Weight}_{\min}(t)$ **then**
$\texttt{MatchingList}_{Best} = \texttt{MatchingList}_k(t)$
end if
end for
{Update pheromone}
for $k = 1$ to $\texttt{NumberOfAnts}$ **do**
for $l = 1$ to $|\texttt{MatchingList}_k(t)|$ and $(x_l, y_l) \in \texttt{MatchingList}_k(t)$ **do**
$\Delta\tau^k_{x_{l-1},x_l}(t) = 1/\texttt{Weight}_k(t)$
if $y_l \neq \phi$ **then**
$\Delta\tau^k_{x_l,y_l}(t) = 1/\texttt{Weight}_k(t)$
else
$\Delta\tau^k_{x_l,x_l}(t) = 1/\texttt{Weight}_k(t)$
end if
end for
end for
for $(u_i, v_j) \in E$ **do**
$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau^k_{ij}(t)$
end for
for $u_i, u_j \in U, u_i \neq u_j$ **do**
$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau^k_{ij}(t)$
end for
$t = t + 1$;
end for

apply the Metropolis algorithm on the maximum weighted bipartite matching (maximum-weighted BM) problem. Let x be the current search state. $x \in \{0,1\}^{|E|}$ describes the set of matchings (u_i, v_j), where $x_{ij} = 1$ denotes a matching edge, $x_{ij} = 0$ denotes not.

In each iteration, choose $(u_i, v_j) \in E$ at random and flip x_{ij}. Thus, in the current state x, choose a new search state $x' = \{x'_{00}, ..., x'_{|U||V|}\}$ where $x'_{ij} = 1 - x_{ij}$, and $x'_{gh} = x_{gh}$, if the edges $(u_g, v_h) \neq (u_i, v_j)$.

After choosing a new state x', using a fitness function f to decide if we choose the new search state x' as the next state. If $f(x') \geq f(x)$, select x'. If $f(x') < f(x)$, select x' with the probability $e^{\frac{f(x')-f(x)}{K}}$, where K is a constant, otherwise, select x. The algorithm is illustrated in Algorithm 6.

Algorithm 6 Apply Metropolis Algorithm to Maximum-Weighted BM

Input: Weighted bipartite graph $G = (U, V, E)$
and each edge $(u_i, v_j) \in E$ has a weight w_{ij}.
loop
choose $(u_i, v_j) \in E$ at random and flip x_{ij}
sample α from $U(0, 1)$
if $f(x') \geq f(x)$ **then**
select x'
else
if $\alpha \leq e^{\frac{f(x')-f(x)}{K}}$ **then**
select x'
end if
end if
end loop

Here we choose 0^m as the starting state x for the maximization problems and choose a fitness function f, where $f(x) = 0$ for the state x which there exist two edges in the matching sharing a common vertex, and $f(x) = \sum_{u_i \in U, v_j \in V} x_{ij} w_{ij}$ for the state x, not which has two edges that share the same vertex. This fitness function f will guarantee the matching is reasonable in each iteration.

3.2.1 The Complexity of Metropolis Algorithm

In the Metropolis algorithm, generating a new search state costs a little. To generate a new search state we choose a edge $(u_i, v_j) \in E$ at random and flip x_{ij}. It will cost $O(1)$ to generate a new search state. Then, it costs $O(1)$ to calculate the weight of the new search state and costs $O(1)$ to verify if the

new state is reasonable. If the new search state is better than the best state, record the new search state as the best state and it will cost $O(U)$. Thus, if there are c iterations, the complexity of the Metropolis algorithm is $O(cU)$.

Chapter 4

Experimental Results

After introducing the two methods on solving the weighted bipartite matching problems, we write them in programs to do some experiments and compare them with the Hungarian algorithm, a well-known algorithm for solving the weighted bipartite matching problems. In a weighted bipartite graph $G = (U, V, E)$ and each edge $(u_i, v_j) \in E$ has an associated weight w_{ij}. We replace each edge weight w_{ij} with $\frac{1}{w_{ij}}$ to transform a maximum weighted bipartite matching problem to a minimum weighted bipartite matching problem. Thus, we can use these algorithms to solve the maximum or minimum weighted bipartite matching problem.

4.1 Comparison of Complexity

Before the experiments, we compare the complexity first. The complexity of each algorithm is listed as follows:

Table 4.1: Comparison of complexity

Algorithm	Complexity
Ant-Matching	$O(t_{\max}(m(U+E)+U^2+E))$
Metropolis Algorithm	$O(cU)$
Hungarian Algorithm	$O(V^3)$

In Table 4.1, we can easily find that the complexity of the Metropolis Algorithm is the smallest. But we can't easily compare the complexity of the ant-matching algorithm with the Hungarian algorithm. To verify that if the

ant-matching algorithm is faster than the Hungarian algorithm or not, we will do some experiments. Moreover, we also test if the Metropolis algorithm and the ant-matching algorithm can generate a reasonable solution for the weighted bipartite matching problem within a smaller time and discuss their usability.

4.2 Graph Sampling Algorithm

Before the experiments, we build some graph generating algorithms to generate the weighted bipartite graphs. There are two graph generating algorithms:

- Generating algorithm of complete bipartite graphs.

  The generating algorithm is used to generate the complete weighted bipartite graph, which $\forall u_i \in U, \forall v_j \in V, \exists (u_i, v_j) \in E$.

- Generating algorithm of sparse bipartite graphs.

  The graph generating algorithm generate a weighted bipartite graph $G = (U, V, E)$ randomly and each vertex $u_i \in U$ has at most a specified number of adjacent vertices. It does not ensure that the generated graphs has a perfect matching.

We will use these algorithms to generate the graph samples to do the experiments.

4.3 Results

In the experiments, we configure some variables at first. In the ant-matching algorithm, there are 200 iterations and 20 ants in each iteration and we make $\alpha = 2$ and $\beta = 1$ to make the relative influence of the pheromone doubles that of the distance. In the Metropolis algorithm, we initialize $1,000,000,000$ cycles to generate the new search states. After configuring the variables, we start the experiments, and the results are listed as follows. We will analyze the results in different aspects:

- Time
  The cost of time in solving the weighted bipartite matching problems.

- Weight
  The maximum weight found by each algorithm. We solve the maximum weighted bipartite matching problems in our experiments.

Size	Ant-Matching	Metropolis	Size	Ant-Matching	Metropolis
100x100	95.13%	60.87%	1100x1100	91.20%	52.07%
200x200	94.21%	56.73%	1200x1200	91.10%	52.29%
300x300	92.94%	55.75%	1300x1300	91.52%	51.98%
400x400	92.93%	54.93%	1400x1400	90.96%	51.59%
500x500	92.94%	53.81%	1500x1500	90.93%	51.64%
600x600	92.67%	54.49%	1600x1600	90.95%	51.72%
700x700	92.36%	52.79%	1700x1700	90.95%	50.92%
800x800	91.66%	52.40%	1800x1800	90.75%	51.25%
900x900	92.16%	52.84%	1900x1900	90.58%	51.59%
1000x1000	91.73%	52.54%	2000x2000	90.62%	51.71%

Table 4.2: The weights as proportions to the weights of the optimal solution on the complete bipartite graph.

4.3.1 Results on Complete Bipartite Graphs

We use the ant-matching algorithm, the Metropolis algorithm, and the Hungarian algorithm to solve the maximum weighted bipartite matching problems on each complete bipartite graph with different problem size. Table 4.5 is the numeral result in solving the matching problems. In Table 4.5, the result found by the Hungarian algorithm is optimal. Table 4.2 lists the weights found by the ant-matching algorithm and the Metropolis algorithm as proportions to the weights of the optimal solution. In Table 4.5, we can see that the weights found by the ant-matching algorithm are close to the weights of the optimal solution. In Figure 4.1, the running time of the ant-matching algorithm increases gradually depending on the sizes of matching problems. Even the running time of the ant-matching algorithm is larger than the running time of the Hungarian algorithm, but the growth rate of running time of the ant-matching algorithm is smaller than the Hungarian algorithm. Thus, if the problem size is very large, the ant-matching algorithm will be faster than the Hungarian algorithm.

The weights found by the Metropolis algorithm are close to the half of the weights of the optimal solution. But the running time of the Metropolis algorithm is very small. In practice, we can find the result with less number of cycles in the Metropolis algorithm. The initialized $1,000,000,000$ cycles is used to raise the running time, it will help to draw the transitions on different problem sizes in Figure 4.1.

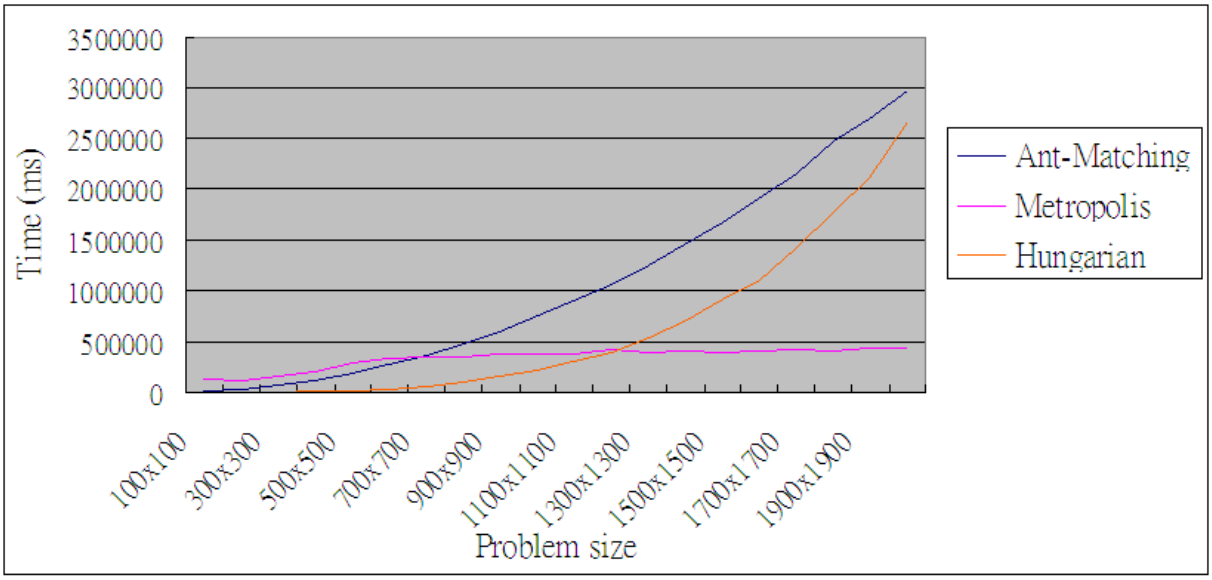

Figure 4.1: The running time of each algorithm on the complete bipartite graph.

4.3.2 Results on Sparse Bipartite Graphs

In the sparse bipartite graph $G = (U, V, E)$ which each vertex $u_i \in U$ has at most $2\% * |V|$ edges, the numeral result of this matching problem is listed in Table 4.10. Table 4.3 is the result of weights found by the ant-matching algorithm and the Metropolis algorithm as proportions to the weights of the optimal solution. In Table 4.3, the weights found by the ant-matching algorithm are close to the weights of the optimal solution. And in Figure 4.2, the running time of the ant-matching algorithm is much less than that of the Hungarian algorithm as the problem size increases. Thus, in a large scale bipartite matching problem, we can use the ant-matching algorithm to find a solution in a smaller time, and the result is close to the optimal solution.

In Table 4.3, we can see the weights found by the Metropolis algorithm are close to half of the weights of the optimal solution. In Figure 4.2, even we initializes $1,000,000,000$ cycles in the Metropolis algorithm, the running time is still small. Thus, if we want to find a matching in a rapid time and don't care the weight, we can use the Metropolis algorithm to find a solution.

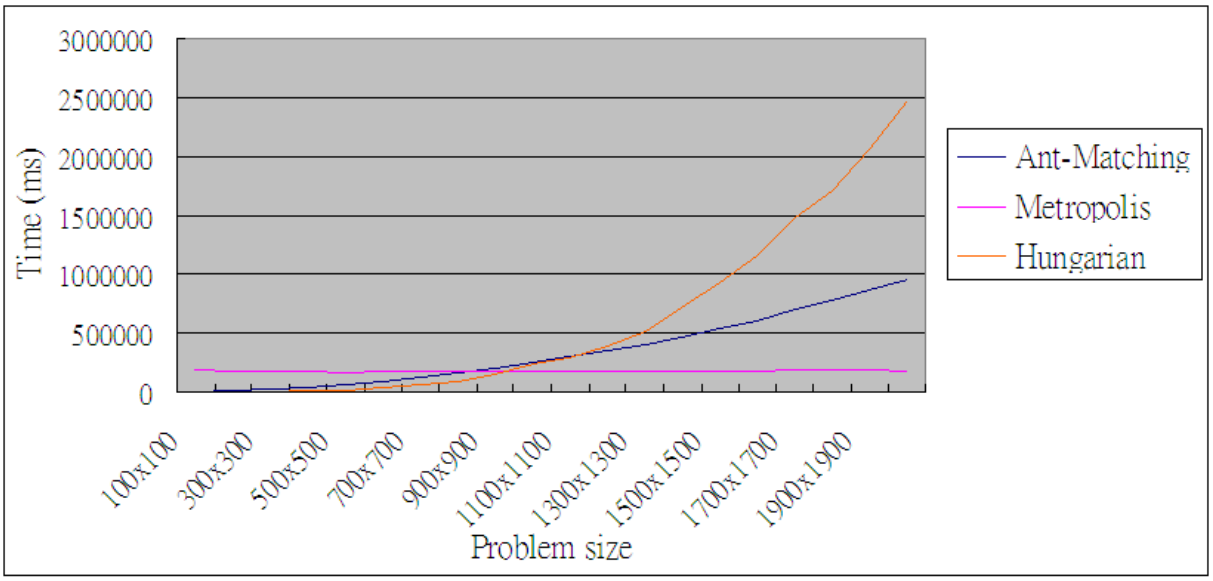

Figure 4.2: The running time of each algorithm on the bipartite graph where each vertex $u_i \in U$ has at most $2\% * |V|$ edges.

Size	Ant-Matching	Metropolis	Size	Ant-Matching	Metropolis
100x100	96.81%	72.27%	100x100	95.57%	47.06%
200x200	98.42%	56.44%	200x200	95.18%	47.56%
300x300	98.35%	53.23%	300x300	95.53%	46.97%
400x400	97.37%	50.95%	400x400	94.82%	47.81%
500x500	97.31%	48.83%	500x500	95.28%	47.09%
600x600	97.11%	48.83%	600x600	95.03%	47.43%
700x700	96.64%	47.78%	700x700	94.58%	47.00%
800x800	96.86%	48.52%	800x800	94.59%	47.25%
900x900	96.09%	48.37%	900x900	94.56%	46.99%
1000x1000	95.80%	48.16%	1000x1000	94.28%	47.30%

Table 4.3: The weights as proportions to the weights of the optimal solution on the bipartite graph where each vertex $u_i \in U$ has at most $2\% * |V|$ edges.

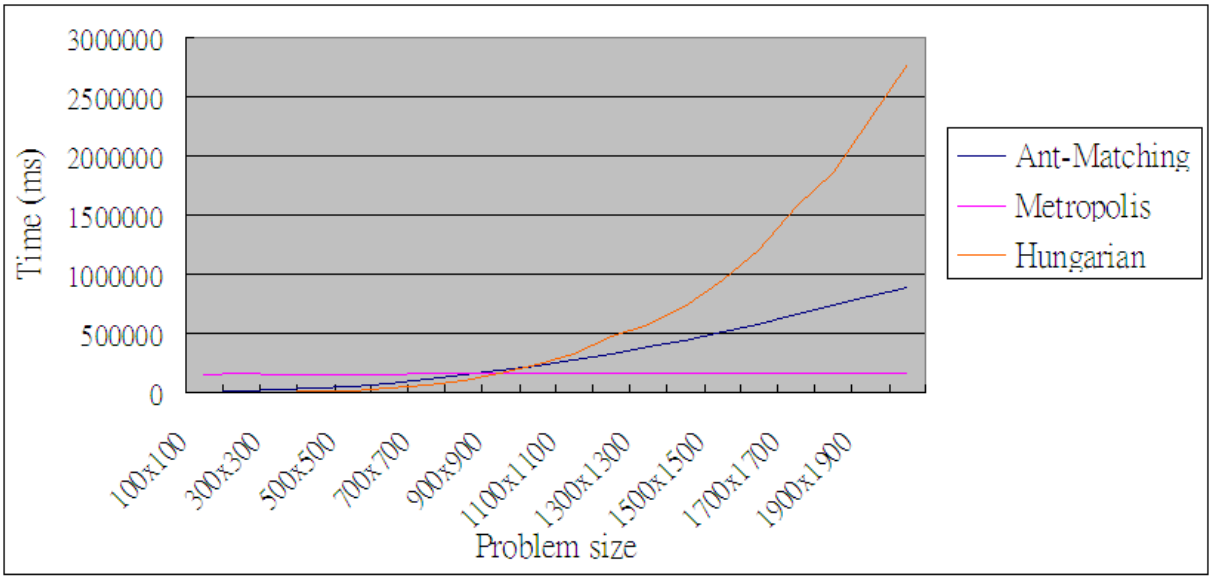

Figure 4.3: The running time of each algorithm on the bipartite graph where each vertex $u_i \in U$ has at most 5 edges.

Here is another sparse bipartite graph $G = (U, V, E)$ where each vertex $u_i \in U$ has at most 5 edges, and the numeral result of this matching problem is listed in Table 4.11. Figure 4.3 is the graphic illustration of the time costs by each algorithm. We can see that the running times of the ant-matching algorithm and the Metropolis algorithm are still much less than the running times of the Hungarian algorithm. In Table 4.4, the weights found by the ant-matching algorithm are close to the optimal solutions. Thus, in a large scale and very sparse graph, we can use the ant-matching algorithm to find a matching in a smaller time and the weight of the matching is close to the weight of the optimal solution.

Size	Ant-Matching	Metropolis	Size	Ant-Matching	Metropolis
100x100	98.23%	66.07%	1100x1100	97.53%	50.51%
200x200	97.75%	59.71%	1200x1200	97.17%	51.07%
300x300	96.00%	56.68%	1300x1300	97.61%	51.01%
400x400	96.57%	56.24%	1400x1400	97.33%	49.85%
500x500	97.93%	54.29%	1500x1500	97.06%	49.63%
600x600	96.74%	52.85%	1600x1600	97.38%	49.12%
700x700	97.27%	52.71%	1700x1700	97.01%	49.21%
800x800	97.35%	51.83%	1800x1800	97.22%	49.01%
900x900	97.42%	51.61%	1900x1900	96.99%	48.63%
1000x1000	96.80%	51.61%	2000x2000	97.06%	48.65%

Table 4.4: The weights as proportions to the weight of the optimal solution on the bipartite graph where each vertex $u_i \in U$ has at most 5 edges.

4.3.3 Results on Ant-Matching Algorithm

In addition to the above-mentioned bipartite matching problems, we also solve the matching problems on different bipartite graphs in our experiments. The numeral results are listed in Tables 4.5–4.13. We will analyze the results and indicate the features of each algorithm.

In the ant-matching algorithm, Figure 4.4 is the graphic illustration for the running times on each different bipartite matching problem. In Figure 4.4, we can easily see that the running time of the ant-matching algorithm depends on number of edges. The less the number of edges in the bipartite graph, the less time the ant-matching algorithm costs on solving the matching problem.

Figure 4.5 illustrates the transitions of the weights on each different graph on different problem sizes. In Figure 4.5, as the problem size increases, the weights found by the ant-matching algorithm on each different graph as proportions to the weights of the optimal solution decrease slowly. Besides, the weights as proportions to the optimal solution are greater than 90%. With the advantages of the smaller running time, we can use the ant-matching algorithm to find a matching in a smaller time, which the weight is close to the optimal solution.

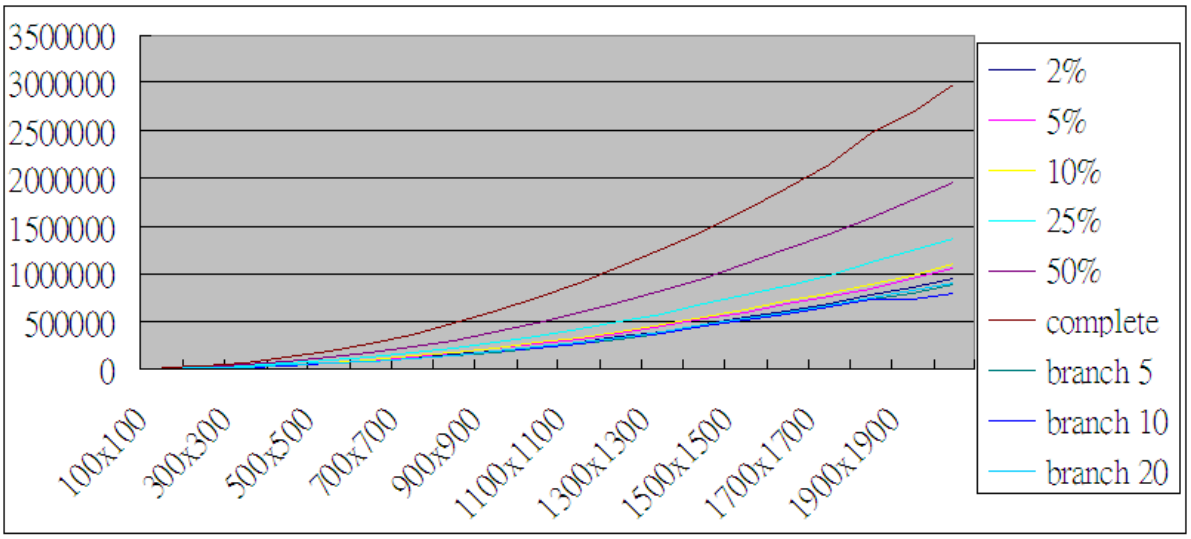

Figure 4.4: The running times of the ant-matching algorithm on each different bipartite graph.

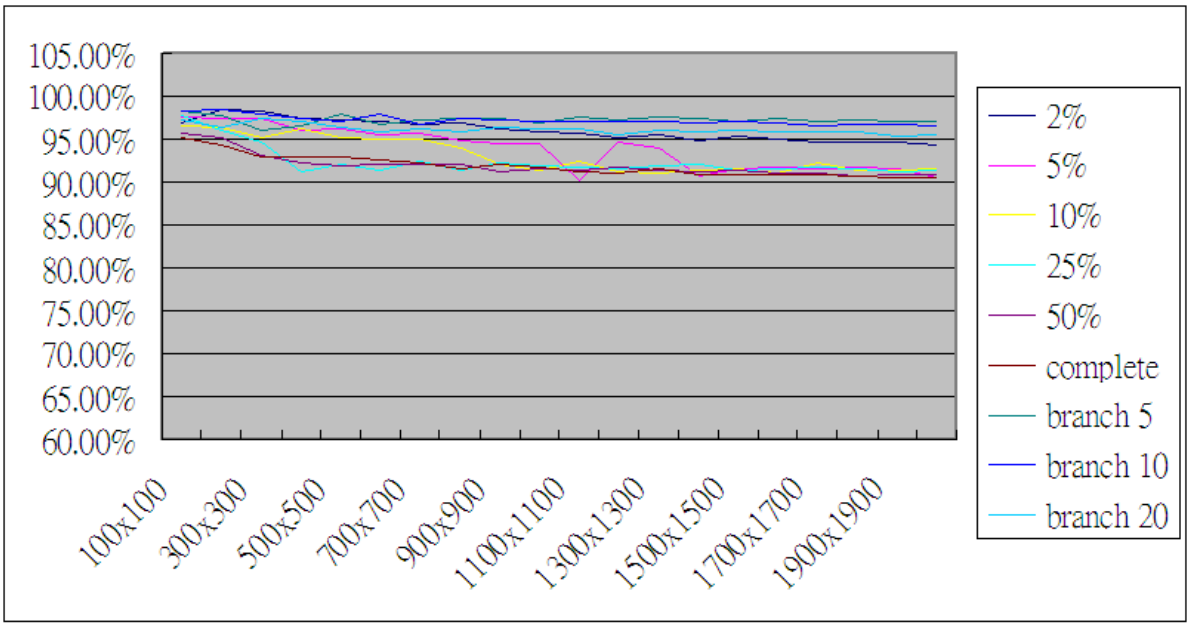

Figure 4.5: The weights found by the ant-matching algorithm as proportions to the weights of the optimal solution on each different bipartite graph.

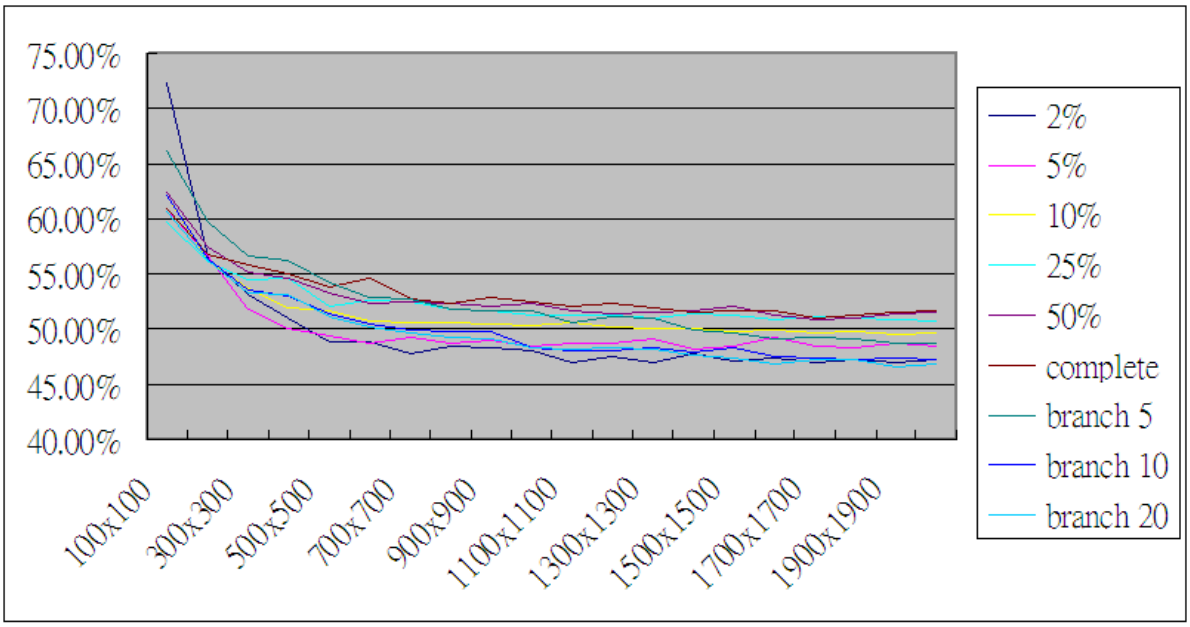

Figure 4.6: The weight found by the Metropolis algorithm as proportions to the weights of the optimal solution on each different bipartite graph.

4.3.4 Results on Metropolis Algorithm

The running time of the Metropolis algorithm is much smaller than the other two algorithms. Even the cost times have slight change on different problem sizes, but they are still smaller than the other two algorithms. But the weights found by the Metropolis algorithm are not as impressive. In Figure 4.6, as the problem size increases, we can see that the weights found by the Metropolis algorithm as proportions to the optimal solution lie in between 45% and 55%. Thus, if we don't care the weights found by the Metropolis algorithm, we can use the Metropolis algorithm to solve the bipartite matching problem within a small time.

Table 4.5: Results on each algorithm for complete bipartite graphs.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	12422	127454	78	93.76212784	59.99137975	98.56287138
200x200	28328	126140	1094	186.7533713	112.4511809	198.2346859
300x300	67969	144172	4125	277.2871941	166.3360957	298.3417949
400x400	119875	198625	10438	370.1943328	218.8004151	398.3521658
500x500	186891	281625	20219	463.1564475	268.1571713	498.3220945
600x600	268281	326422	34484	554.4458956	326.003062	598.293561
700x700	363000	337172	64781	644.9132094	368.5894278	698.2666038
800x800	474672	345922	105953	731.7736169	418.3070854	798.3324054
900x900	602469	371109	146204	827.9483202	474.6728236	898.3740431
1000x1000	742375	376828	204000	915.8608818	524.5644648	998.4223776
1100x1100	902640	372531	300031	1001.691501	571.8960081	1098.35845
1200x1200	1068609	417531	395125	1091.736157	626.5668502	1198.356802
1300x1300	1253609	385781	537032	1188.241281	674.8935237	1298.347934
1400x1400	1448406	400343	708890	1272.006474	721.3961744	1398.370254
1500x1500	1668125	395672	907375	1362.458324	773.7970696	1498.416469
1600x1600	1895234	399672	1098031	1453.659121	826.648454	1598.367255
1700x1700	2150922	419453	1424421	1544.753336	864.8778489	1698.379739
1800x1800	2463172	409375	1785843	1631.940541	921.6115056	1798.37223
1900x1900	2691703	428485	2110999	1719.485088	979.4262441	1898.372438
2000x2000	2973485	426828	2662853	1810.80745	1033.392259	1998.351143

Table 4.6: Results on each algorithm for the bipartite graphs where each vertex has at most $50\% * |V|$ edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	4922	127531	94	92.72780627	60.36169385	96.86275204
200x200	19328	131110	1047	187.1510867	113.034867	196.8652595
300x300	43485	133188	3953	276.3326725	163.4122039	296.5828484
400x400	77406	142297	8813	366.0217489	216.409315	396.7368031
500x500	121000	202375	20172	456.8639815	264.6425751	496.741958
600x600	174125	257937	34109	549.5275268	312.2918048	596.6812997
700x700	236500	291093	56641	641.3882497	366.1369673	696.8035924
800x800	308922	318203	92219	734.1128885	417.2432508	796.7329625
900x900	390672	345640	139641	818.4287195	467.2815802	896.6588299
1000x1000	482782	361360	199453	913.0191076	522.0801954	996.915783
1100x1100	588250	373421	312516	1002.04486	566.1089635	1096.640389
1200x1200	701359	389422	403218	1098.590562	614.3345073	1196.7346
1300x1300	826625	380875	569453	1184.955507	668.5840256	1296.792552
1400x1400	961250	385766	707172	1273.914744	720.995946	1396.763585
1500x1500	1101641	393406	900985	1367.074562	779.0565391	1496.718364
1600x1600	1254047	392016	1133766	1453.96136	817.4839471	1596.719021
1700x1700	1414000	410844	1365297	1545.651154	863.5369943	1696.774299
1800x1800	1586391	413610	1691687	1630.413165	917.1276354	1796.742078
1900x1900	1766453	404891	2063985	1724.556971	974.1730304	1896.714927
2000x2000	1959750	421547	2492984	1813.18771	1028.635455	1996.763675

Table 4.7: Results on each algorithm for the bipartite graphs where each vertex has at most $25\% * |V|$ edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	5875	121047	63	91.08770013	55.61176776	93.14795471
200x200	13375	123406	640	185.5426607	108.4255932	193.4444159
300x300	31531	125875	2494	277.7223822	159.6677912	293.3814713
400x400	56062	139734	6453	359.3233843	214.487364	393.5737603
500x500	87578	135656	14573	454.6455209	256.6291784	493.3040891
600x600	125234	143218	27603	542.8681129	312.7091165	593.6723418
700x700	170110	175547	49844	640.9559432	364.140993	693.6291807
800x800	221953	217125	79956	726.0061843	411.0983714	793.5412079
900x900	279906	244953	126168	824.9739264	461.2034476	893.4246839
1000x1000	344454	271500	176792	914.2384497	509.7117555	993.6410455
1100x1100	416157	288000	249002	1002.637891	560.443724	1093.374392
1200x1200	495171	306469	406296	1092.740705	611.8370349	1193.319361
1300x1300	580250	324141	567642	1188.161513	659.0457447	1293.535325
1400x1400	690297	330562	711439	1284.472726	716.8991148	1393.315201
1500x1500	772047	339609	893031	1365.775754	765.317049	1493.521154
1600x1600	878171	360563	1174935	1459.460002	809.9331559	1593.2746
1700x1700	992062	359453	1358336	1553.841628	865.6447168	1693.54095
1800x1800	1111375	365704	1757088	1641.120297	913.3638176	1793.34458
1900x1900	1236610	376813	1951383	1727.499884	964.0143746	1893.537905
2000x2000	1370641	384891	2623716	1822.796555	1009.587833	1993.525222

Table 4.8: Results on each algorithm for the bipartite graphs where each vertex has at most $10\% * |V|$ edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	4859	126953	78	81.98107088	52.7438673	84.89513053
200x200	10531	126250	655	177.5833111	103.462862	184.1782001
300x300	25203	127531	2509	270.012258	152.4335902	284.0442781
400x400	45078	128047	6312	368.4031894	198.9246604	383.3327097
500x500	70110	134734	13108	459.6486038	249.895601	483.225838
600x600	100797	132078	29364	553.28167	295.4547724	582.7038311
700x700	136641	135094	50156	648.5281659	344.969619	682.8400679
800x800	177578	137641	85630	734.944871	395.7375045	782.5284336
900x900	224875	141422	110333	813.9211635	445.6657887	882.8959439
1000x1000	279047	159000	183946	898.1066518	494.2512265	982.8046155
1100x1100	336750	183922	301091	1001.708216	547.6307167	1082.368178
1200x1200	401532	203891	389089	1078.901513	592.5325897	1182.699767
1300x1300	466312	227079	555828	1168.468644	642.0305438	1282.202102
1400x1400	538859	246172	695479	1263.152943	690.8885926	1382.615354
1500x1500	627906	258297	888542	1356.610098	738.1014563	1482.409127
1600x1600	717609	292109	1118357	1440.810854	789.8336527	1582.324841
1700x1700	796484	285344	1410424	1552.519654	834.9458198	1682.565528
1800x1800	889594	320110	1732306	1628.851506	887.6851586	1782.206735
1900x1900	989969	334172	2209187	1721.188425	932.3221662	1882.134316
2000x2000	1097375	312610	2609406	1817.020365	983.5757335	1982.35423

Table 4.9: Results on each algorithm for the bipartite graphs where each vertex has at most $5\% * |V|$ edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	4766	165344	63	72.55963638	45.26483225	74.35403938
200x200	10516	157875	953	166.5589723	96.90158747	171.122618
300x300	24313	144922	3766	261.3024405	139.076186	268.454759
400x400	43203	152797	9141	353.8664193	184.2110728	368.5471482
500x500	67344	152594	17594	451.1410476	231.1197816	468.8982772
600x600	96468	153016	34468	542.8622255	277.0767393	569.0571676
700x700	131047	154750	61953	638.3631942	328.4142032	667.9605229
800x800	170656	154547	94297	727.6950692	373.3781007	767.8706861
900x900	215015	167984	138031	820.9100048	424.835042	869.3211035
1000x1000	265735	158593	204500	914.296472	469.8069121	967.6872272
1100x1100	321593	163719	276672	962.8716941	519.3492103	1068.466494
1200x1200	381282	161750	429046	1105.298322	567.2015287	1167.502663
1300x1300	442172	177734	545843	1190.38188	621.7023715	1267.845605
1400x1400	530797	181407	709016	1241.8753	660.982468	1369.133415
1500x1500	589953	188625	927703	1343.170651	711.0699621	1467.87038
1600x1600	676547	194172	1142250	1440.372647	770.5902628	1568.227608
1700x1700	767094	189266	1388938	1527.22796	808.5954361	1667.407893
1800x1800	847500	195015	1574485	1620.858498	855.3864881	1767.395637
1900x1900	949563	206562	2055875	1710.799868	908.3755728	1867.813619
2000x2000	1047141	216719	2510532	1780.25997	954.8678434	1967.058802

Table 4.10: Results on each algorithm for the bipartite graphs where each vertex has at most $2\% * |V|$ edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	4500	177890	63	55.18013862	41.19267619	57.00068792
200x200	9641	171047	953	138.7101322	79.54238642	140.9311258
300x300	22422	166000	4203	224.0294476	121.265374	227.7981038
400x400	39266	162171	8484	320.7804736	167.8679841	329.4465666
500x500	61000	160546	18938	414.5875839	208.0267164	426.0298118
600x600	87531	161469	36734	509.6318166	256.243303	524.8016639
700x700	118922	161094	58453	604.8175264	298.9940862	625.8276844
800x800	154672	161297	92375	701.3100012	351.32063	724.0302865
900x900	195234	162375	136469	791.2816921	398.2672566	823.4463629
1000x1000	240234	166891	232406	883.9602428	444.3647732	922.7610247
1100x1100	290187	168672	282844	975.0802044	480.149395	1020.29556
1200x1200	345016	169844	389687	1069.0864	534.2566852	1123.255501
1300x1300	402484	170484	519985	1167.66944	574.1391651	1222.317805
1400x1400	467125	172625	703188	1253.027779	631.881189	1321.521038
1500x1500	537062	169718	927734	1352.065866	668.2464149	1419.073406
1600x1600	610609	171609	1140047	1445.338479	721.3506841	1520.953155
1700x1700	691313	175344	1482578	1532.560839	761.5553572	1620.453898
1800x1800	772937	176188	1704094	1627.420304	812.9416641	1720.464429
1900x1900	857343	180047	2057016	1720.942026	855.1933812	1819.961713
2000x2000	955719	172312	2468094	1809.907837	908.0119217	1919.65387

Table 4.11: Results on each algorithm for the bipartite graphs where each vertex has at most 5 edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	2610	144125	78	71.01632804	47.76754179	72.29785256
200x200	9547	147954	1078	144.3261467	88.1606114	147.6499849
300x300	20719	142969	3625	209.4333408	123.6406937	218.1550131
400x400	36109	145453	9172	274.9515693	160.1249942	284.7093798
500x500	55782	145828	18890	353.1550806	195.7581271	360.6097908
600x600	80657	146922	36235	418.2425001	228.5038435	432.3359955
700x700	109250	155937	63484	491.7125773	266.4793163	505.5378906
800x800	141719	149968	99828	566.709767	301.7321196	582.129979
900x900	176953	151015	155328	642.0896993	340.1776528	659.1108085
1000x1000	219563	150344	227000	705.2530634	376.0164821	728.5519891
1100x1100	270328	153859	325188	782.8807543	405.4215758	802.6950651
1200x1200	321468	150578	468969	836.8582344	439.8375131	861.2299184
1300x1300	379110	152547	581547	919.6835791	480.6155806	942.1875666
1400x1400	440797	153016	714781	991.7097025	507.921043	1018.95962
1500x1500	507610	151703	945812	1058.513904	541.3211637	1090.621186
1600x1600	579688	152766	1186656	1127.451401	568.691967	1157.792919
1700x1700	653219	155954	1581266	1191.915721	604.5855135	1228.676967
1800x1800	726531	153407	1854156	1270.577832	640.5386886	1306.944366
1900x1900	803360	154609	2297234	1340.351306	672.0355465	1381.98123
2000x2000	883531	154938	2773828	1414.73028	709.1355942	1457.643701

Table 4.12: Results on each algorithm for the bipartite graphs where each vertex has at most 10 edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	2906	136203	78	83.44144831	52.7438673	84.89513053
200x200	10110	138828	1031	166.0809005	94.96961994	168.6276584
300x300	21907	134329	3390	249.8521349	136.8434872	255.0612198
400x400	38250	135875	9000	331.8127693	180.8773697	340.6977865
500x500	58516	137531	18329	412.310661	218.2413746	424.5380644
600x600	82735	138672	33031	498.8163551	256.5263599	508.9098678
700x700	112437	139640	55453	573.2013592	295.6081407	592.9172566
800x800	145750	139359	90516	661.1020851	337.5970951	678.512908
900x900	183062	139547	129547	745.2667431	381.3007525	766.6953569
1000x1000	226360	139906	192563	824.9628987	410.934408	850.360086
1100x1100	277265	141469	279922	905.3093731	448.8447992	932.823552
1200x1200	330922	140922	358219	984.2998233	488.0119238	1015.005472
1300x1300	384031	141656	473188	1067.979441	532.281959	1100.93529
1400x1400	445844	142203	657140	1148.890971	567.5875857	1185.088969
1500x1500	510875	142828	860047	1231.703836	614.038624	1269.068467
1600x1600	576782	143797	1056968	1308.3388	641.7723808	1350.805353
1700x1700	648484	152625	1383500	1380.118821	679.1008841	1430.666428
1800x1800	726531	161047	1688266	1468.299406	718.4229659	1518.556813
1900x1900	731750	155047	2000125	1553.872123	760.1485369	1605.381318
2000x2000	799125	156266	2381109	1628.283939	796.7113401	1687.415992

Table 4.13: Results on each algorithm for the bipartite graphs where each vertex has at most 20 edges.

	Time(ms)			Weight		
Size	Ant	Metropolis	Hungarian	Ant	Metropolis	Hungarian
100x100	3344	132703	93	88.9940445	55.41596066	91.47317065
200x200	11078	134531	1125	177.5010706	103.462862	184.1782001
300x300	23453	129156	3703	268.6427545	147.1410087	275.9809938
400x400	40406	130719	9844	357.454115	196.0947465	368.3861489
500x500	61750	133797	19610	442.6658518	234.5835032	459.3350209
600x600	86985	136078	32734	528.6467732	276.5082769	551.4035986
700x700	117078	132828	59937	617.6368435	319.0734327	642.3899665
800x800	151031	133563	92110	703.5222714	361.9469399	734.8569713
900x900	189203	133875	149500	796.7183423	405.7479686	827.3289196
1000x1000	231671	135296	203485	883.53109	444.4491947	918.7229245
1100x1100	281328	136796	289250	974.4097692	488.6040136	1013.999974
1200x1200	334265	139438	397609	1055.555762	535.0272726	1104.694687
1300x1300	392110	139063	551984	1149.346358	577.1818476	1197.436389
1400x1400	454032	139984	652531	1232.319395	613.9420703	1285.780484
1500x1500	520484	142657	876735	1324.146313	653.224308	1378.582993
1600x1600	589375	139704	1101844	1407.439378	689.6848264	1469.744728
1700x1700	662219	140844	1348828	1494.953889	736.8478025	1560.645091
1800x1800	740328	141282	1559656	1585.711844	782.9034991	1654.742173
1900x1900	822156	141703	2015766	1661.610081	814.0886829	1745.015351
2000x2000	908079	143672	2335078	1754.354675	861.036132	1836.726347

Chapter 5

Conclusions

This thesis presents two methods for the maximum and minimum weighted bipartite matching problems. In the ant-matching algorithm, we can find a matching which the weight is close to the optimal solution. And if in a large scale weighted bipartite matching problem, using the ant-matching algorithm to find a matching will be faster than using the Hungarian algorithm. In the Metropolis algorithm, we can use the algorithm to find a matching in a very short time, but the weight of the matching as proportions to the optimal solution is not good. If we want to find a matching in a large weighted bipartite graph within a rapid time, the Metropolis algorithm is the best choice.

Besides, with the property of the ant colony optimization algorithm which the pheromone on the path changes gradually, the ant-matching algorithm will be useful in solving the dynamic weighted bipartite matching problems where the vertices and edges in the bipartite graph change with the time. In the Metropolis algorithm, in each iteration the algorithm randomly chooses an edge as a matching edge or not. It will not be influenced by the change of the bipartite graph. Thus, it is applicable on solving the dynamic weighted matching problems.

Bibliography

[1] S. Chib and E. Greenberg. (1995) Understanding the Metropolis-Hasting Algorithm. *The American Statistician*, Vol. 49, No. 4 (November 1995), pp. 327–335.

[2] J. Munkres. (1957) Algorithms for the Assignment and Transportation Problems. *Journal of the Society of Industrial and Applied Mathematics*, Vol. 5, No. 1 (March 1957), pp. 32–38.

[3] Y. Nakamichi and T. Arita. (2001) Diversity Control in Ant Colony Optimization. *Proceedings of the Inaugural Workshop on Artificial Life (AL'01)*, pp. 70–78, Adelaide, Australia, December 2001.

[4] T. Stützle, M. Dorigo. (1999) ACO Algorithms for the Traveling Salesman Problem. In K. Miettinen, M. Makela, P. Neittaanmaki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science.* Wiley, 1999.

[5] I, Wegener. (2005) Simulated Annealing Beats Metropolis in Combinatorial Optimization. *ICALP 2005.* LNCS 3580, pp. 589–601.