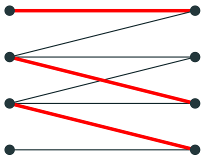
Multiplicative Auction Algorithms

Approximate Maximum Weight Bipartite Matching



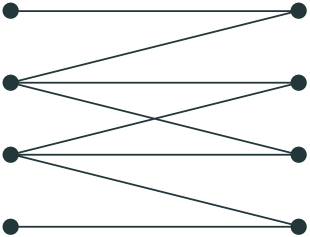
Da Wei Zheng (UIUC) and Monika Henzinger (ISTA)

Sep 13, 2023

Paper presented at IPCO 2023

Matchings in bipartite graphs

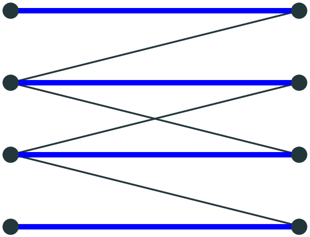
Bipartite graph $G = (U \cup V, E)$
with $n = |U \cup V|$, $m = |E|$.



Matchings in bipartite graphs

Bipartite graph $G = (U \cup V, E)$
with $n = |U \cup V|$, $m = |E|$.

Maximum Cardinality Matching (MCM)



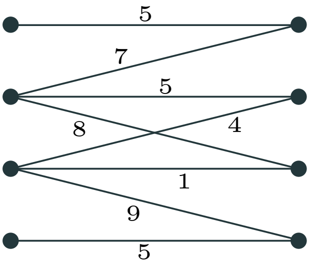
Matchings in bipartite graphs

Bipartite graph $G = (U \cup V, E)$
with $n = |U \cup V|$, $m = |E|$.

Maximum Cardinality Matching (MCM)

Weights $w : E \to \mathbb{R}_{\geq 0}$.

Assume the smallest weight is 1 and the largest is W. Can assume $W = O(n/\varepsilon)$.



Matchings in bipartite graphs

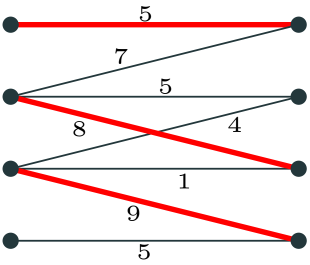
Bipartite graph $G = (U \cup V, E)$
with $n = |U \cup V|$, $m = |E|$.

Maximum Cardinality Matching (MCM)

Weights $w : E \to \mathbb{R}_{\geq 0}$.

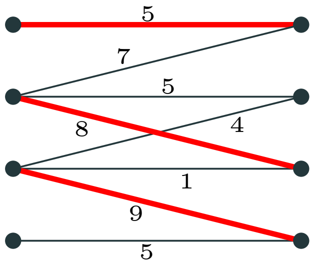
Assume the smallest weight is 1 and the largest is W. Can assume $W = O(n/\varepsilon)$.

Maximum Weight Matching (MWM)



Matchings in bipartite graphs

Bipartite graph $G = (U \cup V, E)$
with $n = |U \cup V|$, $m = |E|$.



Maximum Cardinality Matching (MCM)

Weights $w : E \to \mathbb{R}_{\geq 0}$.

Assume the smallest weight is 1 and the largest is W. Can assume $W = O(n/\varepsilon)$.

Maximum Weight Matching (MWM)

Today: $(1 - \varepsilon)$-approximate maximum weight matching

Goal: Find a matching M such that: $$w(M) \geq (1 - \varepsilon)w(M^*)$$

History of Exact Bipartite MWM Algorithms

Year	Authors	Time bound
1890	Jacobi (written ∼1836)	$\mathrm{poly}(n)$
1946	Easterfield	$2^n \mathrm{poly}(n)$
1953-64	von Neumann, Kuhn, Gleyzal, Munkres, Balinsky–Gomory	$\mathrm{poly}(n)$
1969	Dinic–Kronrod	$O(n^3)$
1970-75	Edmonds–Karp, Tomizawa, Johnson	$\widetilde{O}(mn)$

History of Exact Bipartite MWM Algorithms

Year	Authors	Time bound
1890	Jacobi (written ∼1836)	$\mathrm{poly}(n)$
1946	Easterfield	$2^n \mathrm{poly}(n)$
1953-64	von Neumann, Kuhn, Gleyzal, Munkres, Balinsky–Gomory	$\mathrm{poly}(n)$
1969	Dinic–Kronrod	$O(n^3)$
1970-75	Edmonds–Karp, Tomizawa, Johnson	$\widetilde{O}(mn)$
1983	Gabow	$O(mn^{3/4} \log W)$
1988-97	Gabow–Tarjan, Orlin–Ahuja, Goldberg–Kennedy	$O(m\sqrt{n} \log(nW))$
1996	Cheriyan–Melhorn	$\widetilde{O}(n^{5/2} \log(nW))$
2006	Kao–Lam–Sung–Ting, Sankowski	$O(n^\omega W)$
2012	Duan–Su	$O(m\sqrt{n} \log W)$

History of Exact Bipartite MWM Algorithms

Year	Authors	Time bound
1890	Jacobi (written ∼1836)	$\mathrm{poly}(n)$
1946	Easterfield	$2^n\,\mathrm{poly}(n)$
1953-64	von Neumann, Kuhn, Gleyzal, Munkres, Balinsky–Gomory	$\mathrm{poly}(n)$
1969	Dinic–Kronrod	$O(n^3)$
1970-75	Edmonds–Karp, Tomizawa, Johnson	$\widetilde{O}(mn)$
1983	Gabow	$O(mn^{3/4}\log W)$
1988-97	Gabow–Tarjan, Orlin–Ahuja, Goldberg–Kennedy	$O(m\sqrt{n}\log(nW))$
1996	Cheriyan–Melhorn	$\widetilde{O}(n^{5/2}\log(nW))$
2006	Kao–Lam–Sung–Ting, Sankowski	$O(n^{\omega}W)$
2012	Duan–Su	$O(m\sqrt{n}\log W)$
2020	vd Brand-Lee-Nanogkai-Peng-Saranurak-Sidford-Song-Wang	$\widetilde{O}(m+n^{1.5})$
2022	Chen–Kyng–Liu–Peng–Probst Gutenberg–Sachdeva	$m^{1+o(1)}$

History of Approximate MWM Algorithms

Year	Authors	Approximation	Time bound
-	folklore greedy	1/2	$O(m \log n)$

History of Approximate MWM Algorithms

Year	Authors	Approximation	Time bound
-	folklore greedy	$1/2$	$O(m \log n)$
1988	Gabow–Tarjan	$1-\varepsilon$	$O(m\sqrt{n}\log(n\varepsilon^{-1}))$

History of Approximate MWM Algorithms

Year	Authors	Approximation	Time bound
-	folklore greedy	$1/2$	$O(m \log n)$
1988	Gabow–Tarjan	$1-\varepsilon$	$O(m\sqrt{n} \log(n\varepsilon^{-1}))$
1999/2003	Preis, Drake–Hougardy	$1/2$	$O(m)$
2003	Drake–Hougardy	$2/3-\varepsilon$	$O(m\varepsilon^{-1})$
2004	Pettie–Sanders	$2/3-\varepsilon$	$O(m \log \varepsilon^{-1})$
2010	Duan–Pettie, Hange–Hougardy	$3/4-\varepsilon$	$O(m \log n \log \varepsilon^{-1})$
2014	Duan–Pettie	$1-\varepsilon$	$O(m\varepsilon^{-1} \log \varepsilon^{-1})$

History of Approximate MWM Algorithms

Year	Authors	Approximation	Time bound
-	folklore greedy	$1/2$	$O(m \log n)$
1988	Gabow–Tarjan	$1-\varepsilon$	$O(m\sqrt{n}\log(n\varepsilon^{-1}))$
1999/2003	Preis, Drake–Hougardy	$1/2$	$O(m)$
2003	Drake–Hougardy	$2/3-\varepsilon$	$O(m\varepsilon^{-1})$
2004	Pettie–Sanders	$2/3-\varepsilon$	$O(m \log \varepsilon^{-1})$
2010	Duan–Pettie, Hange–Hougardy	$3/4-\varepsilon$	$O(m \log n \log \varepsilon^{-1})$
2014	Duan–Pettie	$1-\varepsilon$	$O(m\varepsilon^{-1} \log \varepsilon^{-1})$
2023	This talk (Bipartite only)	$1-\varepsilon$	$O(m\varepsilon^{-1})$

History of Dynamic Matchings Algorithms

There is a lot of literature on dynamic matchings.

History of Dynamic Matchings Algorithms

There is a lot of literature on dynamic matchings. **Too much literature...**

History of Dynamic Matchings Algorithms

There is a lot of literature on dynamic matchings. Too much literature...

Variations

- Exact vs approximate (with various ratios 1/2 vs 2/3 vs $(1-\varepsilon)$)
- General graphs vs bipartite graphs
- Maximal matching vs MCM vs MWM
- Fully dynamic vs decremental vs incremental
- Amortized vs average case vs worst case run times

History of Dynamic Matchings Algorithms

There is a lot of literature on dynamic matchings. **Too much literature...**

Variations

- Exact vs approximate (with various ratios 1/2 vs 2/3 vs $(1-\varepsilon)$)
- General graphs vs bipartite graphs
- Maximal matching vs MCM vs MWM
- Fully dynamic vs decremental vs incremental
- Amortized vs average case vs worst case run times

Results

[Wajc '20], [ACCSW '18], [BhaK '21], [PelS '16] [AAGPS '19], [BeFH '19], [ChaS '18], [NeiS '16], [Sank '16], [BhHN '16], [BaGS '11], [BhHN '17], [BhaK '19], [BDHSS '19], [Solo '16], [BhCH '17], [BerS '15], [BerS '16], [Kiss '22], [GLSSS '19], [BehK '22], [BeLM '22], [RoSW '22], [BeRR '22], [GupP '13], ... and many more ...

Our results

1. A simple auction algorithm for $(1-\varepsilon)$-approximate MWM.
2. Efficient dynamic algorithm, supporting one-sided vertex deletion, and other-sided vertex insertion (simultaneously).

Multiplicative Auction Algorithm

The auction algorithm of Bertsekas ’81 and Demange–Gale–Sotomayor ’86

The auction algorithm of Bertsekas '81 and Demange–Gale–Sotomayor '86

While $\exists v \in V$ unallocated, $\textbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Left: Items $u \in U$

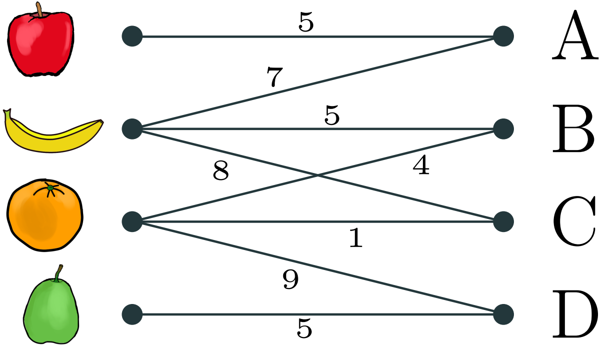
Price y_u initially 0

Utility of v having u:

$\textbf{util}(uv) = w(uv) - y_u$

Right: Bidders $v \in V$

Initially unallocated



The auction algorithm of Bertsekas ’81 and Demange–Gale–Sotomayor ’86

While $\exists v \in V$ unallocated, $\mathbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Left: Items $u \in U$

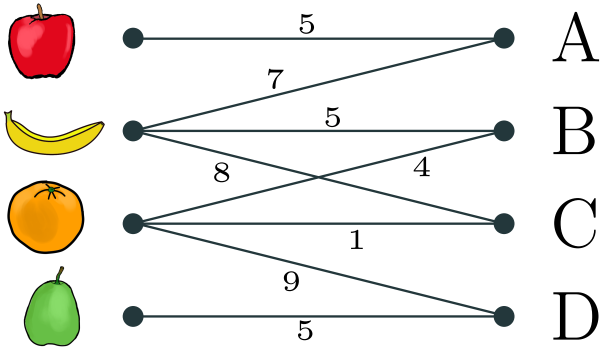
Price y_u initially 0

Utility of v having u:

$\mathbf{util}(uv) = w(uv) - y_u$

Right: Bidders $v \in V$

Initially unallocated



The auction algorithm of Bertsekas '81 and Demange–Gale–Sotomayor '86

While $\exists v \in V$ unallocated, $\textbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Left: Items $u \in U$

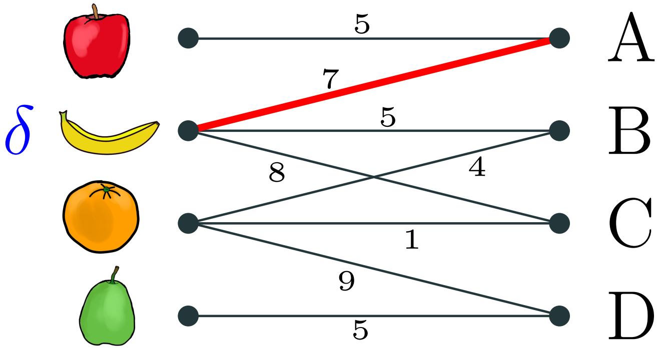
Price y_u initially 0

Utility of v having u:

$\textbf{util}(uv) = w(uv) - y_u$

Right: Bidders $v \in V$

Initially unallocated



The auction algorithm of Bertsekas '81 and Demange–Gale–Sotomayor '86

While $\exists v \in V$ unallocated, $\mathbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Left: Items $u \in U$

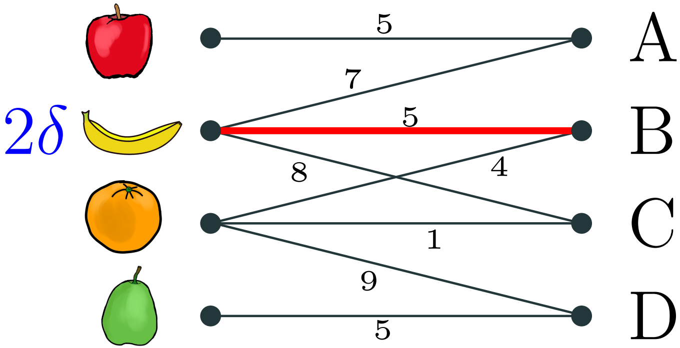
Price y_u initially 0

Utility of v having u:

$\mathbf{util}(uv) = w(uv) - y_u$

Right: Bidders $v \in V$

Initially unallocated



The auction algorithm of Bertsekas '81 and Demange–Gale–Sotomayor '86

While $\exists v \in V$ unallocated, $\mathbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Left: Items $u \in U$

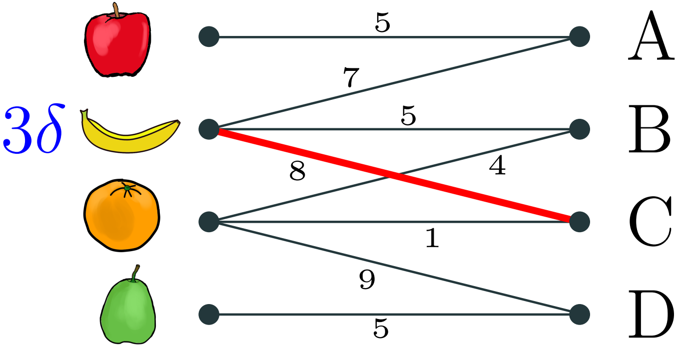
Price y_u initially 0

Utility of v **having** u**:**

$\mathbf{util}(uv) = w(uv) - y_u$

Right: Bidders $v \in V$

Initially unallocated



The auction algorithm of Bertsekas '81 and Demange–Gale–Sotomayor '86

While $\exists v \in V$ unallocated, $\mathbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Left: Items $u \in U$

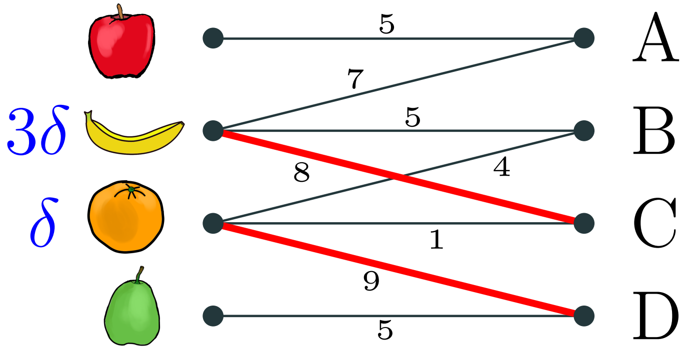
Price y_u initially 0

Utility of v having u:

$\mathbf{util}(uv) = w(uv) - y_u$

Right: Bidders $v \in V$

Initially unallocated



While $\exists v \in V$ unallocated, $\mathbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Left: Items $u \in U$

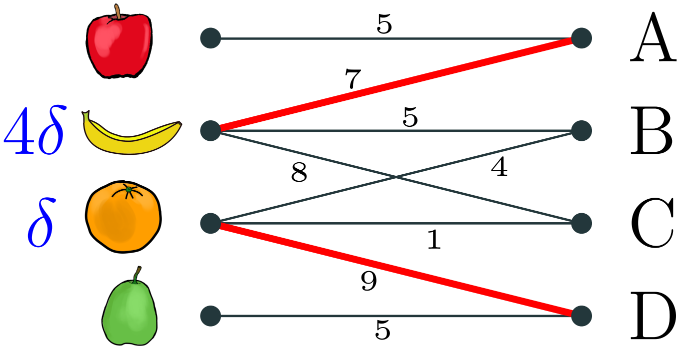
Price y_u initially 0

Utility of v having u:

$\mathbf{util}(uv) = w(uv) - y_u$

Right: Bidders $v \in V$

Initially unallocated



Original Auction Algorithm

While $\exists v \in V$ unallocated, $\max_u \text{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Original Auction Algorithm

While $\exists v \in V$ unallocated, $\max_u \textbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Can be implemented in $O(m\delta^{-1}W)$ time, gets additive error of $n\delta$.

New algorithm

Original Auction Algorithm

While $\exists v \in V$ unallocated, $\max_u \mathbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Can be implemented in $O(m\delta^{-1}W)$ time, gets additive error of $n\delta$.

New algorithm

Original Auction Algorithm

While $\exists v \in V$ unallocated, $\max_u \textbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

Can be implemented in $O(m\delta^{-1}W)$ time, gets additive error of $n\delta$.

Multiplicative Auction Algorithm (NEW!)

While $\exists v \in V$ unallocated, $\textbf{util}(uv) > \varepsilon \cdot w(uv)$, v bids $y_u + \varepsilon \cdot w(uv)$ and allocated max util u.

New algorithm

Original Auction Algorithm

While $\exists v \in V$ unallocated, $\max_u \mathbf{util}(uv) > 0$, v bids $y_u + \delta$ and allocated max util u.

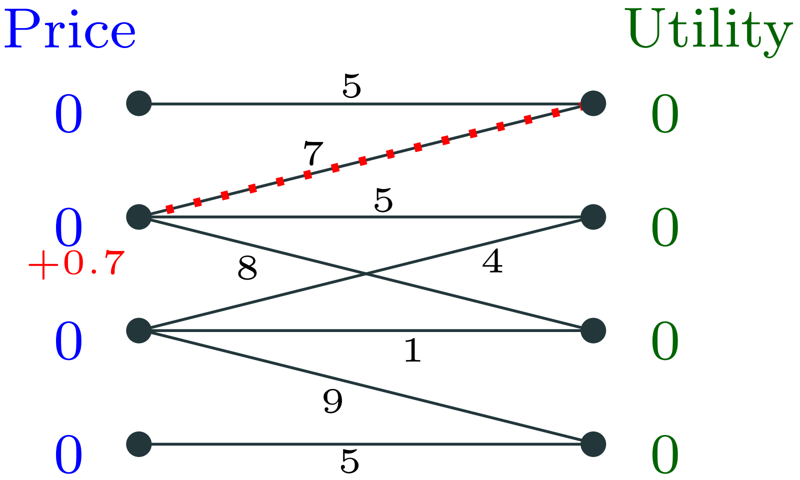
Can be implemented in $O(m\delta^{-1}W)$ time, gets additive error of $n\delta$.

Multiplicative Auction Algorithm (NEW!)

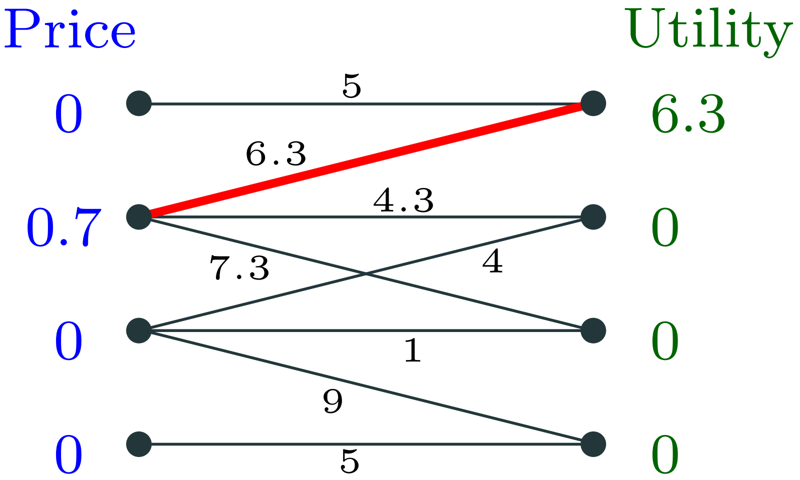
While $\exists v \in V$ unallocated, $\mathbf{util}(uv) > \varepsilon \cdot w(uv)$, v bids $y_u + \varepsilon \cdot w(uv)$ and allocated max util u.

Can be implemented in time $O(m\varepsilon^{-1})$, gets multiplicative error of $(1 - \varepsilon)$.

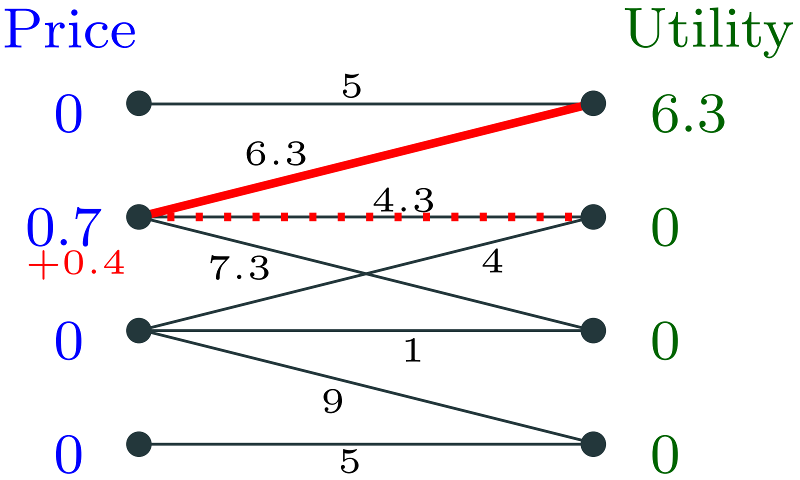
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



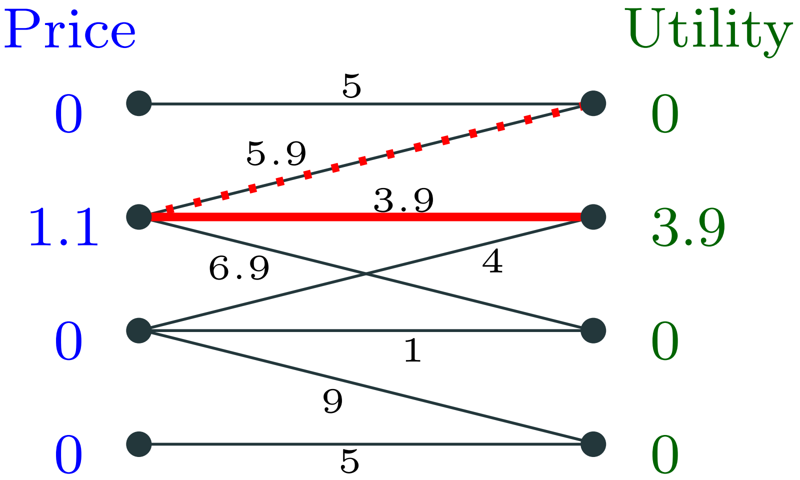
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



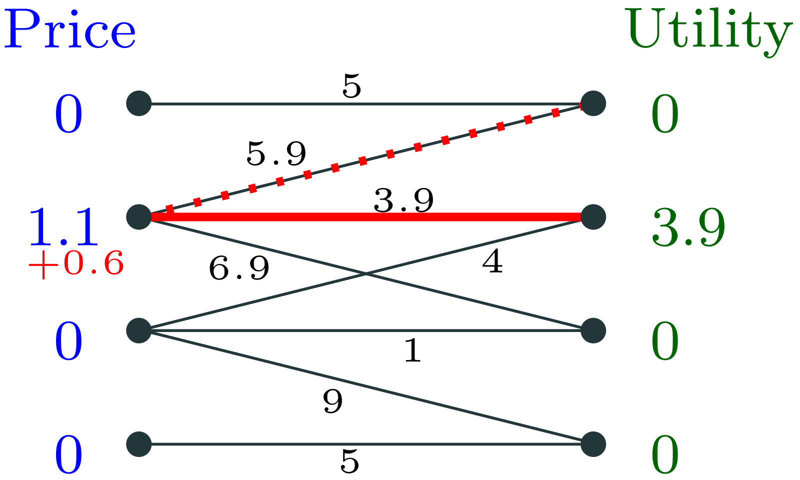
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



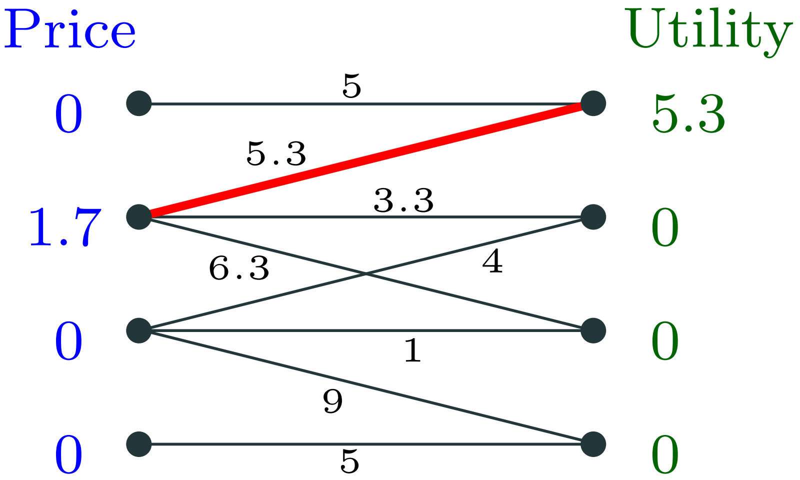
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



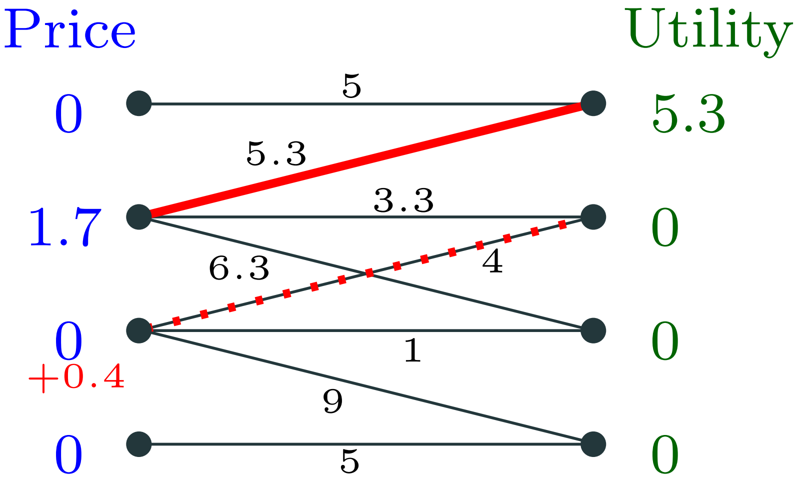
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



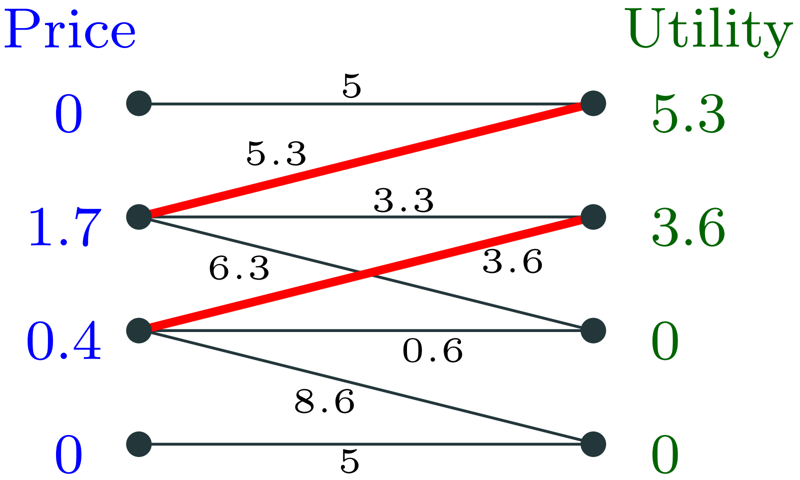
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



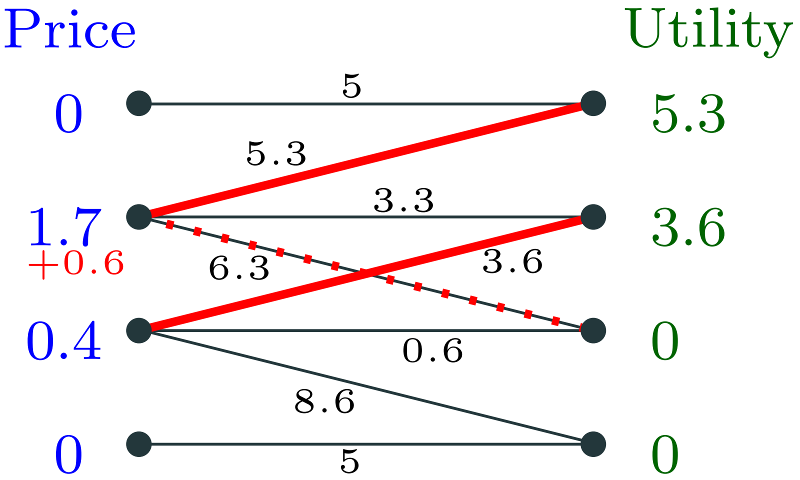
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



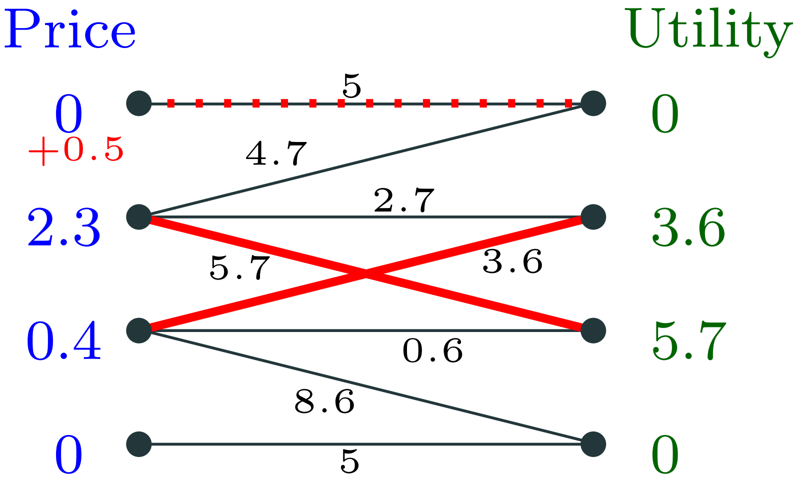
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



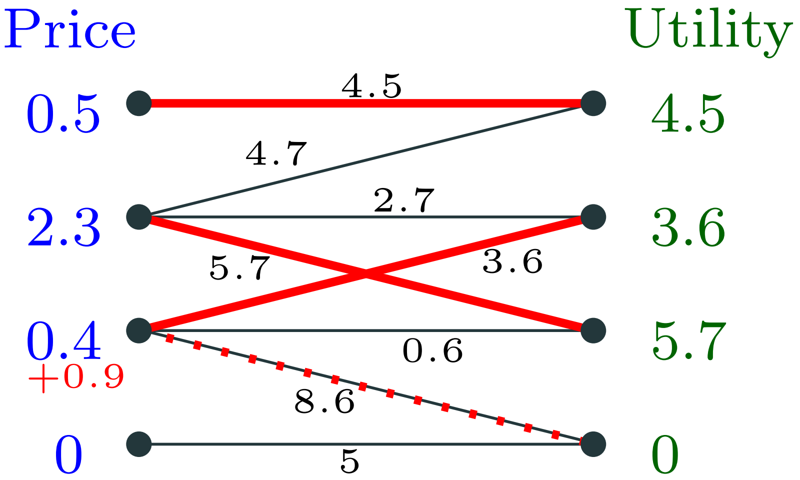
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



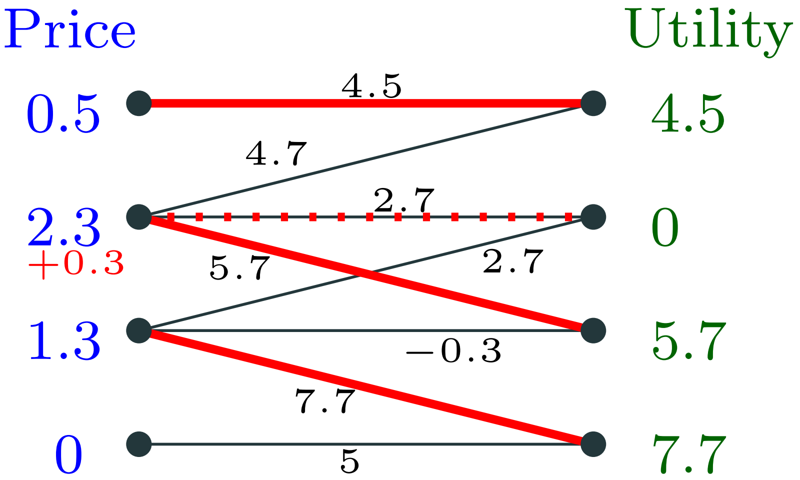
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



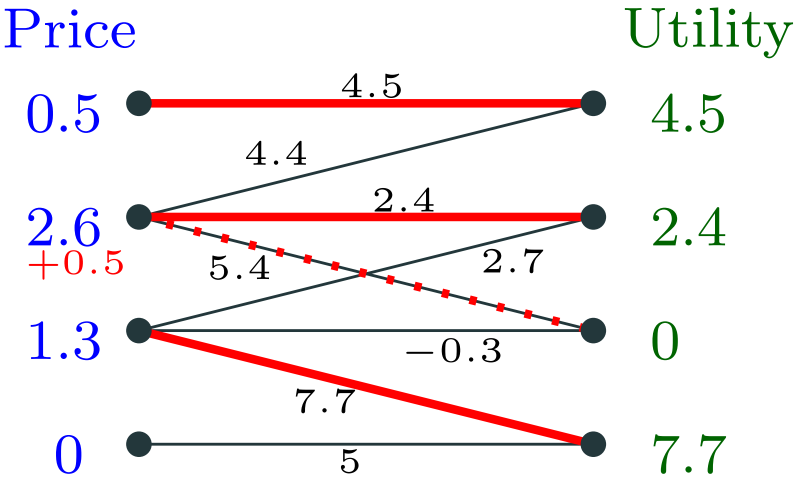
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



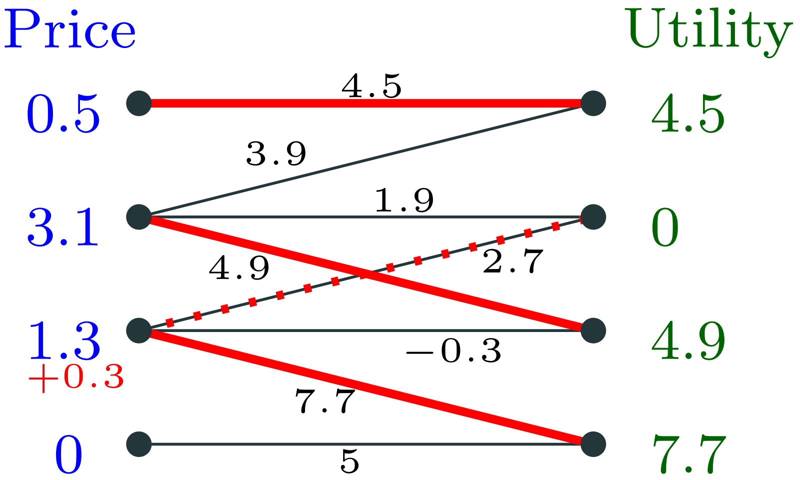
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



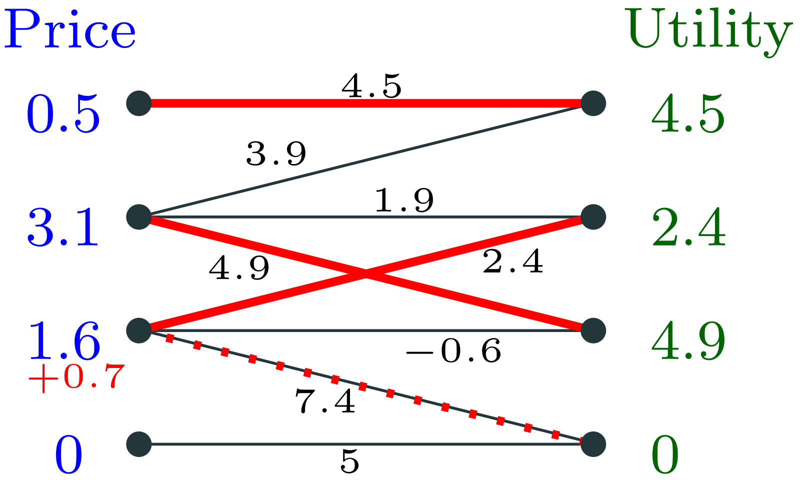
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



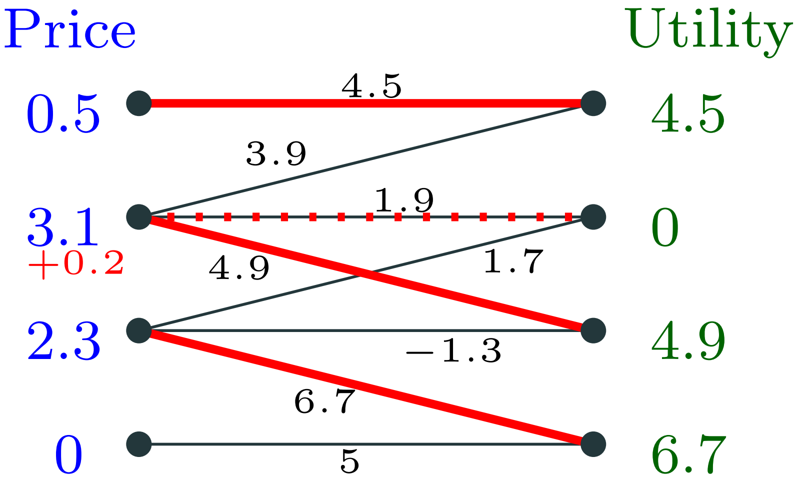
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



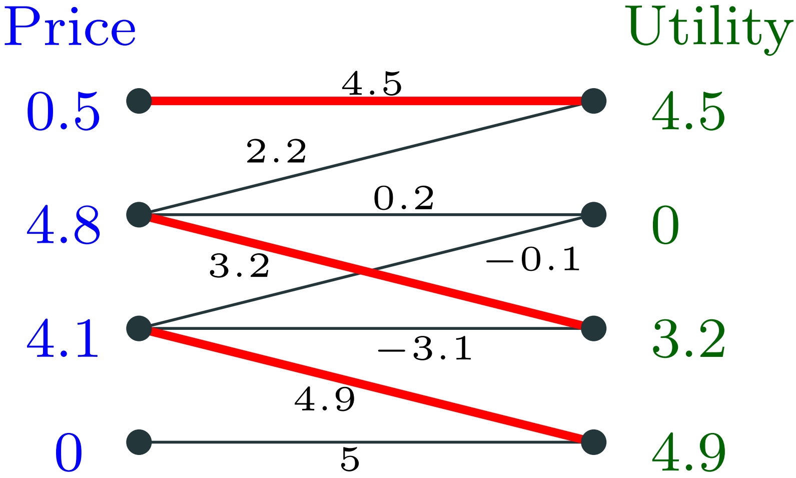
Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



Example of the multiplicative auction algorithm with $\varepsilon = 0.1$

$\cdots$

Example of the multiplicative auction algorithm with $\varepsilon = 0.1$



Implementation and runtime of the algorithm

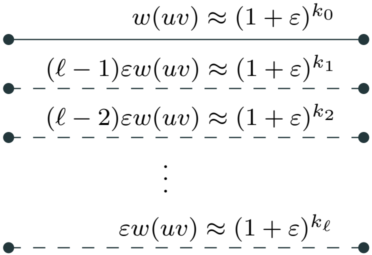
Implementation details

1. Round all edges to powers of $(1+\varepsilon)$, i.e. $(1+\varepsilon)^0, (1+\varepsilon)^1, (1+\varepsilon)^2 \ldots$

Implementation details

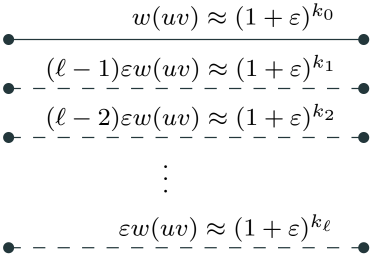
1. Round all edges to powers of $(1+\varepsilon)$, i.e. $(1+\varepsilon)^0, (1+\varepsilon)^1, (1+\varepsilon)^2 \ldots$
2. For each edge uv we only need to consider them at weights $i\varepsilon w(uv)$ for $i = 1, \ldots, \ell$ where $\ell = 1/\varepsilon$.
   We can also round these to powers of $(1+\varepsilon)$.

Implementation details



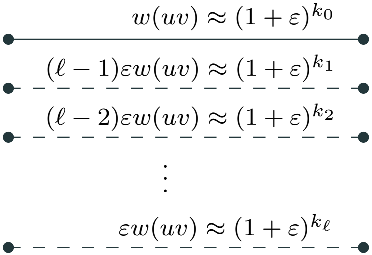
1. Round all edges to powers of $(1+\varepsilon)$, i.e. $(1+\varepsilon)^0, (1+\varepsilon)^1, (1+\varepsilon)^2 \ldots$
2. For each edge uv we only need to consider them at weights $i\varepsilon w(uv)$ for $i = 1, \ldots, \ell$ where $\ell = 1/\varepsilon$.
   We can also round these to powers of $(1+\varepsilon)$.
3. $\forall v \in V$ store "copies" of an edge in a (priority) queue after doing an initial sort.

Implementation details



1. Round all edges to powers of $(1+\varepsilon)$, i.e. $(1+\varepsilon)^0, (1+\varepsilon)^1, (1+\varepsilon)^2 \ldots$
2. For each edge uv we only need to consider them at weights $i\varepsilon w(uv)$ for $i = 1, \ldots, \ell$ where $\ell = 1/\varepsilon$.
   We can also round these to powers of $(1+\varepsilon)$.
3. $\forall v \in V$ store "copies" of an edge in a (priority) queue after doing an initial sort.
4. Run the multiplicative auction algorithm by checking edges in (priority) queue order of decreasing weight.

Implementation details



1. Round all edges to powers of $(1+\varepsilon)$, i.e. $(1+\varepsilon)^0, (1+\varepsilon)^1, (1+\varepsilon)^2 \ldots$
2. For each edge uv we only need to consider them at weights $i\varepsilon w(uv)$ for $i = 1, \ldots, \ell$ where $\ell = 1/\varepsilon$.
   We can also round these to powers of $(1+\varepsilon)$.
3. $\forall v \in V$ store "copies" of an edge in a (priority) queue after doing an initial sort.
4. Run the multiplicative auction algorithm by checking edges in (priority) queue order of decreasing weight.

$O(m\varepsilon^{-1})$ to sort integers in $[0, \varepsilon^{-1} \log n]$, and $O(m\varepsilon^{-1})$ for the algorithm.

Dynamic algorithm details

Deleting a vertex $u \in U$

Dynamic algorithm details

Deleting a vertex $u \in U$

If there is a $v \in V$ that was matched to u, v becomes unmatched after deletion.

Dynamic algorithm details

Deleting a vertex $u \in U$

If there is a $v \in V$ that was matched to u, v becomes unmatched after deletion.

Treat v as unallocated and continue running multiplicative auction algorithm.

Dynamic algorithm details

Deleting a vertex $u \in U$

If there is a $v \in V$ that was matched to u, v becomes unmatched after deletion.

Treat v as unallocated and continue running multiplicative auction algorithm.

Adding a new vertex $v \in V$ along with incident edges

Treat v as unallocated and run multiplicative auction algorithm.

Correctness of the algorithm

LP for MWM

Variables x_{uv} for each edge $uv \in E$.

$$
\begin{aligned}
\max \quad & \sum_{uv \in E} w(uv) x_{uv} & \\
s.t. \quad & \sum_{v \in N(u)} x_{uv} \leq 1 & \forall u \in U \\
& \sum_{u \in N(v)} x_{uv} \leq 1 & \forall v \in V \\
& x_{uv} \geq 0 & \forall uv \in E
\end{aligned}
$$

LP for MWM

Variables x_{uv} for each edge $uv \in E$.

$$
\begin{aligned}
\max \quad & \sum_{uv \in E} w(uv) x_{uv} & \\
s.t. \quad & \sum_{v \in N(u)} x_{uv} \leq 1 & \forall u \in U \\
& \sum_{u \in N(v)} x_{uv} \leq 1 & \forall v \in V \\
& x_{uv} \geq 0 & \forall uv \in E
\end{aligned}
$$

Variables y_u for $u \in U$, y_v for $v \in V$.

$$
\begin{aligned}
\min \quad & \sum_{u \in U} y_u + \sum_{v \in V} y_v & \\
s.t. \quad & y_u + y_v \geq w(uv) & \forall uv \in E \\
& y_u \geq 0 & \forall u \in U \\
& y_v \geq 0 & \forall v \in V
\end{aligned}
$$

LP for MWM

Variables x_{uv} for each edge $uv \in E$.

$$\begin{aligned} \max \quad & \sum_{uv \in E} w(uv) x_{uv} & \\ s.t. \quad & \sum_{v \in N(u)} x_{uv} \leq 1 & \forall u \in U \\ & \sum_{u \in N(v)} x_{uv} \leq 1 & \forall v \in V \\ & x_{uv} \geq 0 & \forall uv \in E \end{aligned}$$

Variables y_u for $u \in U$, y_v for $v \in V$.

$$\begin{aligned} \min \quad & \sum_{u \in U} y_u + \sum_{v \in V} y_v & \\ s.t. \quad & y_u + y_v \geq w(uv) & \forall uv \in E \\ & y_u \geq 0 & \forall u \in U \\ & y_v \geq 0 & \forall v \in V \end{aligned}$$

Approximate dominance: $[y_u + y_v \geq (1 - \varepsilon_1) \cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

LP for MWM

Variables x_{uv} for each edge $uv \in E$.

$$\begin{aligned} \max \quad & \sum_{uv \in E} w(uv) x_{uv} & \\ s.t. \quad & \sum_{v \in N(u)} x_{uv} \le 1 & \forall u \in U \\ & \sum_{u \in N(v)} x_{uv} \le 1 & \forall v \in V \\ & x_{uv} \ge 0 & \forall uv \in E \end{aligned}$$

Variables y_u for $u \in U$, y_v for $v \in V$.

$$\begin{aligned} \min \quad & \sum_{u \in U} y_u + \sum_{v \in V} y_v & \\ s.t. \quad & y_u + y_v \ge w(uv) & \forall uv \in E \\ & y_u \ge 0 & \forall u \in U \\ & y_v \ge 0 & \forall v \in V \end{aligned}$$

Approximate dominance: $[y_u + y_v \ge (1 - \varepsilon_1) \cdot w(uv) \ \forall uv \in E]$ & $[y_z \ge 0 \ \forall z]$.

Approx. comp. slackness: $[y_u + y_v \le (1 + \varepsilon_0) \cdot w(uv) \ \forall uv \in M]$ & $[y_z = 0 \ \forall z \notin M\ ]$.

Comp. Slackness + Approx. dominance = Approx. Optimality

Approximate dominance: $[y_u + y_v \geq (1-\varepsilon_1)\cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

Approx. comp. slackness: $[y_u + y_v \leq (1+\varepsilon_0)\cdot w(uv)\ \forall uv \in M]$ & $[y_z = 0\ \forall z \notin M]$.

Comp. Slackness + Approx. dominance = Approx. Optimality

Approximate dominance: $[y_u + y_v \geq (1-\varepsilon_1) \cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

Approx. comp. slackness: $[y_u + y_v \leq (1+\varepsilon_0) \cdot w(uv)\ \forall uv \in M]$ & $[y_z = 0\ \forall z \notin M]$.

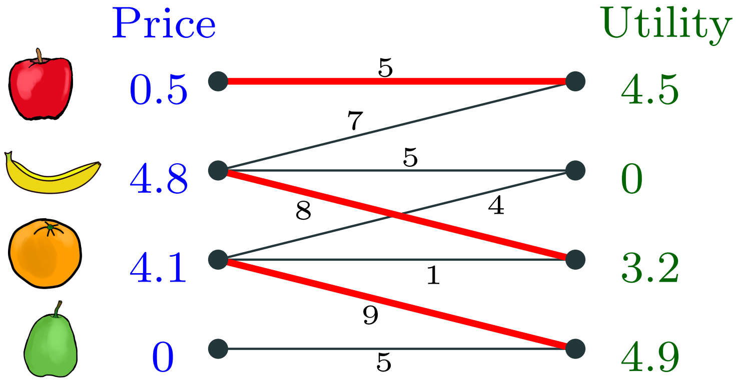
Let M^* be the maximum weight matching.

$$\begin{aligned}
w(M) &= \sum_{uv \in M} w(uv) \\
&\geq \sum_{uv \in M} (1+\varepsilon_0)^{-1} \cdot (y_u + y_v) && \text{Approx. comp. slackness} \\
&= (1+\varepsilon_0)^{-1} \sum_{z \in U \cup V} y_z && \text{Complementarity} \\
&\geq (1+\varepsilon_0)^{-1} \sum_{uv \in M^*} (y_u + y_v) && \text{Non-negativity of } y_z \\
&\geq (1+\varepsilon_0)^{-1} (1-\varepsilon_1) \cdot w(M^*) && \text{Approximate dominance}
\end{aligned}$$

Comp. Slackness + Approx. Dominance(example with $\varepsilon = 0.1$)

Approximate dominance: $[y_u + y_v \geq (1 - \varepsilon_1) \cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

Approx. comp. slackness: $[y_u + y_v \leq (1 + \varepsilon_0) \cdot w(uv)\ \forall uv \in M]$ & $[y_z = 0\ \forall z \notin M\ ]$.

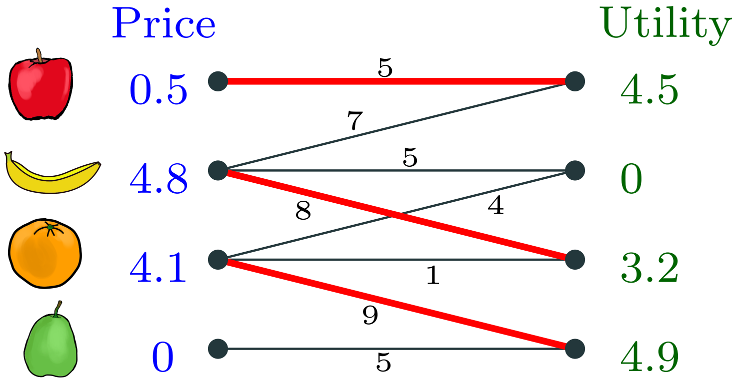


Comp. Slackness + Approx. Dominance(example with $\varepsilon = 0.1$)

Approximate dominance: $[y_u + y_v \geq (1-\varepsilon_1)\cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

Approx. comp. slackness: $[y_u + y_v \leq (1+\varepsilon_0)\cdot w(uv)\ \forall uv \in M]$ & $[y_z = 0\ \forall z \notin M\ ]$.

Consider edge $uv \in E$:



Comp. Slackness + Approx. Dominance(example with $\varepsilon = 0.1$)

Approximate dominance: $[y_u + y_v \geq (1 - \varepsilon_1) \cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

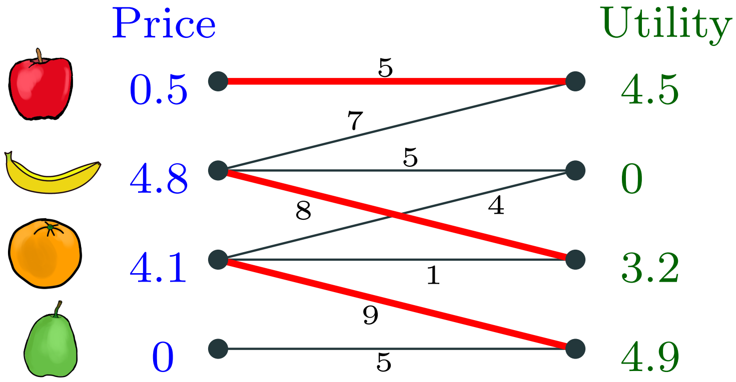
Approx. comp. slackness: $[y_u + y_v \leq (1 + \varepsilon_0) \cdot w(uv)\ \forall uv \in M]$ & $[y_z = 0\ \forall z \notin M\ ]$.

Consider edge $uv \in E$:

Case 1:

uv is in the matching.

$$y_u + \textbf{util}(uv) = w(uv).$$



Comp. Slackness + Approx. Dominance(example with $\varepsilon = 0.1$)

Approximate dominance: $[y_u + y_v \geq (1 - \varepsilon_1) \cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

Approx. comp. slackness: $[y_u + y_v \leq (1 + \varepsilon_0) \cdot w(uv)\ \forall uv \in M]$ & $[y_z = 0\ \forall z \notin M\ ]$.

Consider edge $uv \in E$:

Case 1:

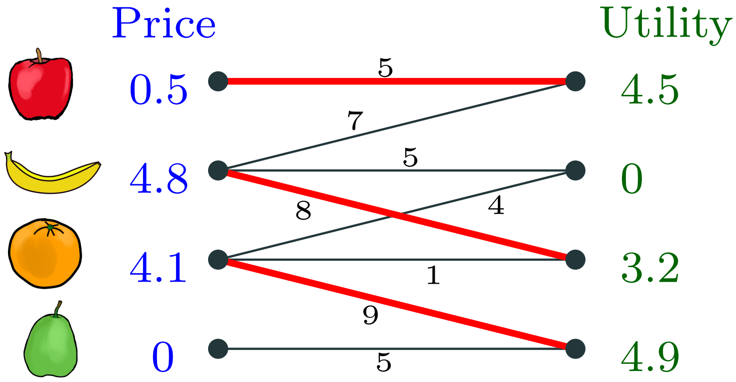
uv is in the matching.

$$y_u + \textbf{util}(uv) = w(uv).$$

Case 2:

$v \in V$ is matched.

v preferred another item.



Comp. Slackness + Approx. Dominance(example with $\varepsilon = 0.1$)

Approximate dominance: $[y_u + y_v \geq (1-\varepsilon_1)\cdot w(uv)\ \forall uv \in E]$ & $[y_z \geq 0\ \forall z]$.

Approx. comp. slackness: $[y_u + y_v \leq (1+\varepsilon_0)\cdot w(uv)\ \forall uv \in M]$ & $[y_z = 0\ \forall z \notin M]$.

Consider edge $uv \in E$:

Case 1:

uv is in the matching.

$$y_u + \textbf{util}(uv) = w(uv).$$

Case 2:

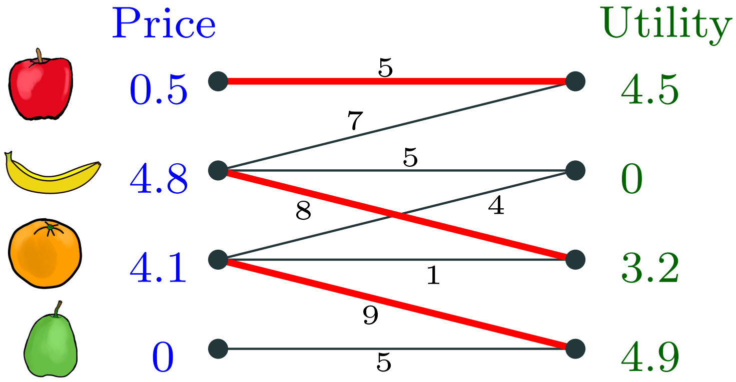
$v \in V$ is matched.

v preferred another item.

Case 3:

$v \in V$ is unmatched.

All items have high price.



Conclusion

- We present a much simpler algorithm for approximate MWM.

Conclusion

- We present a much simpler algorithm for approximate MWM.
- The algorithm generalizes easily to (vertex) dynamic settings.

Conclusion

- We present a much simpler algorithm for approximate MWM.
- The algorithm generalizes easily to (vertex) dynamic settings.

Open questions

Conclusion

- We present a much simpler algorithm for approximate MWM.
- The algorithm generalizes easily to (vertex) dynamic settings.

Open questions

1. Relation to multiplicative weight update (MWU) and local ratio algorithms?

Conclusion

- We present a much simpler algorithm for approximate MWM.
- The algorithm generalizes easily to (vertex) dynamic settings.

Open questions

1. Relation to multiplicative weight update (MWU) and local ratio algorithms?
2. Edge insertions / deletions? Fully dynamic?

Conclusion

- We present a much simpler algorithm for approximate MWM.
- The algorithm generalizes easily to (vertex) dynamic settings.

Open questions

1. Relation to multiplicative weight update (MWU) and local ratio algorithms?
2. Edge insertions / deletions? Fully dynamic?
3. Simple parallel / distributed / streaming algorithms?

Conclusion

- We present a much simpler algorithm for approximate MWM.
- The algorithm generalizes easily to (vertex) dynamic settings.

Open questions

1. Relation to multiplicative weight update (MWU) and local ratio algorithms?
2. Edge insertions / deletions? Fully dynamic?
3. Simple parallel / distributed / streaming algorithms?
4. General graphs?

Conclusion

- We present a much simpler algorithm for approximate MWM.
- The algorithm generalizes easily to (vertex) dynamic settings.

Open questions

1. Relation to multiplicative weight update (MWU) and local ratio algorithms?
2. Edge insertions / deletions? Fully dynamic?
3. Simple parallel / distributed / streaming algorithms?
4. General graphs?
5. (Decremental / incremental) $(1-\varepsilon)$-approximate SSSP / transshipment?