Multiplicative Auction Algorithm for Approximate Maximum Weight Bipartite Matching[1]

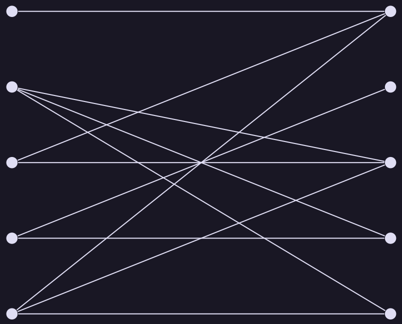
Thorben Tröbst

Theory Seminar, February 10, 2023

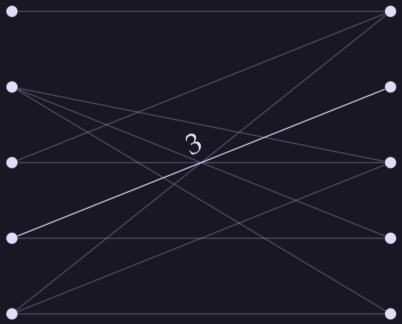
Department of Computer Science, University of California, Irvine

[1] by Da Wei Zheng and Monika Henzinger, to appear in IPCO 2023

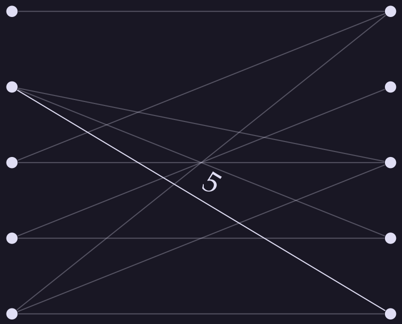
Maximum Weight Bipartite Matching



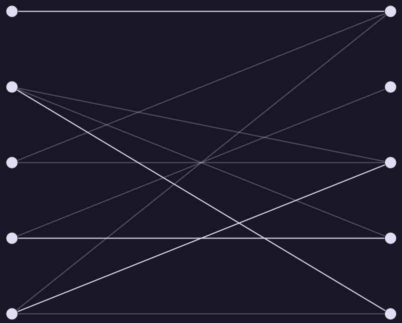
Maximum Weight Bipartite Matching



Maximum Weight Bipartite Matching



Maximum Weight Bipartite Matching



- Classics:

- Classics:
  - Hungarian method $O(n^4)$ or $O(n^3)$ time for Weighted Bipartite Maximum Matching

- Classics:
    - Hungarian method $O(n^4)$ or $O(n^3)$ time for WEIGHTED BIPARTITE MAXIMUM MATCHING
    - Gabow, Tarjan $O(m\sqrt{n}\log(n/\epsilon))$ for WEIGHTED BIPARTITE MAXIMUM MATCHING

- Classics:
  - Hungarian method $O(n^4)$ or $O(n^3)$ time for Weighted Bipartite Maximum Matching
  - Gabow, Tarjan $O(m\sqrt{n}\log(n/\epsilon))$ for Weighted Bipartite Maximum Matching
- More recent, near-linear time:

Algorithms for Matchings and Flows

- Classics:
  - Hungarian method $O(n^4)$ or $O(n^3)$ time for Weighted Bipartite Maximum Matching
  - Gabow, Tarjan $O(m\sqrt{n}\log(n/\epsilon))$ for Weighted Bipartite Maximum Matching
- More recent, near-linear time:
  - Duan, Pettie 2014 $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ for Weighted Maximum Matching

- Classics:
  - Hungarian method $O(n^4)$ or $O(n^3)$ time for Weighted Bipartite Maximum Matching
  - Gabow, Tarjan $O(m\sqrt{n}\log(n/\epsilon))$ for Weighted Bipartite Maximum Matching
- More recent, near-linear time:
  - Duan, Pettie 2014 $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ for Weighted Maximum Matching
  - Chen, Kyng, Liu, Peng, Gutenberg, Sachdeva 2022 $O(m^{1+o(1)})$ for Max Flow

Results

Zheng and Henzinger achieve:

Results

Zheng and Henzinger achieve:

- $(1-\epsilon)$-apx in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ for Weighted Bipartite Maximum Matching with a much simpler algorithm

Results

Zheng and Henzinger achieve:

- $(1-\epsilon)$-apx in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ for Weighted Bipartite Maximum Matching with a much simpler algorithm
- Algorithm is based on multiplicative weights but beats traditional ϵ^{-2} barrier

Results

Zheng and Henzinger achieve:

- $(1-\epsilon)$-apx in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ for Weighted Bipartite Maximum Matching with a much simpler algorithm
- Algorithm is based on multiplicative weights but beats traditional ϵ^{-2} barrier
- Dynamic edge deletions and one-sided vertex insertions in $O(\epsilon^{-1}\log(\epsilon^{-1}))$ time per edge (amortized)

Recall the primal and dual LPs for Bipartite Maximum Weight Matching on $(G \cup B, E)$.

Primal and Dual LP

Recall the primal and dual LPs for Bipartite Maximum Weight Matching on $(G \cup B, E)$.

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e x_e \\ \text{s.t.} \quad & x(\delta(j)) \leq 1 \ \forall j \in G, \\ & x(\delta(i)) \leq 1 \ \forall i \in B, \\ & x \geq 0. \end{aligned}$$

$$\begin{aligned} \min \quad & \sum_{j \in G} p_j + \sum_{i \in B} u_i \\ \text{s.t.} \quad & p_j + u_i \geq w_{ij} \forall \{j, i\} \in E, \\ & p \geq 0, \\ & q \geq 0. \end{aligned}$$

Complementary Slackness

Lemma (Complementary Slackness)

Let x be a matching and p, u dual variables such that:

- *If $p_j > 0$, then j is matched.*
- *If $u_i > 0$, then i is matched.*
- *If i is matched to j then $w_{ij} = p_j + u_i$.*
- *For all $\{i, j\} \in E$, $p_j + u_i \geq w_{ij}$.*

Then x is a maximum weight matching.

Lemma (Complementary Slackness)

Let x be a matching and p, u dual variables such that:

- *If $p_j > 0$, then j is matched.*
- *If $u_i > 0$, then i is matched.*
- *If i is matched to j then $w_{ij} = p_j + u_i$.*
- *For all $\{i, j\} \in E$, $p_j + u_i \geq w_{ij}$.*

Then x is a maximum weight matching.

Proof. Complementary slackness.

Approximate Complementary Slackness

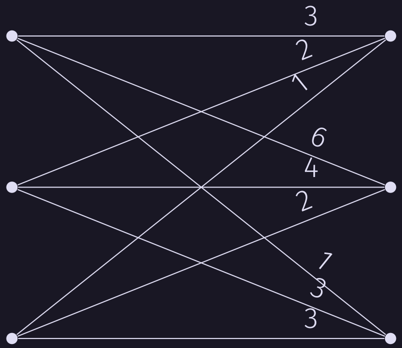
Lemma (Approximate Complementary Slackness)

Let x be a matching and p, q dual variables such that:

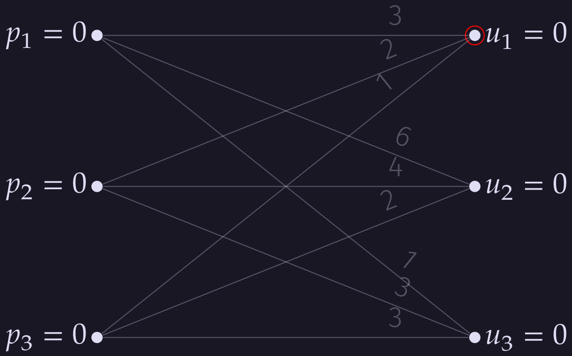
- *If $p_j > 0$, then j is matched.*
- *If $u_i > 0$, then i is matched.*
- *If i is matched to j then $w_{ij} = p_j + u_i$.*
- *For all $\{i, j\} \in E$, $p_j + u_i \geq (1 - \epsilon) w_{ij}$.*

Then x is a $(1 - \epsilon)$-approximate maximum weight matching.

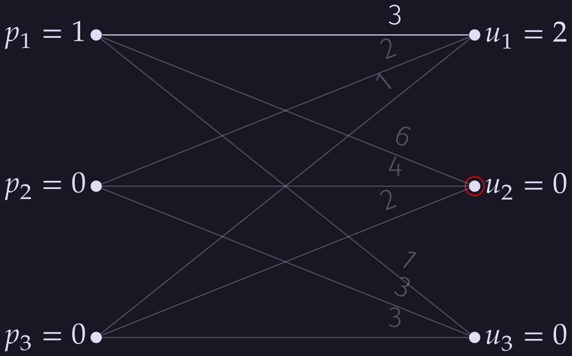
Additive Auction Example



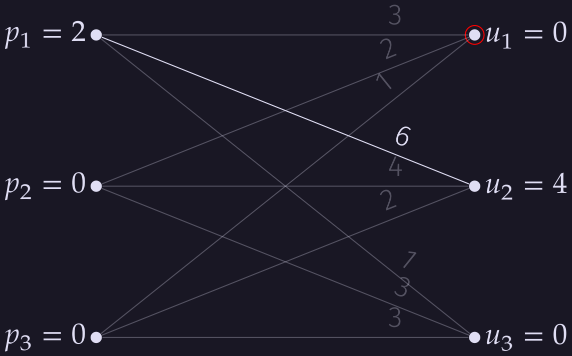
Additive Auction Example



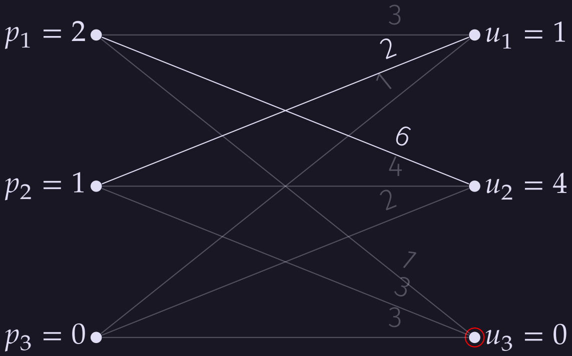
Additive Auction Example



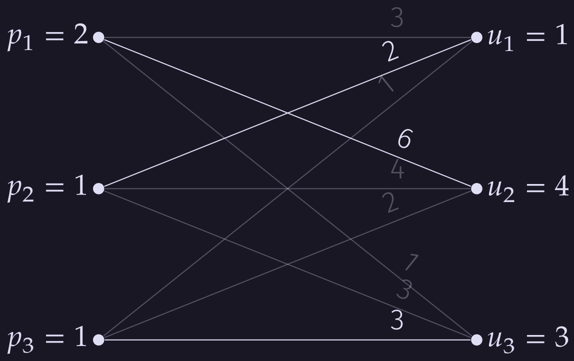
Additive Auction Example



Additive Auction Example



Additive Auction Example



Invariants

The auction algorithm automatically has the following invariants:

INVARIANTS

The auction algorithm automatically has the following invariants:

- If $p_j > 0$, then j is matched.

Invariants

The auction algorithm automatically has the following invariants:

- If $p_j > 0$, then j is matched.
- If $u_i > 0$, then i is matched.

The auction algorithm automatically has the following invariants:

- If $p_j > 0$, then j is matched.
- If $u_i > 0$, then i is matched.
- If $\{i, j\} \in M$, then $p_j + u_i = w_{ij}$.

INVARIANTS

The auction algorithm automatically has the following invariants:

- If $p_j > 0$, then j is matched.
- If $u_i > 0$, then i is matched.
- If $\{i, j\} \in M$, then $p_j + u_i = w_{ij}$.
- If for some i, we have $p_j + u_i \geq (1 - \epsilon)w_{ij}$ for all j at the time that i was matched, then this continues to hold until the match is destroyed.

Goals

We thus have the following goals:

Goals

We thus have the following goals:

- Ensure that $p_j + u_i \geq (1 - \epsilon)w_{ij}$ holds at the time of match.

We thus have the following goals:

- Ensure that $p_j + u_i \geq (1 - \epsilon)w_{ij}$ holds at the time of match.
- Ensure that $p_j \geq (1 - \epsilon)w_{ij}$ holds for all i which are unmatched.

Goals

We thus have the following goals:

- Ensure that $p_j + u_i \geq (1 - \epsilon)w_{ij}$ holds at the time of match.
- Ensure that $p_j \geq (1 - \epsilon)w_{ij}$ holds for all i which are unmatched.
- Ensure that prices rise fast enough to get a good runtime.

Simplifying Assumptions

We can make some simplifying assumptions:

Simplifying Assumptions

We can make some simplifying assumptions:

- $\frac{w_{\max}}{w_{\min}} \leq \frac{n}{\epsilon}$

Simplifying Assumptions

We can make some simplifying assumptions:

- $\frac{w_{\max}}{w_{\min}} \leq \frac{n}{\epsilon}$
- Each w_{ij} is of the form $(1+\epsilon)^{l_{ij}}$ for some $0 \leq l \leq log_{1+\epsilon}(n/\epsilon)$.

Multiplicative Auction

Algorithm 1: Multiplicative Auction

Create a list of pairs Q.

For each $\{i, j\} \in E$, add triples (t, i, j), $(t + 1, i, j)$, ..., (l_{ij}, i, j) to Q where t is maximal such that $(1 + \epsilon)^{l_{ij} - t} > \frac{1}{\epsilon}$.

Sort Q in non-increasing order using bucket sort.

For each i, let $Q_i = \{(k, j) \mid (k, i, j) \in Q\}$.

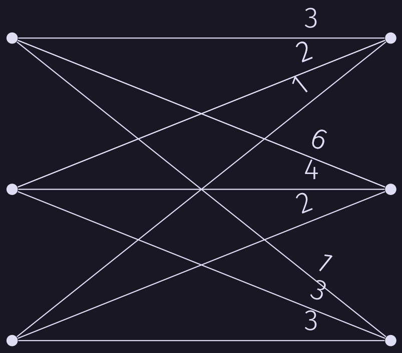
Call $\textsc{Match}(i)$ on unmatched i until the matching stabilizes.

MATCH(i)

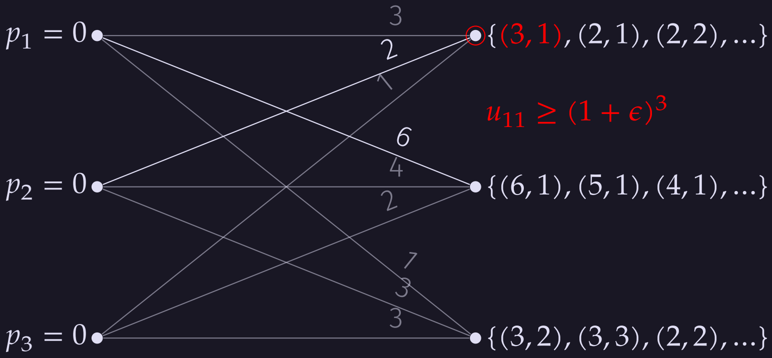
Algorithm 2: MATCH(i)

```
while Q_i is not empty do
    Pop top element (k, j) from Q.
    u_ij := w_ij − p_j
    if u_ij ≥ (1 + ε)^k then
        Match i to j (unmatching previous partner).
        p_j ← p_j + ε u_ij
```

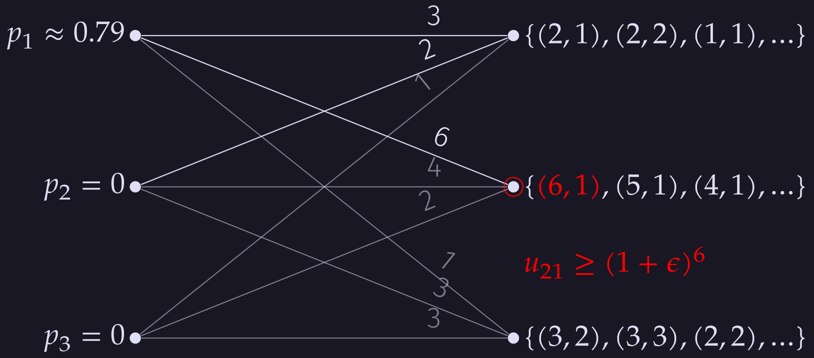
Multiplicative Auction Example $\epsilon = 1/3$



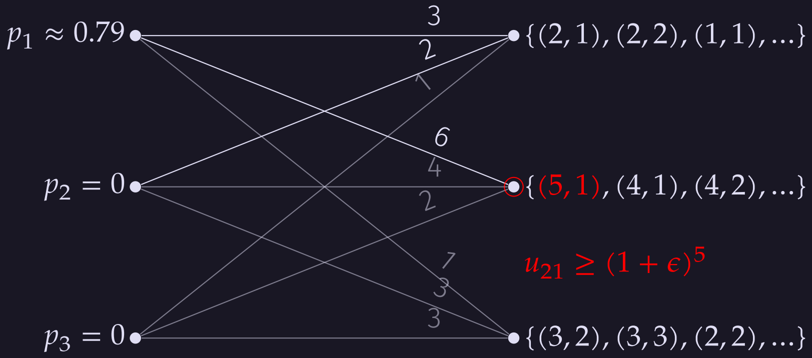
Multiplicative Auction Example $\epsilon = 1/3$



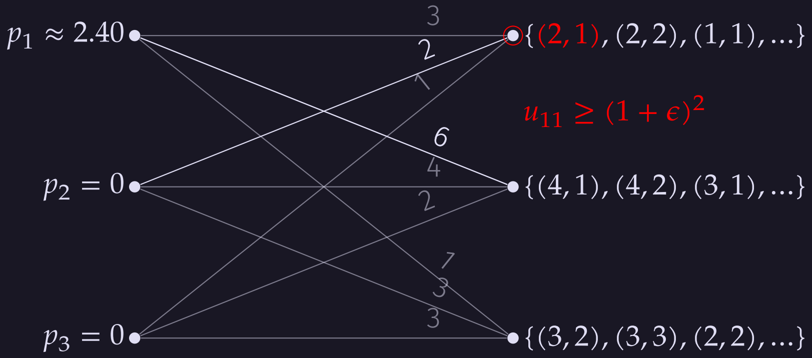
Multiplicative Auction Example $\epsilon = 1/3$



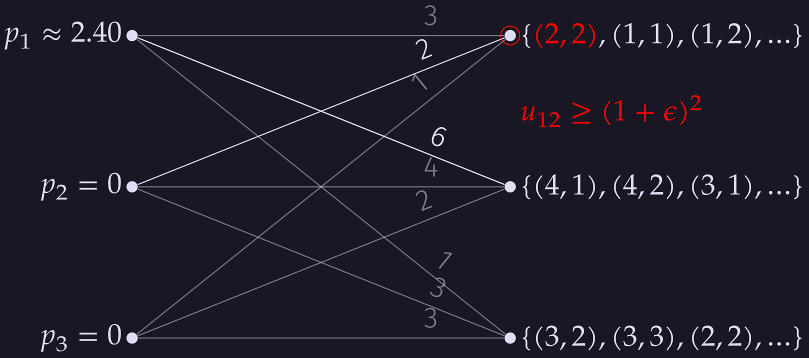
Multiplicative Auction Example $\epsilon = 1/3$



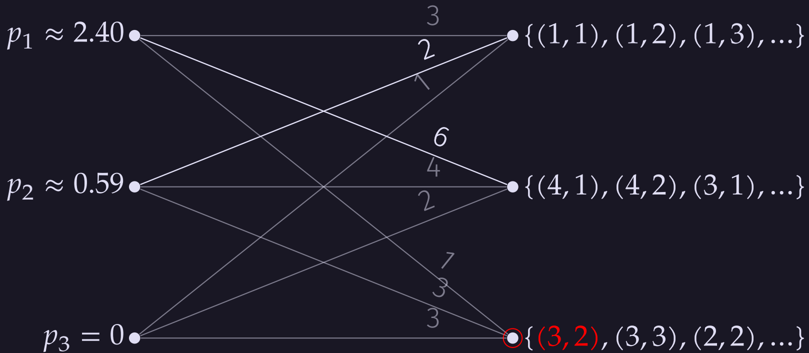
Multiplicative Auction Example $\epsilon = 1/3$



Multiplicative Auction Example $\epsilon = 1/3$

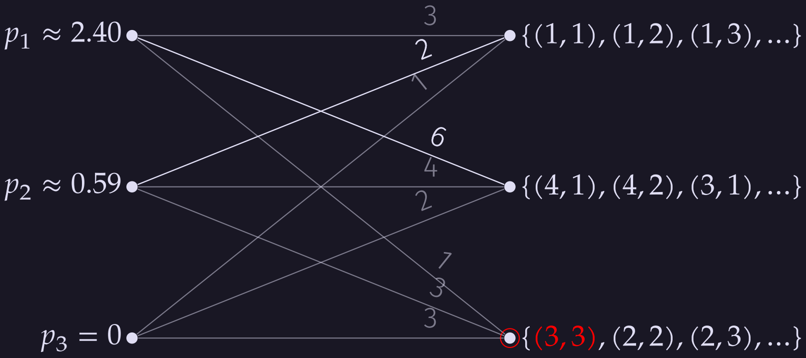


Multiplicative Auction Example $\epsilon = 1/3$



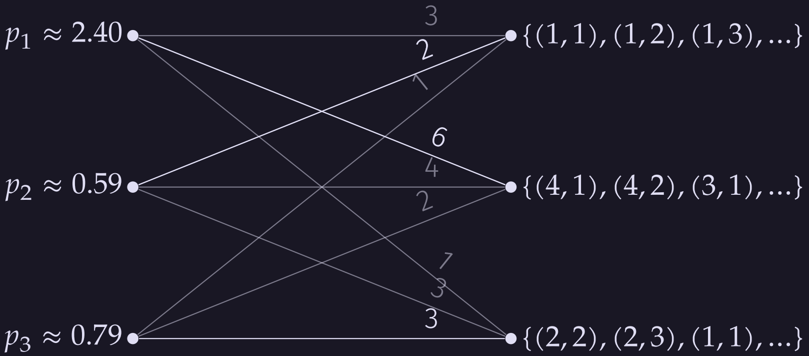
$u_{32} \geq (1+\epsilon)^3$

Multiplicative Auction Example $\epsilon = 1/3$



$u_{33} \geq (1+\epsilon)^3$

Multiplicative Auction Example $\epsilon = 1/3$



Correctness

When (k, j) gets removed from Q_i, we know that $u_{ij} < (1 + \epsilon)^k$ from now on.

Correctness

When (k, j) gets removed from Q_i, we know that $u_{ij} < (1+\epsilon)^k$ from now on. Because if i is matched to j, $u_{ij} \leq (1-\epsilon)(1+\epsilon)^{k+1}$.

Correctness

When (k, j) gets removed from Q_i, we know that $u_{ij} < (1+\epsilon)^k$ from now on. Because if i is matched to j, $u_{ij} \leq (1-\epsilon)(1+\epsilon)^{k+1}$.

Before i gets matched to j, we know $u_{ij} \geq (1+\epsilon)^k$ for some k and $u_{ij'} < (1+\epsilon)^{k+1}$ for all j' because all pairs $(k+1, j')$ have been removed.

Correctness

When (k, j) gets removed from Q_i, we know that $u_{ij} < (1+\epsilon)^k$ from now on. Because if i is matched to j, $u_{ij} \leq (1-\epsilon)(1+\epsilon)^{k+1}$.

Before i gets matched to j, we know $u_{ij} \geq (1+\epsilon)^k$ for some k and $u_{ij'} < (1+\epsilon)^{k+1}$ for all j' because all pairs $(k+1, j')$ have been removed.

So, after matching i to j:

$$\begin{aligned} u_i + p_{j'} = u_{ij} + w_{ij'} - u_{ij'} &\geq \frac{1-\epsilon}{1+\epsilon} u_{ij'} + w_{ij'} - u_{ij'} \\ &\geq (1-2\epsilon)u_{ij'} + w_{ij'} - u_{ij'} = w_{ij'} - 2\epsilon u_{ij'} \\ &\geq (1-2\epsilon) w_{ij'}. \end{aligned}$$

Correctness II

Assume i is unmatched at the end, this means its Q_i is empty.

Correctness II

Assume i is unmatched at the end, this means its Q_i is empty.

So for every j, we know $u_{ij} < \epsilon w_{ij}$ because we removed (t, j) and $(1+\epsilon)^t < \epsilon(1+\epsilon)^{l_{ij}} = \epsilon w_{ij}$.

Correctness II

Assume i is unmatched at the end, this means its Q_i is empty.

So for every j, we know $u_{ij} < \epsilon w_{ij}$ because we removed (t, j) and $(1+\epsilon)^t < \epsilon(1+\epsilon)^{l_{ij}} = \epsilon w_{ij}$.

Thus:

$$u_i + p_j = w_{ij} - u_{ij} \geq (1-\epsilon) w_{ij}. \quad \square$$

Runtime

Runtime is dominated by bucket sort on Q, so we have two questions:

Runtime

Runtime is dominated by bucket sort on Q, so we have two questions:

- How many elements in Q?

Runtime

Runtime is dominated by bucket sort on Q, so we have two questions:

- How many elements in Q?
  - For each edge, we add k elements where k is minimal such that $(1+\epsilon)^{-k} < \epsilon$.

Runtime

Runtime is dominated by bucket sort on Q, so we have two questions:

- How many elements in Q?
  - For each edge, we add k elements where k is minimal such that $(1+\epsilon)^{-k} < \epsilon$.
  - So there are $O(m \log_{1+\epsilon}(\epsilon^{-1})) = O(m\epsilon^{-1} \log(\epsilon^{-1}))$ elements.

Runtime

Runtime is dominated by bucket sort on Q, so we have two questions:

- How many elements in Q?
  - For each edge, we add k elements where k is minimal such that $(1+\epsilon)^{-k} < \epsilon$.
  - So there are $O(m \log_{1+\epsilon}(\epsilon^{-1})) = O(m\epsilon^{-1} \log(\epsilon^{-1}))$ elements.
- How many buckets?

Runtime

Runtime is dominated by bucket sort on Q, so we have two questions:

- How many elements in Q?
  - For each edge, we add k elements where k is minimal such that $(1+\epsilon)^{-k} < \epsilon$.
  - So there are $O(m \log_{1+\epsilon}(\epsilon^{-1})) = O(m\epsilon^{-1}\log(\epsilon^{-1}))$ elements.
- How many buckets?
  - We assume $\frac{w_{\max}}{w_{\min}} \leq \frac{n}{\epsilon}$ and the smallest weight in Q will be $\epsilon w_{\min}$.

Runtime

Runtime is dominated by bucket sort on Q, so we have two questions:

- How many elements in Q?
  - For each edge, we add k elements where k is minimal such that $(1+\epsilon)^{-k} < \epsilon$.
  - So there are $O(m \log_{1+\epsilon}(\epsilon^{-1})) = O(m\epsilon^{-1}\log(\epsilon^{-1}))$ elements.
- How many buckets?
  - We assume $\frac{w_{\max}}{w_{\min}} \leq \frac{n}{\epsilon}$ and the smallest weight in Q will be $\epsilon w_{\min}$.
  - So there are $O(\log_{1+\epsilon}(\frac{n}{\epsilon^2}))$ buckets.

The algorithm can easily be made dynamic by maintaining all Q_i:

The algorithm can easily be made dynamic by maintaining all Q_i:

- Edge deletions

The algorithm can easily be made dynamic by maintaining all Q_i:

- Edge deletions
  - If a non-matching edge was deleted, nothing changes.

The algorithm can easily be made dynamic by maintaining all Q_i:

- Edge deletions
  - If a non-matching edge was deleted, nothing changes.
  - Otherwise continue running $\textsc{Match}(i)$ on the i that was unmatched.

The algorithm can easily be made dynamic by maintaining all Q_i:

- Edge deletions
  - If a non-matching edge was deleted, nothing changes.
  - Otherwise continue running $\textsc{Match}(i)$ on the i that was unmatched.
- Vertex insertions on the buyer side

The algorithm can easily be made dynamic by maintaining all Q_i:

- Edge deletions
    - If a non-matching edge was deleted, nothing changes.
    - Otherwise continue running $\textsc{Match}(i)$ on the i that was unmatched.
- Vertex insertions on the buyer side
    - Create new Q_i for the i that was inserted. Takes $O(\epsilon^{-1}\log(\epsilon^{-1}))$ time per edge if edges are presorted.

The algorithm can easily be made dynamic by maintaining all Q_i:

- Edge deletions
  - If a non-matching edge was deleted, nothing changes.
  - Otherwise continue running $\textsc{Match}(i)$ on the i that was unmatched.
- Vertex insertions on the buyer side
  - Create new Q_i for the i that was inserted. Takes $O(\epsilon^{-1}\log(\epsilon^{-1}))$ time per edge if edges are presorted.
  - Continue running $\textsc{Match}(i)$.

Thank your for listening!