Experimental Study: Approximation Algorithms for Bipartite Matching with Metric and Geometric Costs

Mohanad Almiski

Abstract

The bipartite graph matching problem appears in many different contexts and is useful in a variety of algorithmic problems and contexts. An exact solution for bipartite matching runs in time O(n^3). Following a paper by Agarwal and Sharathkumar, we implement two approximation algorithms for bipartite graph matching, and show that the algorithm is useful for limited cases FFFFF [change later]. We also discuss implementation challenges, and potential work that might be done in the future to build on and improve these algorithms.

I. Introduction

A bipartite graph is a graph (V, E) with edges only from points in one set of distinct vertices to points in another set of distinct vertices. A matching of a bipartite graph is a set of edges whose vertices are distinct. A perfect matching of a bipartite graph is a matching that matches all vertices in each of the of sets. A minimum-weight perfect match is a perfect match whose selected edges are of minimum weight, the problem of finding a minimum-weight perfect matching is called the assignment problem.

The assignment problem is the motivating problem for min-weight perfect matchings, and involves assigning jobs to workers given the payment that each worker will take for doing that job. The goal is to minimize the cost of completing all the jobs, by assigning the cheapest worker for a job given the other jobs that have to be completed, and the payment that other workers will take for each job []. This problem can easily be formulated as a weighted bipartite graph which is what motivated the matching algorithmic approaches.

Bipartite graphs often appear in a variety of algorithmic problems and contexts, often with the goal of matching one set of items to another set. One example use of weighted bipartite matching problem is in computer vision, when tracking objects or features moving from one frame of a video to the next, a bipartite graph between the objects in the two frames and a distance function that assigns each edge a weight proportional to the difference between the two objects, allows for a simple way of keeping track of an object. More applications exist in computer graphics and in analysis of geometry [1]

This paper will implement two approximation algorithms that solve the assignment problem. The algorithms are based on the paper by Agarwal et al. [1], and run in time O($n^{2+\delta} log(n) log^2(1/\delta)$) and in time O($\varepsilon^{-2} n^{1+\delta} \tau(n, \varepsilon) log^2(n/\varepsilon) log(1/\delta)$) generate a O($1/\delta^{\alpha}$)-approximation cost matching ($\alpha = log_3(2) = .63$).

A. Related Work

The Hungarian algorithm was the first polynomial time algorithm for solving the assignment problem running in O(n^4) time, and was developed in 1955, by , []. It was then further refined to run in O(n^3) time by []. The Hungarian algorithm is a deterministic algorithm that works on weighted bipartite graphs, and returns a guaranteed min-cost assignment using an early idea of primal-dual methods to determine the adjustments to make to each matching.
The Hungarian algorithm was followed by a multitude of approximation algorithms that attempted to provide an approximate min-cost assignment that ran in faster time. Gabow and Tarjan were able to prove for integer edge costs that the matching could occur in time O($m\sqrt{(}n) log^{3/2}(n)$) []. [add more here]

II. Theory

The main idea that this paper presents is a modification of some of the conditions in the Gabow-Tarjan and the Hungarian algorithm it is based off of. The two algorithms presented here all rely on bipartite graphs whose edge weights are from a metric d($\cdot, \cdot$), i.e that d(a,b) + d(b,c) $\leq$ d(a,c), d(a,a) = 0, and d(x,y)=d(y,x). Therefore, these algorithms are limited to these specific types of bipartite graphs, and the modifications that the paper uses rely on this property to get their approximation.
The first modification is the consideration of different types of edges in the augmentation step. Instead of only considering edges that are already matched, the paper expands the types of edges that can be used to generate an augmenting path by including eligible edges.
The second modification is the addition of edge scaling to reduce the edge costs of the edge weights to ensure the runtime of the algorithm does not depend on the edge weights and to generate an approximation in a reasonable amount of time.
Finally, the last modification is that the paper only approximately matches a fraction of the original bipartite graph and uses the Hungarian algorithm to perfectly match the rest of the graph, increasing the accuracy of the algorithm.

The second algorithm in the paper, then takes these ideas further by making use of an Approximate Nearest Neighbor data structure to reduce the query times necessary for determining points likely to be the best match for that point. The authors of the paper then further increase the edges that the algorithm considers in the augmentation step by allowing so-called ε-eligible edges, a superset of the eligible edges from the previous algorithm.
For notation, the set A and B refer to the two sets of vertices in the bipartite graph. The set of eligible edges or set of ε-edges is a subset of the set of edges $A \times B$. $\tau(n, \varepsilon)$ refers to the update time of the ANN data structure. $\dot{(a, b)}$ refers to the edge weight function between a and b and is rounded up and scaled depending on the step in the algorithm.

A. *Eligible Edges*

[1] introduce the concept of eligible edges in their paper. An edge (a,b) is eligible if it satisfies one of the following:
(a,b) $\in$ Matches and y(a) + y(b) = d(a,b)
(a,b) $\notin$ Matches and y(a) + y(b) = d(a,b) + 1
During the search for a match, we take all eligible edges and make a set of vertex disjoint paths that start at the free vertices of B. These are used in the augmentation step to generate a matching that are 1-feasible, meaning the sum of y(a) + y(b) $\leq$ d(a,b) + 1 for all edges.

B. *Alternating Paths*

As introduced in the Hungarian algorithm, an alternating path, is simply a path starting at a free vertex of B, and ending at a free vertex of A, passing through potentially already matched vertices, with each vertex alternating between A and B in the path. The alternating path is useful in the augmenting step, where the previous matches are augmented with each augmenting path to add new matches.

C. *Overview of Algorithm 1*

Since this algorithm is heavily based on the Gabow-Tarjan algorithm [1], it will largely be very similar to their algorithm except for a couple of minor adjustments.
The algorithm works by keeping dual-weights associated to each vertex of the bipartite-graph. These dual weights are called potentials and assign a value to each vertex. A pair is matched in the algorithm when the sum of the dual weights of the pair add to their edge weight.
The algorithm runs a method called the Phase-Match until it has matched 1/3 of the vertices in the graph. Then it uses the Hungarian algorithm to perfectly match the graph generated by the remaining $n^{2/3}$ vertices.
The algorithm uses eligible edges to expand the number of alternating paths that it can add in the Phase-DFS method to the augmentation step to increase the matching each step of the way. To match the first 1/3 vertices,

it calls the Phase-Match procedure. This procedure, iteratively updates weights of free vertices until they have an eligible edge. called aintains these dual weights and generates a directed graph G from the original bipartite graph. For any match collected so far, (a,b), we direct the edge from b to a, for any other edge we direct from a to b. Part of the procedure requires a parameter ω such that $w(M*)/2 \leq \omega \leq w(M*)$, where $M*$ is the optimal matching. This can be achieved by using an n-approximation algorithm to get ω and dividing omega by 2^i, i from 1 to $\log_2$(n) and choosing the least cost matching from the $\log_2$(n) runs.

D. ε-relaxed matchings and strongly/weakly eligible Edges

In the second algorithm, the concept of 1-feasible matchings is extended by an arbitrary free parameter $\varepsilon \in (0, 1)$ to consider ε-relaxed matchings.
A distinction is made between strongly eligible edges and weakly eligible.
Those edges between a $\in$ A, b$\in$ B such that the dual weights satisfy either of the following two conditions are called strongly eligible:
(a,b) $\in$ Matches and y(a) + y(b) == d(a,b) or
(a,b) $\notin$ Matches and y(a) + y(b) == d(a,b) + $\lceil \varepsilon d(a, b) \rceil$
While edges that satisfy the following are called weakly eligible:
(a,b) $\notin$ Matches and d(a,b) $<$ y(a) + y(b) $<$ d(a,b) + $\lceil \varepsilon d(a, b) \rceil$
Using this expanded version of the 1-feasible matching definition allows us to match more of the edges with a better approximation, and allows us to only use the Hungarian algorithm for the last 1/3 of the vertices.

E. Overview of Algorithm 2

Following this expanded definition and through the use of any data structure that allows approximate nearest neighbor queries and deletions to be made in time $\tau(n, \varepsilon)$, the first algorithm is modified to the second algorithm. The changes from the previous algorithm are:
1) The number of stages that need to be taken in each step and the scale that the distance function is scaled by changes from the order of 3^i to the order of 2^i.
2) We now generate the vertex disjoint set of augmenting paths using the ANN data structure and the dual weights to find edges that can be traversed from each free vertex of B, if and only if the vertex b has a strongly eligible edge adjacent to it.
3) Finally, the last change is to maintain the ε-relaxed matching M by augmenting the dual weights for new matches by setting the dual weight of the matching vertex so that y(a) + y(b) = d(a,b).
Except for these differences, the algorithm is the same as before. However the resulting changes significantly affect the theoretical run time of the algorithm.

Algorithm 1: Approximate bipartite matching using 1-feasible matchings

```
Result: Write here the result
Function Phase-Match($G$ = ($A \bigcup B$, $E$), $d(\cdot,\cdot)$, $y(\cdot)$, $k$):
    M = $\emptyset$
    for $k$ steps do
        Generate $P$, set of vertex disjoint paths on eligible edges (determined using $d(\cdot,\cdot)$ and y($\cdot$)) in G,
          starting from all free vertices of B
        for $p \in P$ do
            M = M $\bigoplus$ p, $\bigoplus$ is symmetric difference
        end
        for $b \in$ free vertices of $B$ do
            Create directed graph $\overrightarrow{G}$, (a $\rightarrow$ b) if (a,b) $\in$ M, otherwise (b $\rightarrow$ a)
            for $a \in A$ that is reachable from $b$ do
                y(a) -= 1
            end

            for $b \in B$ that is reachable from $b$ do
                y(b) += 1
            end

        end

        if $|M|$ = $|A|$ then
            return M
        end

    end

    return M

Function Approximate-Bipartite-Match($G$ = ($A \bigcup B$, $E$), $d(\cdot,\cdot)$, $\delta$, $\omega$):
    n = $\|A\|$
    $A_i$ = A
    $B_i$ = B
    M = $\emptyset$
    $\epsilon$ = $\frac{1}{2\log_3(\frac{1}{\delta})}$
    $d_i(\cdot,\cdot)$ = $\lceil \frac{2 \cdot n \cdot d(\cdot,\cdot)}{\epsilon\omega} \rceil$
    while $\|M\| \leq n$ do
        $k$ = $\frac{30}{\epsilon} n^{3^i \delta}$
        if $\|A_i\| \leq n^{\frac{2}{3}}$ then
            M = M $\bigcup$ Hungarian-Algorithm($G = (A_i \bigcup B_i, E)$, $d_i(\cdot,\cdot)$, $y(\cdot)$, $k$)
            return M
        end
        else
            M = M $\bigcup$ Phase-Match(G, $d_i(\cdot,\cdot)$, $y(\cdot)$, $k$)
            $A_{i+1}$ = remaining free vertices of $A_i$
            $B_{i+1}$ = remaining free vertices of $B_i$
            $d_{i+1}(\cdot,\cdot)$ = $\lceil \frac{d_i(\cdot,\cdot)}{2(1+\epsilon)^2 n^{3^{i-1}\delta}} \rceil$
        end

    end

    return M
return
```

Algorithm 2: Approximate bipartite matching using strongly/weakly edges in $\varepsilon-$relaxed matchings

```
Result: Write here the result
Function Phase-Match-With-ANN-DS($G$ = ($A \bigcup B$, $E$), $d(\cdot,\cdot)$, $y(\cdot)$, $k$), $\varepsilon$:
    M = $\emptyset$
    for $k$ steps do
        Construct, $D$, a ANN-DS on the vertices of A using d($\cdot,\cdot$) with accuracy $\varepsilon$
        Generate $P$, set of vertex disjoint paths on strongly/weakly eligible edges (created by using $D$ and
          y($\cdot$) to determine $\varepsilon$-relaxed edges, whenever we visit a $\in$ A, we delete it from $D$) in G, starting
          from every free vertex in B
        for $p \in P$ do
            M = M $\bigoplus$ p, $\bigoplus$ is symmetric difference
            For every new edge (a,b) added to M, set y(b) = d(a,b) - y(a)
        end
        for $b \in$ free vertices of $B$ do
            Create directed graph $\overrightarrow{G}$, (a $\rightarrow$ b) if (a,b) $\in$ M, otherwise (b $\rightarrow$ a)
            for $a \in A$ that is reachable from $b$ do
                y(a) -= 1
            end

            for $b \in B$ that is reachable from $b$ do
                y(b) += 1
            end

        end

        if $|M|$ = $|A|$ then
            return M
        end

    end

    return M

Function Approximate-Bipartite-Match($G$ = ($A \bigcup B$, $E$), $d(\cdot,\cdot)$, $\delta$, $\omega$):
    n = $\|A\|$
    $A_i$ = A
    $B_i$ = B
    M = $\emptyset$
    $\epsilon$ = $\frac{1}{2\log_3(\frac{1}{\delta})}$
    $d_i(\cdot,\cdot)$ = $\lceil \frac{2 \cdot n \cdot d(\cdot,\cdot)}{\epsilon\omega} \rceil$
    while $\|M\| \leq n$ do
        $k$ = $\frac{30}{\epsilon} n^{2^i \delta}$
        if $\|A_i\| \leq n^{\frac{1}{3}}$ then
            M = M $\bigcup$ Hungarian-Algorithm($G = (A_i \bigcup B_i, E)$, $d_i(\cdot,\cdot)$, $y(\cdot)$, $k$)
            return M
        end
        else
            M = M $\bigcup$ Phase-Match-With-ANN-DS(G, $d_i(\cdot,\cdot)$, $y(\cdot)$, $k$)
            $A_{i+1}$ = remaining free vertices of $A_i$
            $B_{i+1}$ = remaining free vertices of $B_i$
            $d_{i+1}(\cdot,\cdot)$ = $\lceil \frac{d_i(\cdot,\cdot)}{2(1+\epsilon)^2 n^{2^{i-1}\delta}} \rceil$
        end

    end

    return M
return
```

F. Comparison algorithm

To compare the algorithms to each other, we will be running the algorithms on a variety of bipartite weighted graph data and comparing the performances of the two algorithms against the Hungarian algorithm as well as an additional algorithm from [I NEED A COMPARISON ALGORITHM]. The performance metrics that we will use will be the runtime of the algorithms and the memory used by the algorithm. [MAYBE NEED TO THINK ABOUT IT]

III. Data

IV. Results

V. Conclusions / Future Work

References

[1] Pankaj K. Agarwal and R. Sharathkumar. 2014. Approximation algorithms for bipartite matching with metric and geometric costs. In Proceedings of the forty-sixth annual ACM symposium on Theory of computing (STOC '14). Association for Computing Machinery, New York, NY, USA, 555–564. DOI:https://doi.org/10.1145/2591796.2591844

[2] H. N. Gabow and R. Tarjan, Faster scaling algorithms for network problems, SIAM J. Comput., 18 (1989), 1013–1036.

[3] R. Sharathkumar and P. K. Agarwal, Algorithms for transportation problem in geometric settings, Proc. 23rd Annual ACM-SIAM Sympos. Discrete Algo., 2012, pp. 306–317.

[4] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM, 34 (1987), 596–615.

[5]

[6] L. Ford and D. Fulkerson. Flows in Networks. Princeton University Press, 1962.

[7] Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM (JACM), 19(2), 248-264.

[8] https://blender.stackexchange.com/a/38123

[9] https://turbosquid.com

Fig. 1. Art data set

```
Final Project/art_comp.png
```

Fig. 2. Synthetic Kitchen

```
Final Project/blender_comp.png
```

Fig. 3. Real data

alg.	art	books	dolls	laundry	moebius	reindeer
ssd	47.4	56.4	48.0	49.7	61.7	55.1
swap ssd	47.5	56.4	47.8	49.0	61.6	54.6
exp. ssd	43.1	54.0	45.1	52.0	62.7	48.5
ncc	46.3	54.6	48.8	50.3	61.4	54.6
swap ncc	45.0	54.2	48.6	49.9	61.3	54.1
exp. ncc	27.2	45.4	36.1	53.3	61.8	39.3
monocular	30.6	24.2	25.4	34.8	41.9	36.1

Fig. 4. Generated Data

alg.	slam	kitchen	class	building	planet
ssd	63.6	25.5	16.0	57.0	42.4
swap ssd	63.6	25.4	16.0	56.9	42.5
exp. ssd	64.2	24.8	16.0	58.2	42.0
ncc	59.1	25.2	16.1	46.3	40.9
swap ncc	59.1	25.2	16.1	46.3	40.9
exp. ncc	65.1	22.8	16.1	36.8	41.2
monocular	60.0	44.8	30.5	46.5	49.2