Mechanisms for matching markets on ride-sharing

Jiawei Zhang, Ruofan Yu
2022/11/15

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN


Uber

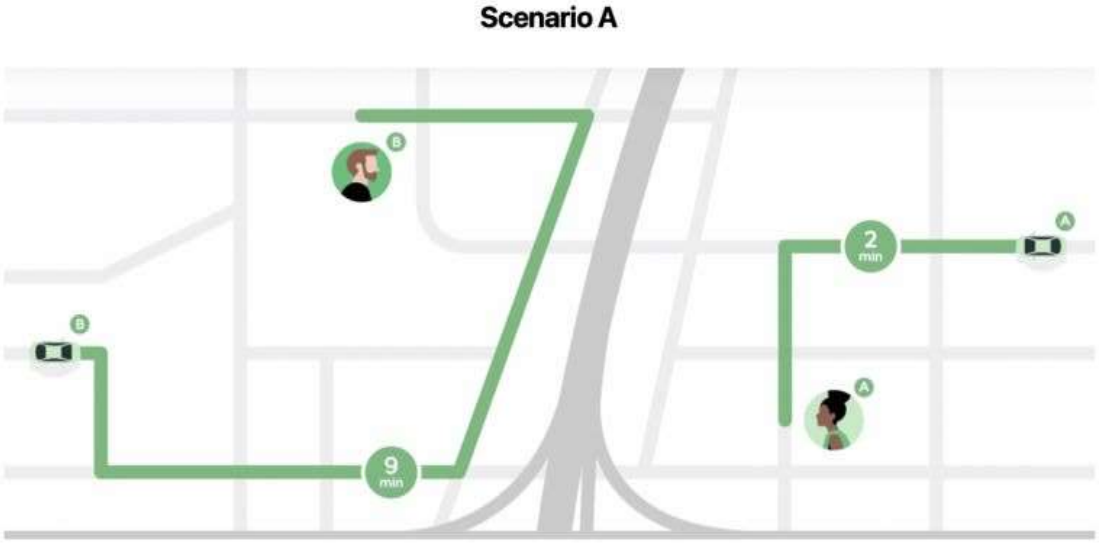
Question: How does Uber do the matching between the drivers and riders?

Question: How does Uber do the matching between the drivers and riders?

- "As-the-crow-flies":
  Match based on the ***shortest distance*** between drivers and riders

Question: How does Uber do the matching between the drivers and riders?

- "As-the-crow-flies":
  Match based on the ***shortest distance*** between drivers and riders

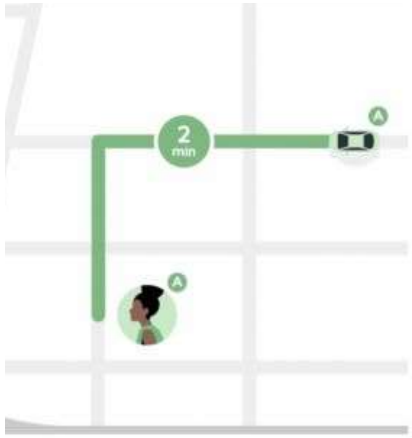


Question: How does Uber do the matching between the drivers and riders?

- Swith to ETA (estimated time of arrival) method:
  Match based on the ***smallest ETA*** between drivers and riders

Question: How does Uber do the matching between the drivers and riders?

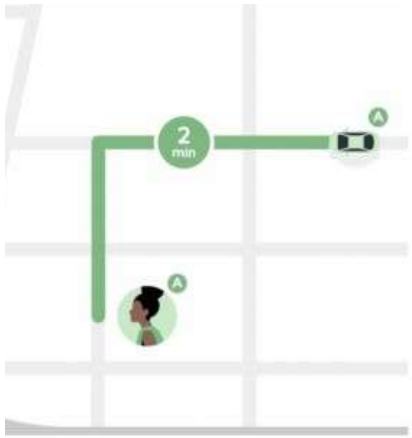
- Swith to ETA (estimated time of arrival) method:
  Match based on the ***smallest ETA*** between drivers and riders



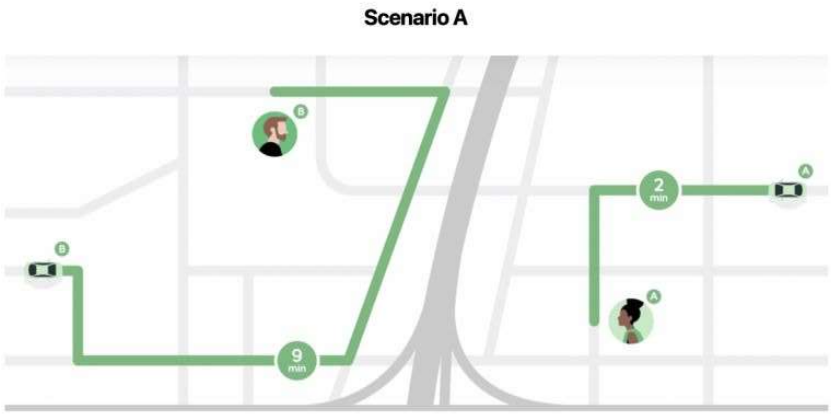
A: First request

Question: How does Uber do the matching between the drivers and riders?

- Swith to as ETA (estimated time of arrival) method:
  Match based on the ***smallest ETA*** between drivers and riders



A: First request



B: Second request

Question: How does Uber do the matching between the drivers and riders?

- Batched matching (currently used in Uber):
  Evaluate nearby drivers and riders in one batch
  => introduce some delay to minimize the averaged waiting time

Question: How does Uber do the matching between the drivers and riders?

- Batched matching (currently used in Uber):
  Evaluate nearby drivers and riders in one batch
  => introduce some delay to minimize the averaged waiting time

Rough Idea

- We have n vertices with a weight function $w(i, j)$ in a general graph
  [Not necessary be a bipartite graph, will be explained later]

Rough Idea

- We have *n* vertices with a weight function *w(i, j)* in a general graph
  [Not necessary be a bipartite graph, will be explained later]
- The weight *w(i, j)* will consider all the possible related factors including distance, ETA, traffic, etc.

Rough Idea

- We have *n* vertices with a weight function *w(i, j)* in a general graph
  [Not necessary be a bipartite graph, will be explained later]
- The weight *w(i, j)* will consider all the possible related factors including distance, ETA, traffic, etc.
- Vertices arrive and leave randomly over time.

Rough Idea

- We have *n* vertices with a weight function *w(i, j)* in a general graph
  [Not necessary be a bipartite graph, will be explained later]
- The weight *w(i, j)* will consider all the possible related factors including distance, ETA, traffic, etc.
- Vertices arrive and leave randomly over time.
- Batching algorithm: do the matching over a small time window

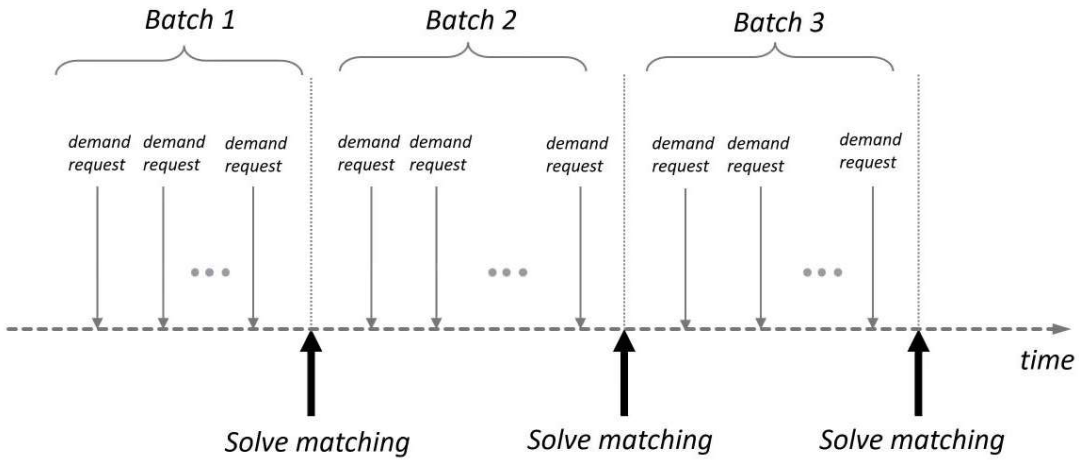


Image Source: Yiding Feng

Rough Idea

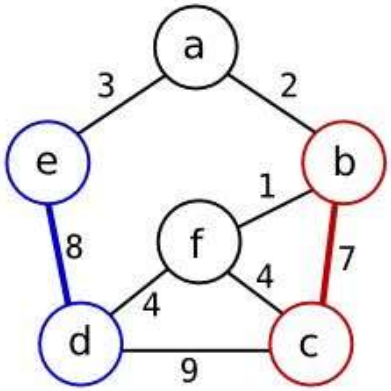
- We have *n* vertices with a weight function *w(i, j)* in a general graph
  [Not necessary be a bipartite graph, will be explained later]
- The weight *w(i, j)* will consider all the possible related factors including distance, ETA, traffic, etc.
- Vertices arrive and leave randomly over time.
- Batching algorithm: do the matching over a small time window
- Goal: maximize the expected weight of the matching

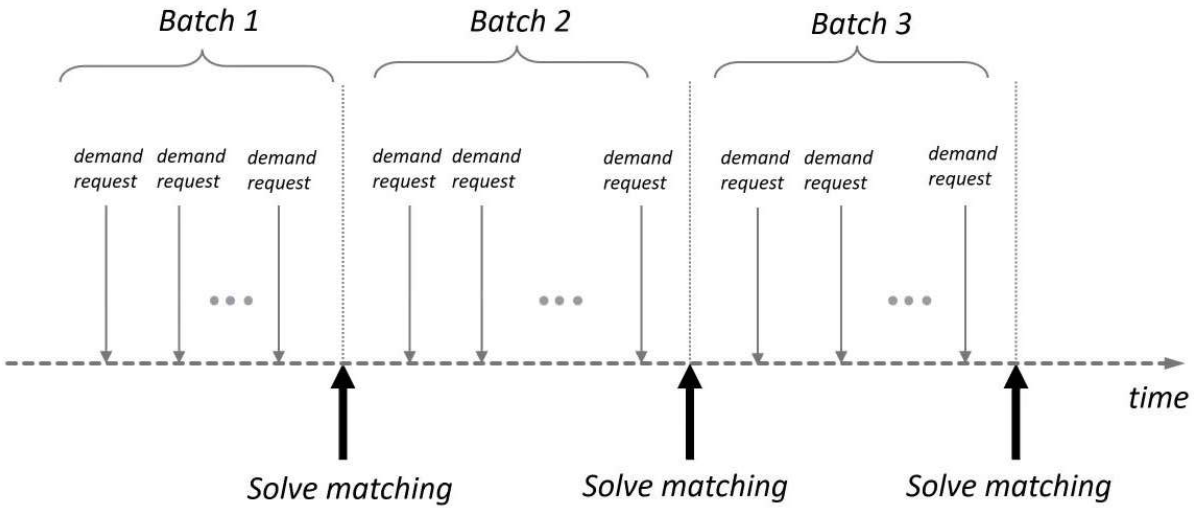
Rough Idea

- We have *n* vertices with a weight function *w(i, j)* in a general graph
  [Not necessary be a bipartite graph, will be explained later]
- The weight *w(i, j)* will consider all the possible related factors including distance, ETA, traffic, etc.
- Vertices arrive and leave randomly over time.
- Batching algorithm: do the matching over a small time window
- Goal: maximize the expected weight of the matching

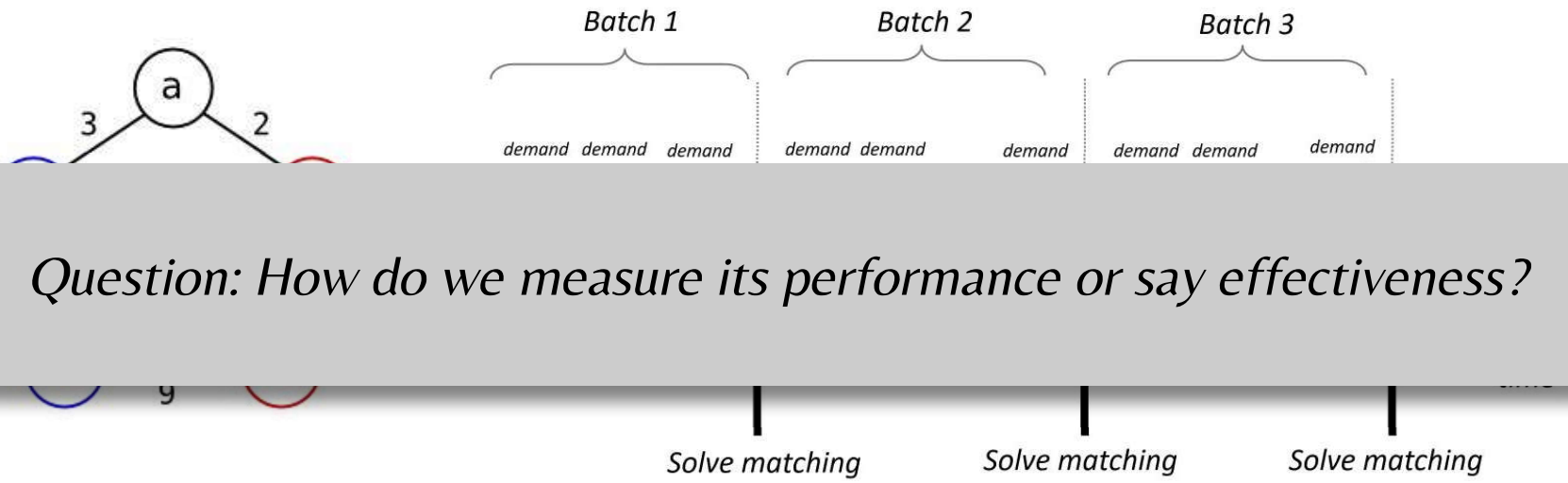
Question: Any idea about solving such matching (Hint: HW2 1(b))?

Question: Any simple idea about solving such matching (Hint: HW2 1(b))?
Maximum-weight matching!





Question: Any simple idea about solving such matching (Hint: HW2 1(b))?
Maximum-weight matching!



Question: How do we measure its performance or say effectiveness?

Question: How do we measure its performance or say effectiveness?

Online algorithm: do the matching based on what we see so far.

Offline algorithm: do the maximum weight matching in hindsight, namely, we know all the information in advance.

Question: How do we measure its performance or say effectiveness?

Online algorithm: do the matching based on what we see so far.

Offline algorithm: do the maximum weight matching in hindsight, namely, we know all the information in advance.

Competitive ratio: The worst case ratio between the performance of these two.

Namely, worst performance of our online algorithm/ performance of optimal offline algorithm

Online Windowed Matching (Ashlagi et al. 2018):

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

Online Windowed Matching (Ashlagi et al. 2018):

- Consider an edge-weighted undirected graph G with n vertices indexed by $i \in [n]$.

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

Online Windowed Matching (Ashlagi et al. 2018):

- Consider an edge-weighted undirected graph G with n vertices indexed by $i \in [n]$.
- The vertices arrive sequentially over n time periods, denoted by $t \in [n]$.

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

Online Windowed Matching (Ashlagi et al. 2018):

- Consider an edge-weighted undirected graph G with n vertices indexed by $i \in [n]$.
- The vertices arrive sequentially over n time periods, denoted by $t \in [n]$.
- Denote by $\sigma(i)$ the arrival time of vertex i.

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

Online Windowed Matching (Ashlagi et al. 2018):

- Consider an edge-weighted undirected graph G with n vertices indexed by $i \in [n]$.
- The vertices arrive sequentially over n time periods, denoted by $t \in [n]$.
- Denote by $\sigma(i)$ the arrival time of vertex i.
- The time deadline is d, namely, we only consider the matching between vertices that satify $|\sigma(i)-\sigma(j)| \leq d$.

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

Online Windowed Matching (Ashlagi et al. 2018):

- Consider an edge-weighted undirected graph G with n vertices indexed by $i \in [n]$.
- The vertices arrive sequentially over n time periods, denoted by $t \in [n]$.
- Denote by $\sigma(i)$ the arrival time of vertex i.
- The time deadline is d, namely, we only consider the matching between vertices that satify $|\sigma(i)-\sigma(j)| \leq d$.
- Random arrival: the vertices arrive in a uniformly random order.

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

Online Windowed Matching (Ashlagi et al. 2018):

- Consider an edge-weighted undirected graph *G* with *n* vertices indexed by $i \in [n]$.
- The vertices arrive sequentially over *n* time periods, denoted by $t \in [n]$.
- Denote by $\sigma(i)$ the arrival time of vertex *i*.
- The time deadline is *d,* namely, we only consider the matching between vertices that satify $|\sigma(i)-\sigma(j)| \le d$.
- Random arrival: the vertices arrive in a uniformly random order.

$$\begin{aligned}
\text{maximize} \quad & \sum_{i,j\in[n]:\ i<j,\ |\sigma(i)-\sigma(j)|\le d} x_{ij} v_{ij} \\
\text{subject to} \quad & \sum_{j\in[n]:\ i<j} x_{ij} + \sum_{j\in[n]:\ j<i} x_{ji} \le 1 \qquad \forall i \in [n], \qquad \text{(Offline Matching)} \\
& x_{ij} \in \{0,1\} \ \forall i<j,\ i,j\in[n].
\end{aligned}$$

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

Online Windowed Matching (Ashlagi et al. 2018):

- Consider an edge-weighted undirected graph *G* with *n* vertices indexed by $i \in [n]$.
- The vertices arrive sequentially over *n* time periods, denoted by $t \in [n]$.
- Denote by $\sigma(i)$ the arrival time of vertex *i*.
- The time deadline is *d,* namely, we only consider the matching between vertices that satify $|\sigma(i)-\sigma(j)| \le d$.
- Random arrival: the vertices arrive in a uniformly random order.

$$\begin{aligned}
\text{maximize} \quad & \sum_{i,j\in[n]:\ i<j,\ |\sigma(i)-\sigma(j)|\le d} x_{ij} v_{ij} \\
\text{subject to} \quad & \sum_{j\in[n]:\ i<j} x_{ij} + \sum_{j\in[n]:\ j<i} x_{ji} \le 1 \qquad \forall i \in [n], \qquad \text{(Offline Matching)} \\
& x_{ij} \in \{0,1\} \ \forall i<j,\ i,j\in[n].
\end{aligned}$$

BATCHING: compute and select maximum-weight matchings periodically, namely, every *d+1* time steps. Every vertex in the selected matching is then matched, and all other vertices are discarded. (Online Matching)

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

$$\begin{aligned}
\text{maximize} \quad & \sum_{i,j\in[n]:\ i<j,\ |\sigma(i)-\sigma(j)|\le d} x_{ij} v_{ij} \\
\text{subject to} \quad & \sum_{j\in[n]:\ i<j} x_{ij} + \sum_{j\in[n]:\ j<i} x_{ji} \le 1 \qquad \forall i \in [n], \qquad \text{(Offline Matching)} \\
& x_{ij} \in \{0,1\} \ \forall i<j,\ i,j\in[n].
\end{aligned}$$

BATCHING: compute and select maximum-weight matchings periodically, namely, every *d+1* time steps. Every vertex in the selected matching is then matched, and all other vertices are discarded. (Online Matching)

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

$$\begin{aligned}
\text{maximize} \quad & \sum_{i,j\in[n]:\ i<j,\ |\sigma(i)-\sigma(j)|\le d} x_{ij} v_{ij} \\
\text{subject to} \quad & \sum_{j\in[n]:\ i<j} x_{ij} + \sum_{j\in[n]:\ j<i} x_{ji} \le 1 \qquad \forall i \in [n], \qquad \text{(Offline Matching)} \\
& x_{ij} \in \{0,1\} \ \forall i<j,\ i,j\in[n].
\end{aligned}$$

BATCHING: compute and select maximum-weight matchings periodically, namely, every *d+1* time steps. Every vertex in the selected matching is then matched, and all other vertices are discarded. (Online Matching)

Thm 1 (*random arrival***):** *The competitive ratio of BATCHING is at least (0.279 + O(1/n)). Proof sketch: 1. Reduce to a graph coverring problem; 2. amplificatiooin gadgets for small n to large n & small d to large d; 3. computer aided LP-based proof for small n and d.*

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

$$\begin{aligned}
\text{maximize} \quad & \sum_{i,j\in[n]:\ i<j,\ |\sigma(i)-\sigma(j)|\le d} x_{ij} v_{ij} \\
\text{subject to} \quad & \sum_{j\in[n]:\ i<j} x_{ij} + \sum_{j\in[n]:\ j<i} x_{ji} \le 1 \qquad \forall i \in [n], \qquad \text{(Offline Matching)} \\
& x_{ij} \in \{0,1\} \ \forall i<j,\ i,j\in[n].
\end{aligned}$$

BATCHING: compute and select maximum-weight matchings periodically, namely, every *d+1* time steps. Every vertex in the selected matching is then matched, and all other vertices are discarded. (Online Matching)

Thm 1 (*random arrival*)**:** *The competitive ratio of BATCHING is at least (0.279 + O(1/n)).*
Proof sketch: 1. Reduce to a graph coverring problem; 2. amplificatiooin gadgets for small n to large n & small d to large d; 3. computer aided LP-based proof for small n and d.

Thm 2 (*random arrival*)**:** *No algorithm is more than 1/2-competitive.*
Proof sketch: If it is (½+ε)-competitive, consider a graph with three vertices {1, 2, 3}, with $v_{12} = \alpha$ *and* $v_{23} = v_{13} = \varepsilon'\alpha$*, d = 1, then choosing* $\varepsilon' < \frac{2}{3}\varepsilon$ *gives us the desired contradiction.*

Source: Edge Weighted Online Windowed Matching, Mathematics of Operations Research, May 2022

➢ Secretary Matching (Ezra et al. 2020) - 5/12 competitive ratio

ALGORITHM 1: 5/12-matching secretary for vertex arrival (for $k = \lfloor \frac{n}{2} \rfloor$)

Let $v_1, \ldots, v_n$ be the vertices in arrival order
$A = V$ // A is the set of available vertices
$\mu = \varnothing$ // μ is the returned matching
for $t \in \{k+1, ..., n\}$ **do**
  Let $V_t = \{v_1, \ldots, v_t\}$ // V_t is the set of vertices arrived up to time t
  if t is odd **then**
    Select $r_t \in \{1, \ldots t-1\}$ uniformly at random
    Set $V'_t = V_t \backslash \{v_{r_t}\}$ // delete a random vertex from $v_1, \ldots, v_{t-1}$
  else
    Set $V'_t = V_t$
  end if
  Let μ_t be the maximum weighted matching in $G(V'_t)$
  if $\mu_t(v_t) \in V_t \cap A$ **then**
    Add $e_t \stackrel{\text{def}}{=} v_t \mu_t(v_t)$ to μ // add the chosen edge to the matching
    Remove v_t and $\mu_t(v_t)$ from A
  end if
end for
Return matching μ

Notice: Vertices will not depart in this algorithm, besides the bound here is proven to be tight. The corresponding offline algorithm is also different.

Source: Secretary Matching with General Arrivals, Nov 2020

Recap

❑ Recap

Notice we only assume the graph here is general instead of as a bipartite graph, Why?

- Some scenarios cannot be captured by a bipartite graph at all, e.g., a pool of students who should be paired into roommates, or exchange markets for pairwise kidney exchange.

❑ Recap

Notice we only assume the graph here is general instead of as a bipartite graph, Why?

- Some scenarios cannot be captured by a bipartite graph at all, e.g., a pool of students who should be paired into roommates, or exchange markets for pairwise kidney exchange.

Physical understanding in previous models for ride-sharing

1. Vertices can be viewed as passengers who request to share a ride.
   Then, leaving unmatched after *d* periods can be interpreted as sending a passenger who will not be shared with another passenger on a single trip.
2. Vertices can be viewed as the union of the riders and drivers

❑ Recap

Notice we only assume the graph here is general instead of as a bipartite graph, Why?

- Some scenarios cannot be captured by a bipartite graph at all, e.g., a pool of students who should be paired into roommates, or exchange markets for pairwise kidney exchange.

Physical understanding in previous models for ride-sharing

1. Vertices can be viewed as passengers who request to share a ride.
   Then, leaving unmatched after *d* periods can be interpreted as sending a passenger who will not be shared with another passenger on a single trip.
2. Vertices can be viewed as the union of the riders and drivers

Problems:

1. Do not consider the sensitivity to waiting.
2. Not so sure if competitive ratio is a good metric, although it seems reasonable.
3. Batching can be arbitrarily bad in a different setting where vertices have widely heterogeneous sojourn times in the market (Aouad et al. 2020).

❑ Recap

Notice we only assume the graph here is general instead of as a bipartite graph, Why?

- Some scenarios cannot be captured by a bipartite graph at all, e.g., a pool of students who should be paired into roommates, or exchange markets for pairwise kidney exchange.

Physical understanding in previous models for ride-sharing

Qeustion: Any better idea?

who will not be shared with another passenger on a single trip.

2. Vertices can be viewed as the union of the riders and drivers

Problems:

1. Do not consider the sensitivity to waiting.
2. Not so sure if competitive ratio is a good metric, although it seems reasonable.
3. Batching can be arbitrarily bad in a different setting where vertices have widely heterogeneous sojourn times in the market (Aouad et al. 2020).

❑ Recap

Notice we only assume the graph here is general instead of as a bipartite graph, Why?

- Some scenarios cannot be captured by a bipartite graph at all, e.g., a pool of students who should be paired into roommates, or exchange markets for pairwise kidney exchange.

Physical understanding in previous models for ride-sharing

Qeustion: Any better idea?

Could we do the Batching with some stochastic information in the future?

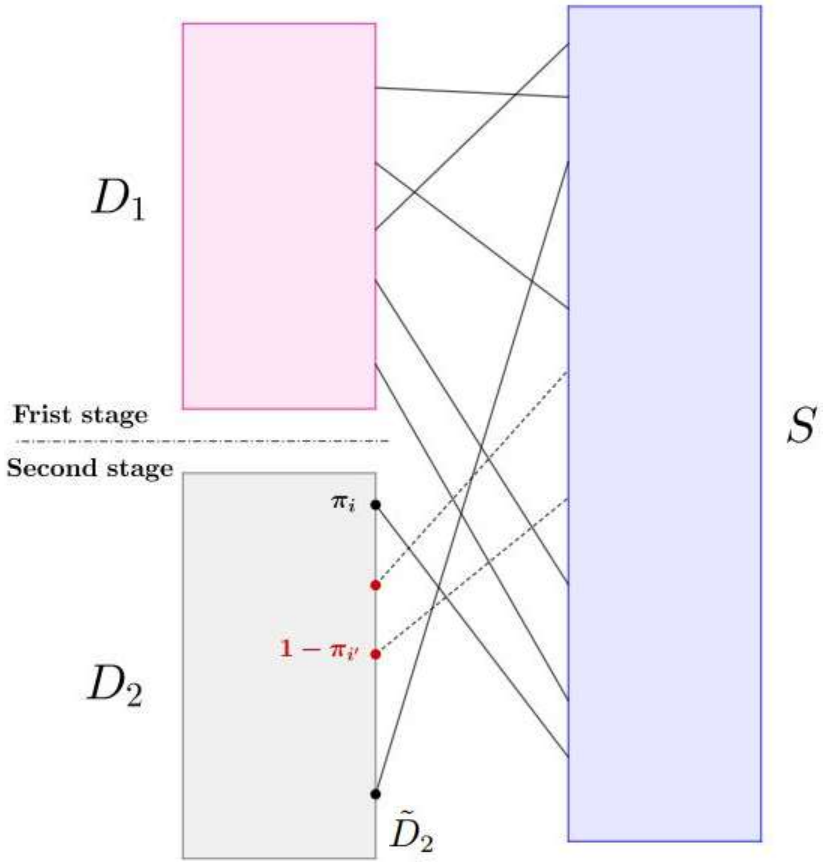
who will not be shared with another passenger on a single trip.

2. Vertices can be viewed as the union of the riders and drivers

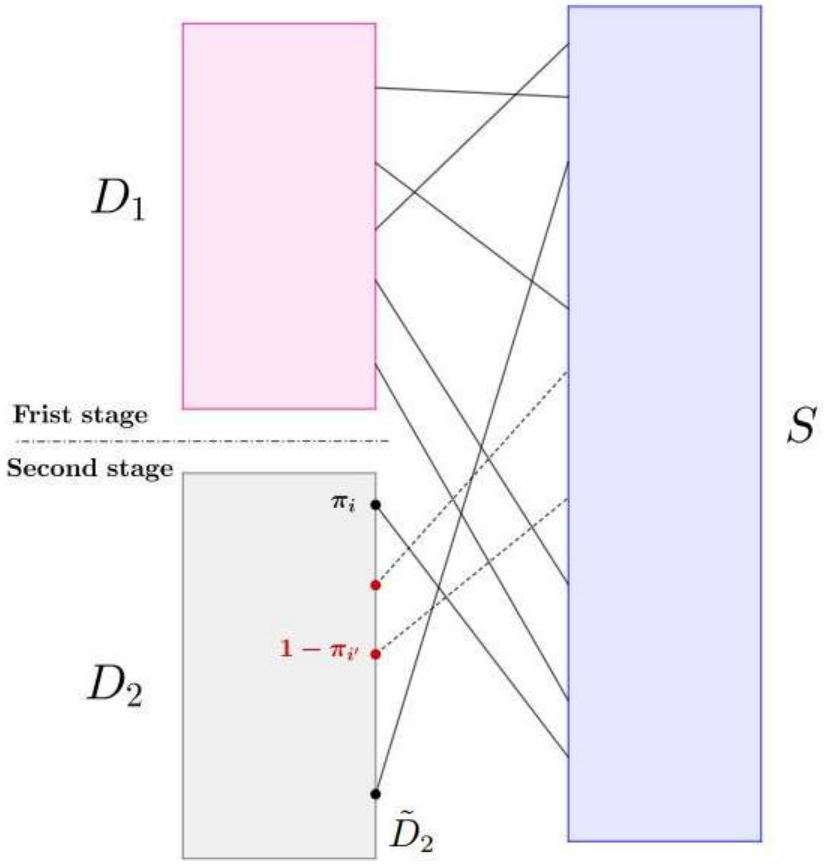
Problems:

1. Do not consider the sensitivity to waiting.
2. Not so sure if competitive ratio is a good metric, although it seems reasonable.
3. Batching can be arbitrarily bad in a different setting where vertices have widely heterogeneous sojourn times in the market (Aouad et al. 2020).

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)

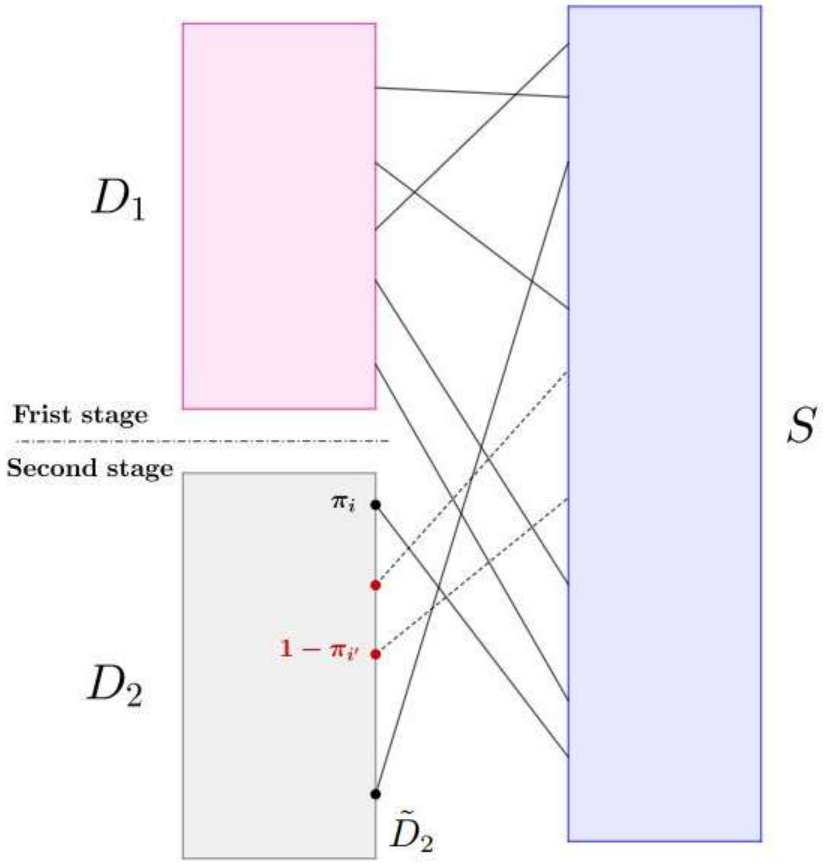


❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



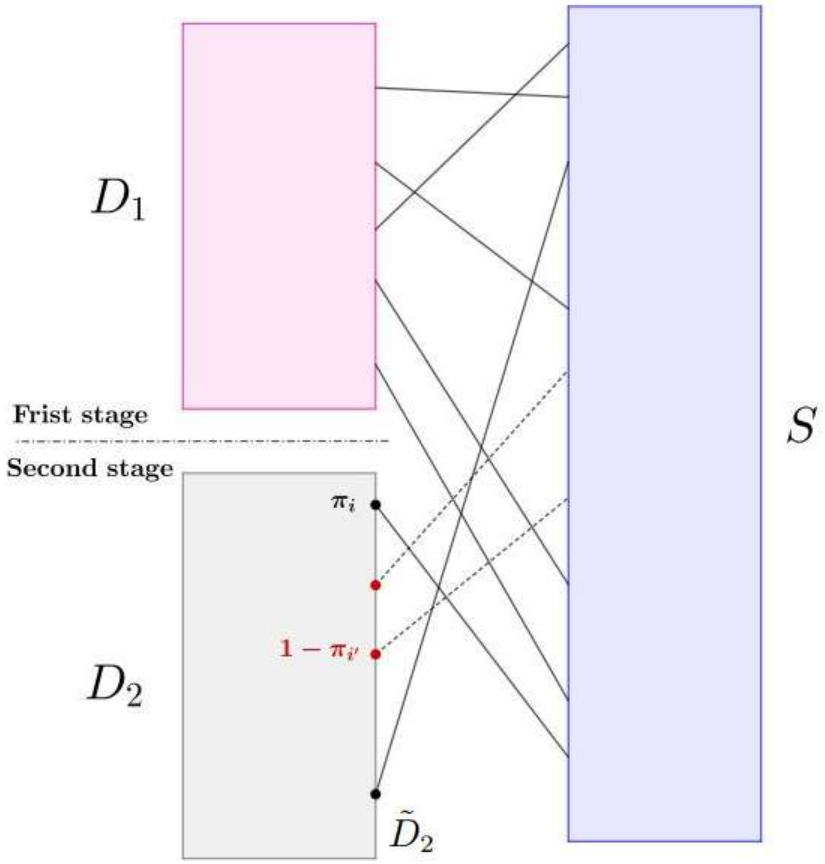
- Bipartite Graph *G = (D, S, E)*
  - ❖ rider set D, driver set S, edge set E

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



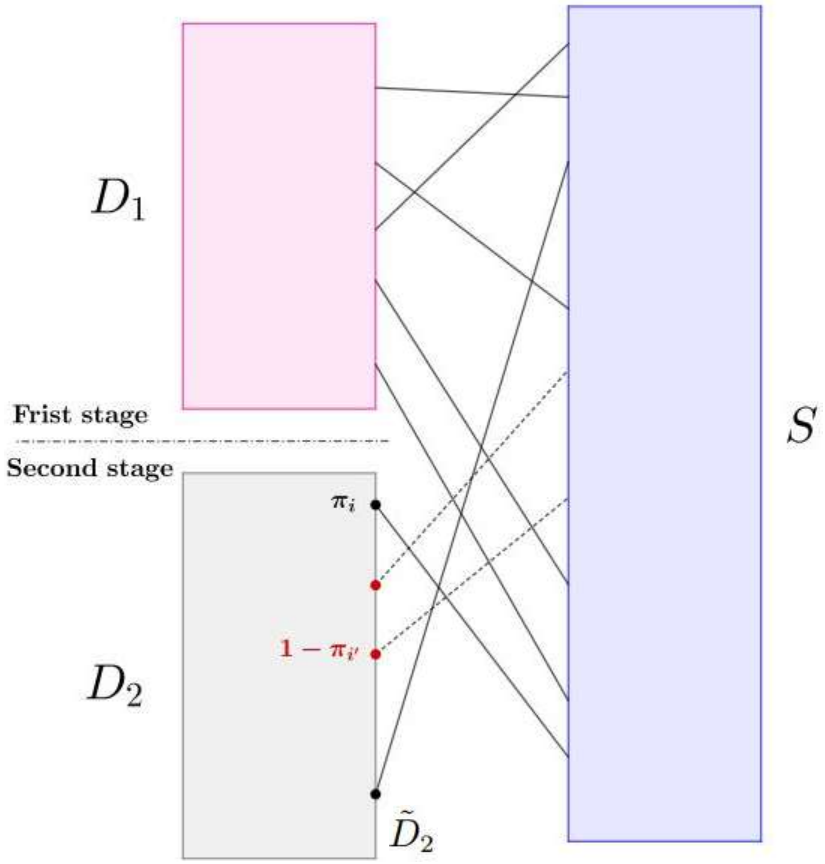
- Bipartite Graph *G = (D, S, E)*
  - ❖ rider set D, driver set S, edge set E
  - ❖ weights
    - ➢ case 1: general weight between rider and driver, the *w(i, j)* is just as shown before
    - ➢ case 2: driver weight w_j

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



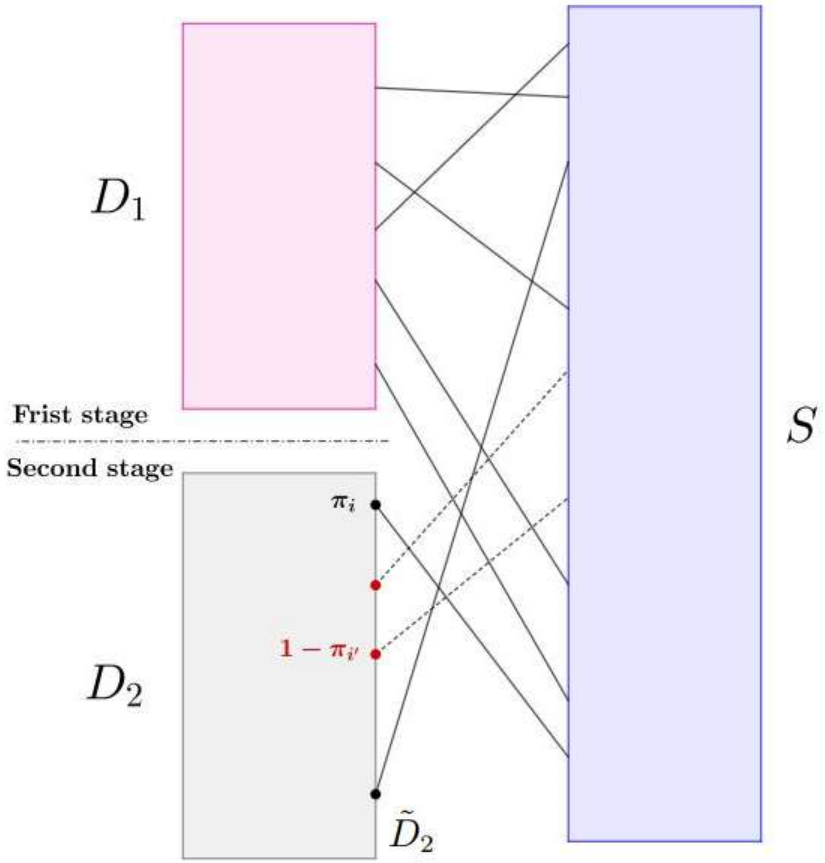
- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E
  - ❖ weights
    - ➢ case 1: general weight between rider and driver, the $w(i, j)$ is just as shown before
    - ➢ case 2: driver weight w_j
  - ❖ D is partitioned into D_1 and D_2

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



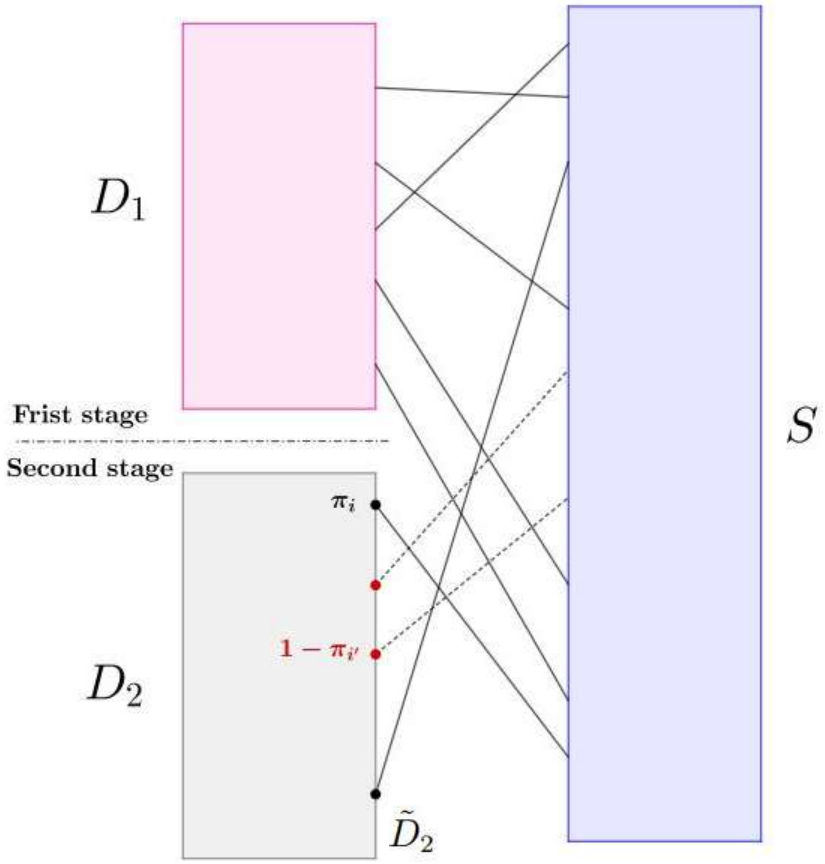
- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E
  - ❖ weights
    - ➢ case 1: general weight between rider and driver, the $w(i, j)$ is just as shown before
    - ➢ case 2: driver weight w_j
  - ❖ D is partitioned into D_1 and D_2
- Two stage

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



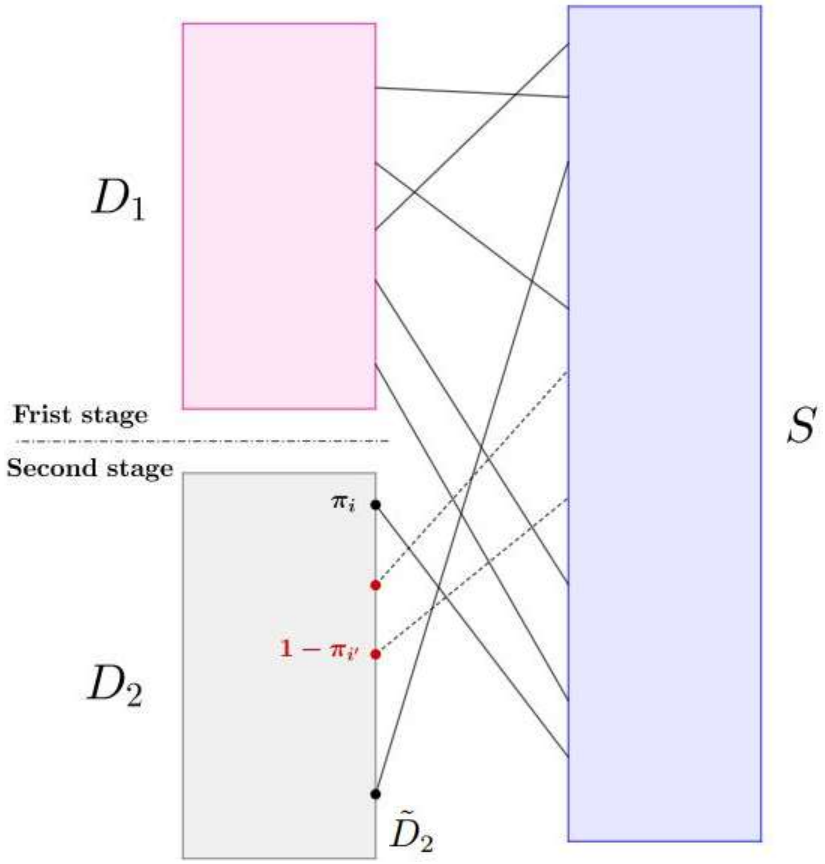
- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E
  - ❖ weights
    - ➢ case 1: general weight between rider and driver, the $w(i, j)$ is just as shown before
    - ➢ case 2: driver weight w_j
  - ❖ D is partitioned into D_1 and D_2
- Two stage
  - ❖ first stage: pick matching between D_1 and S

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



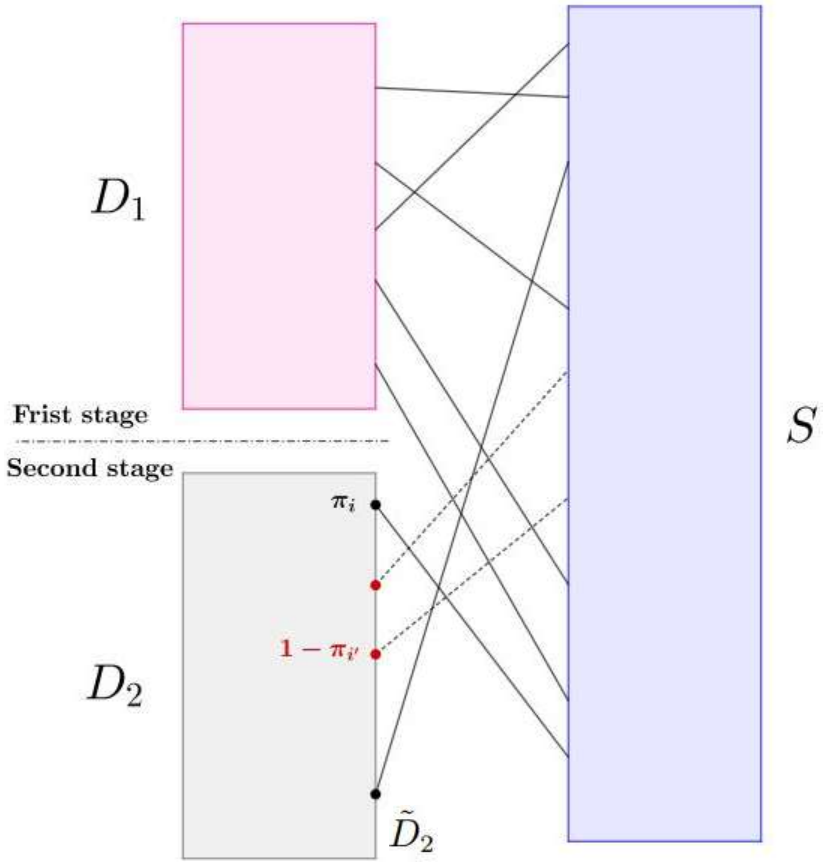
- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E
  - ❖ weights
    - ➢ case 1: general weight between rider and driver, the $w(i, j)$ is just as shown before
    - ➢ case 2: driver weight w_j
  - ❖ D is partitioned into D_1 and D_2
- Two stage
  - ❖ first stage: pick matching between D_1 and S
  - ❖ second stage: each rider in D_2 arrives independently at random with probability π_i, then the platfofrm picks the matching between the arrived rider in D_2 and the unmatched driver in S

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



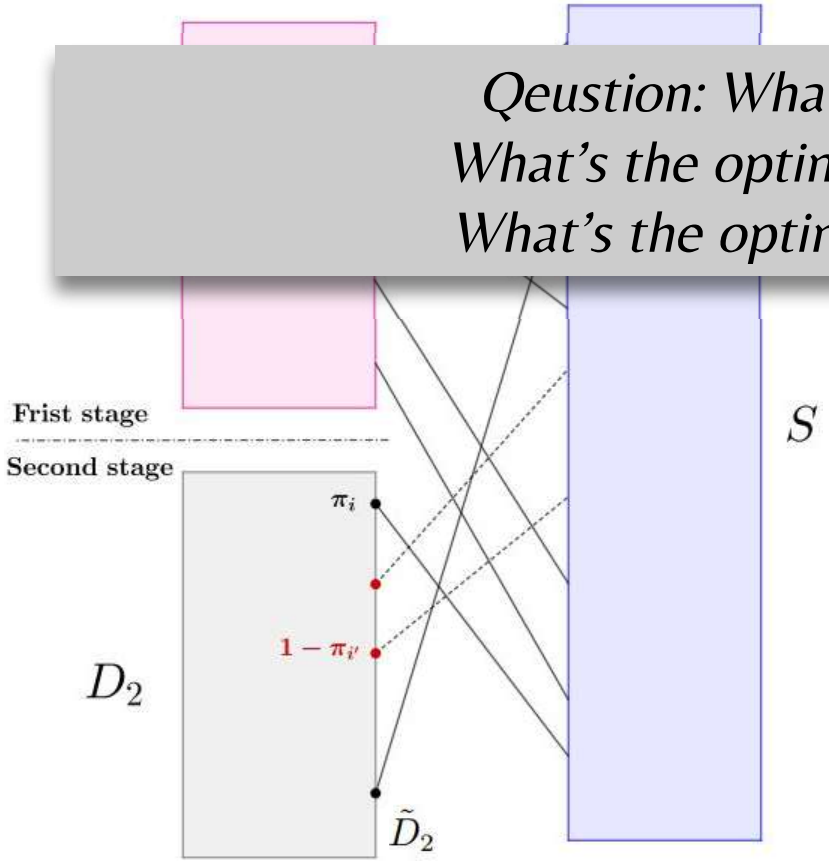
- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E
  - ❖ weights
    - ➢ case 1: general weight between rider and driver, the $w(i, j)$ is just as shown before
    - ➢ case 2: driver weight w_j
  - ❖ D is partitioned into D_1 and D_2
- Two stage
  - ❖ first stage: pick matching between D_1 and S
  - ❖ second stage: each rider in D_2 arrives independently at random with probability π_i, then the platfofrm picks the matching between the arrived rider in D_2 and the unmatched driver in S
- Goal: maximize the expected efficiency, i.e., the total weights on the matching

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E
  - ❖ weights
    - ➢ case 1: general weight between rider and driver, the $w(i, j)$ is just as shown before
    - ➢ case 2: driver weight w_j
  - ❖ D is partitioned into D_1 and D_2
- Two stage
  - ❖ first stage: pick matching between D_1 and S
  - ❖ second stage: each rider in D_2 arrives independently at random with probability π_i, then the platfofrm picks the matching between the arrived rider in D_2 and the unmatched driver in S
- Goal: maximize the expected efficiency, i.e., the total weights on the matching
- Physical meaning:
  - ❖ D_1 represents the set of riders who have made a request for a ride
  - ❖ D_2 represents the set of riders who are entering their destinations in the application and may or may not make a request.

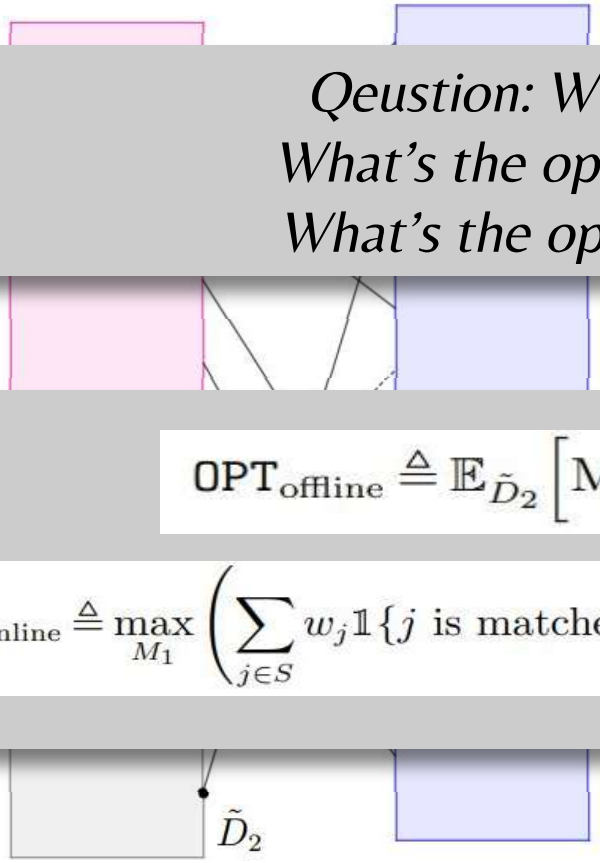
❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



Qeustion: What's the randomness here?
What's the optimal offline algorithm here?
What's the optimal online algorithm here?

- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E
  - ❖ D is partitioned into D_1 and D_2
- Two stage
  - ❖ first stage: pick matching between D_1 and S
  - ❖ second stage: each rider in D_2 arrives independently at random with probability π_i, then the platfofrm picks the matching between the arrived rider in D_2 and the unmatched driver in S
- Goal: maximize the expected efficiency, i.e., the total weights on the matching
- Physical meaning:
  - ❖ D_1 represents the set of riders who have made a request for a ride
  - ❖ D_2 represents the set of riders who are entering their destinations in the application and may or may not make a request.

❑ Two-stage Matching in a batch (Feng, Niazadeh, Saberi, 2020)



- Bipartite Graph $G = (D, S, E)$
  - ❖ rider set D, driver set S, edge set E

Qeustion: What's the randomness here?
What's the optimal offline algorithm here?
What's the optimal online algorithm here?

- Two stage
  - ❖ first stage: pick matching between D_1 and S

$$\texttt{OPT}_{\text{offline}} \triangleq \mathbb{E}_{\tilde{D}_2}\left[\textsc{Max-WeightMatch}^{\mathbf{w}}(D_1 \cup \tilde{D}_2, S)\right]$$

$$\texttt{OPT}_{\text{online}} \triangleq \max_{M_1}\left(\sum_{j\in S} w_j \mathbb{1}\{j \text{ is matched in } M_1\} + \mathbb{E}_{\tilde{D}_2}\left[\textsc{Max-WeightMatch}^{\mathbf{w}}(\tilde{D}_2, S\setminus S_1)\right]\right)$$

request for a ride

  - ❖ D_2 represents the set of riders who are entering their destinations in the application and may or may not make a request.

❑ Case 1: general weight *w(i,j)*

Algorithm 5 Greedy with Random Discard

1: **input:** bipartite graph $G = (D, S, E)$, non-negative edge weights $\{w_e\}_{e \in E}$.

2: **output:** bipartite matching M_1 in the graph $G[D_1, S]$.

3: Let M_1 be the maximum weighted matching in the graph $G[D_1, S]$.

4: Discard each edge $e \in M_1$ from matching M_1 with probability $1/2$.

5: Return M_1.

Thm 3 : *It is ½-competitive against the optimum offline policy.*

Thm 4 : *In the two-stage matching problem with weighted demand vertices, no policy can obtain a competitive ratio better than ½ against the optimum offline policy.*

❑ Case 1: general weight *w(i,j)*

Thm 3 : *It is ½-competitive against the optimum offline policy.*

Proof.

- At the first stage, the algorithm will first do the maximum weighted matching and then randomly discards each edge in the matching with probability 1/2, its final matching M1 is at least 1/2-approximation to the total weights of edges matched in the optimum offline policy at the first stage.

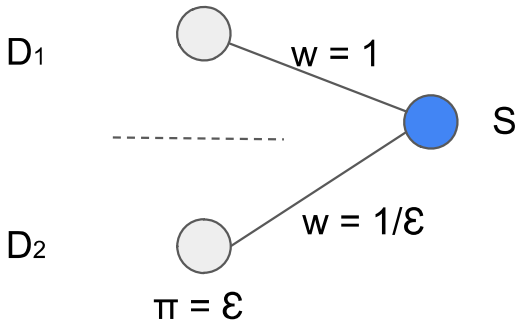
- At the second stage, notice each supply vertex in S is matched at the first stage with probability at most 1/2, then the maximum weighted matching with all remaining supply vertices and new demand vertices in D_2 at the second stage is at least 1/2-approximation to the total weights of edges matched in the optimum offline policy at the second stage.

Case 1: general weight *w(i,j)*

Thm 4 : *In the two-stage matching problem with weighted demand vertices, no policy can obtain a competitive ratio better than ½ against the optimum offline policy.*

Proof.

- Consider the following instance of two-stage matching with weighted demand vertices: There is one supply vertex and two demand vertices. Demand vertex 1 is at the first stage with weight 1. Demand vertex 2 with weight 1/Ɛ appears at the second stage with probability Ɛ. Both demand vertices can be matched to the supply vertex.

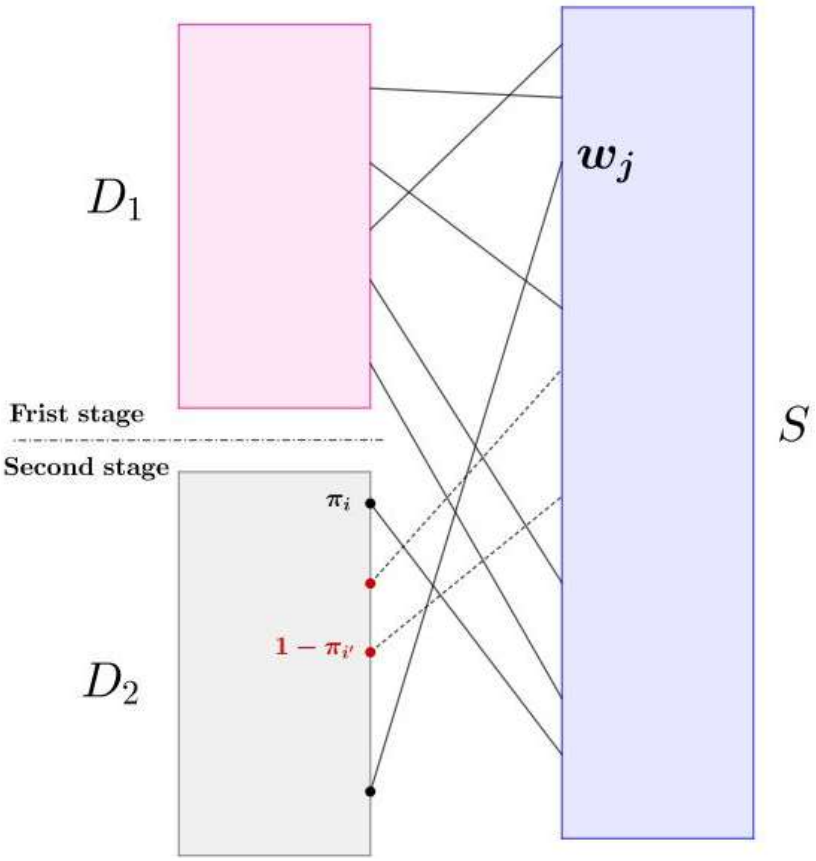


The expected total weight for any online policy is 1

The expected total weight for optimum offline policy is

$$Ɛ * 1 / Ɛ + (1 - Ɛ) * 1 = 2 - Ɛ$$

Case 2: with only driver weight *w(j)*



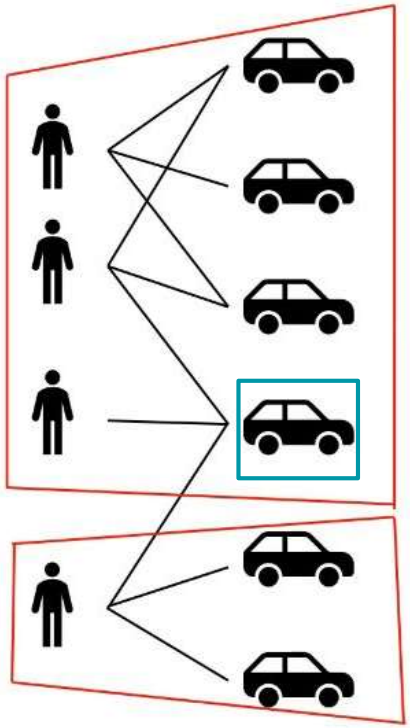
- HEDGE: ¾ competitive against optimal offline

Intuition: Consider the unweighted case first (treat all w_j as 1)

- Suppose we do not know anything about the second stage. How would we pick the matching for D_1 and S
- If it is a *complete* bipartite graph in the first stage, then we should output the matching uniformly at random, namely, each driver in S will be matched with c = min (1, $|D_1|/S$), which is the demand per driver.
- What if it is not a *complete* bipartite graph?
  => Demand-balanced (DB) Decomposition

  Goal: decompose it to several small complete graph, and then do the matching uniformly at random in each sub graph.

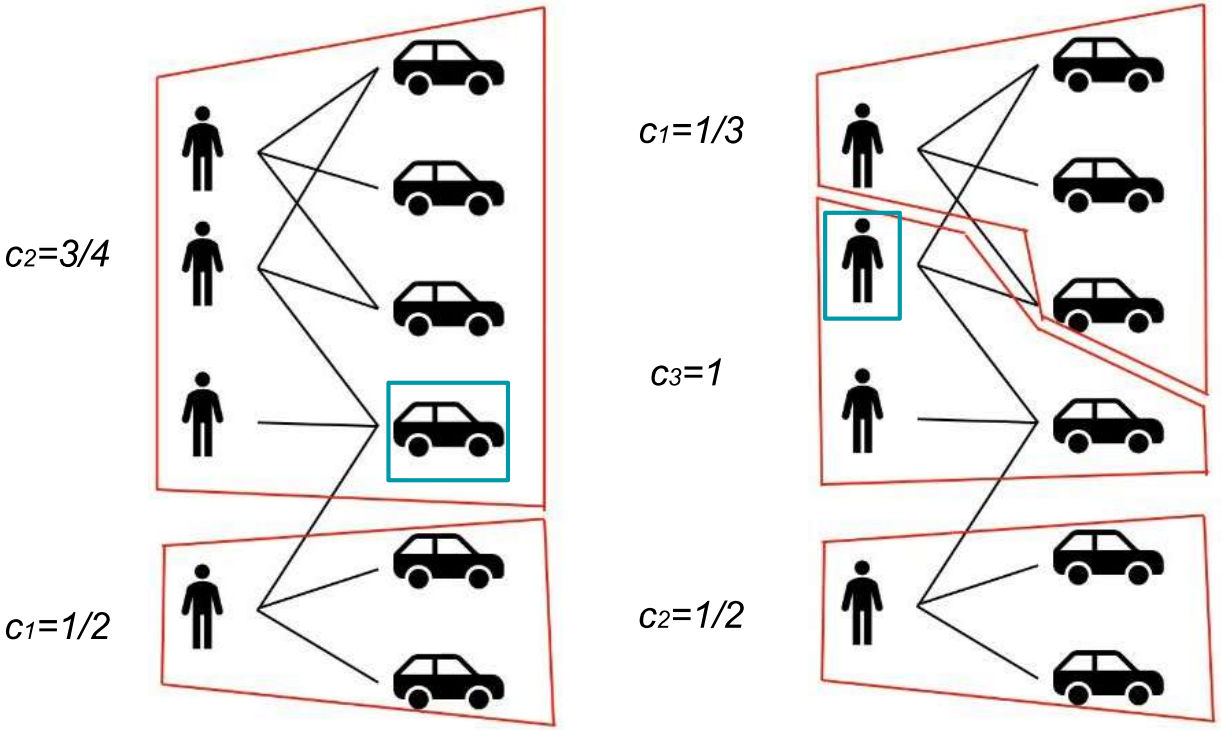
Demand-balanced (DB) Decomposition



Uniformity I: the driver in each sub graph will be matched with the same probability $c = \min(1, |D_{sub}|/S_{sub})$

Source: Two-stage Matching and Pricing with Applications to Ride Hailing, Jul 2020

Demand-balanced (DB) Decomposition

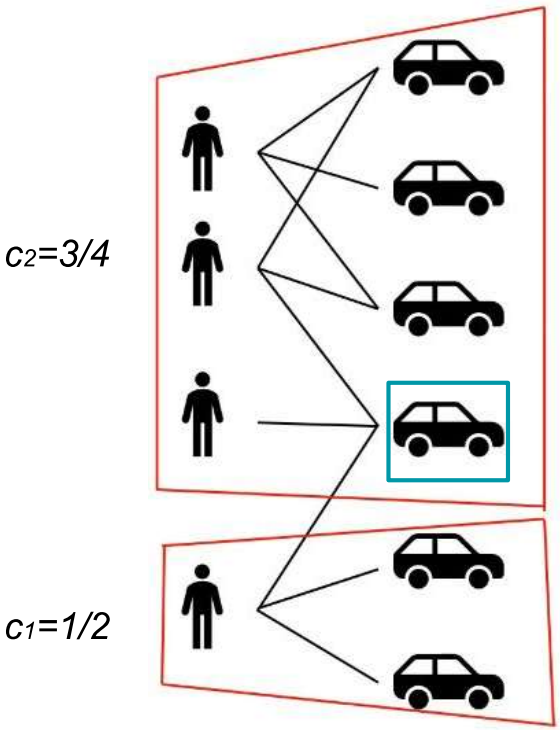


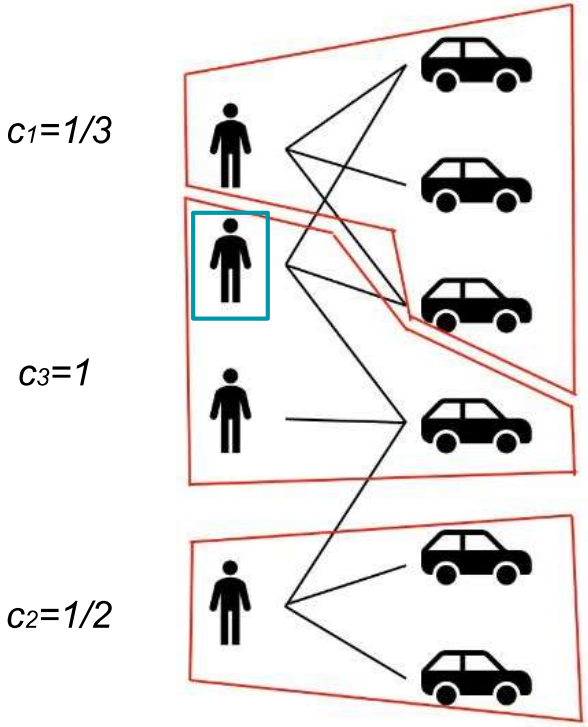
Uniformity I: the driver in each sub graph will be matched with the same probability $c = \min(1, |D_{sub}|/S_{sub})$

Uniformity II: no edge rom riders in high demand pair to drivers in low demand pair

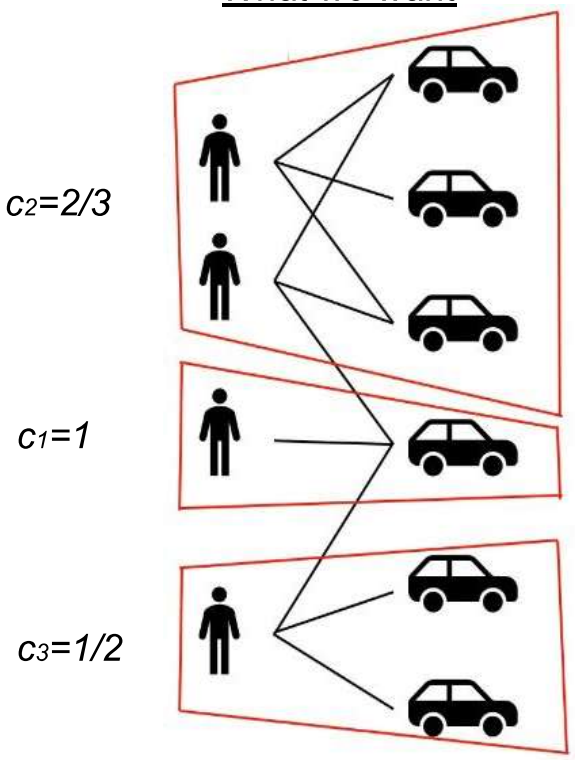
Source: Two-stage Matching and Pricing with Applications to Ride Hailing, Jul 2020

❑ Demand-balanced (DB) Decomposition





What we want



Uniformity I: the driver in each sub graph will be matched with the same probability $c = \min(1, |D_{sub}|/S_{sub})$

Uniformity II: no edge rom riders in high demand pair to drivers in low demand pair

Monotonicity: the c can be strictly sorted.

Source: Two-stage Matching and Pricing with Applications to Ride Hailing, Jul 2020

❑ Demand-balanced (DB) Decomposition

- Convex programming for balanced supply utilization

$$\{x_{ij}^*\} \in \arg\min \sum_{j \in S} g\left(w_j\left(1 - \sum_{i \in N(j)} x_{ij}\right)\right) \text{ s.t.}$$

$$\sum_{j \in N(i)} x_{ij} \leq 1 \qquad i \in D_1 \ ,$$

$$\sum_{i \in N(j)} x_{ij} \leq 1 \qquad j \in S \ ,$$

$$x_{ij} \geq 0 \qquad i \in D_1, j \in S, (i,j) \in E$$

Unweighted version

Source: Two-stage Matching and Pricing with Applications to Ride Hailing, Jul 2020

❑ Demand-balanced (DB) Decomposition

- Convex programming for balanced supply utilization

$$\{x_{ij}^*\} \in \arg\min \sum_{j\in S} g\left(w_j\left(1-\sum_{i\in N(j)} x_{ij}\right)\right) \text{ s.t.}$$
$$\sum_{j\in N(i)} x_{ij} \le 1 \qquad i \in D_1\ ,$$
$$\sum_{i\in N(j)} x_{ij} \le 1 \qquad j \in S\ ,$$
$$x_{ij} \ge 0 \qquad i \in D_1, j \in S, (i,j) \in E$$

Unweighted version

$$\{x_{ij}^*\} \in \arg\min \sum_{j\in S} \frac{1}{w_j} g\left(w_j\left(1-\sum_{i\in N(j)} x_{ij}\right)\right) \text{ s.t.}$$
$$\sum_{j\in N(i)} x_{ij} \le 1 \qquad i \in D_1\ , \qquad (\mathscr{P}^g)$$
$$\sum_{i\in N(j)} x_{ij} \le 1 \qquad j \in S\ ,$$
$$x_{ij} \ge 0 \qquad i \in D_1, j \in S, (i,j) \in E$$

Weighted version

Source: Two-stage Matching and Pricing with Applications to Ride Hailing, Jul 2020

❑ Demand-balanced (DB) Decomposition

- Convex programming for balanced supply utilization

$$\{x_{ij}^*\} \in \arg\min \sum_{j\in S} g\left(w_j\left(1-\sum_{i\in N(j)} x_{ij}\right)\right) \text{ s.t.}$$
$$\sum_{j\in N(i)} x_{ij} \le 1 \qquad i \in D_1\ ,$$
$$\sum_{i\in N(j)} x_{ij} \le 1 \qquad j \in S\ ,$$
$$x_{ij} \ge 0 \qquad i \in D_1, j \in S, (i,j) \in E$$

Unweighted version

$$\{x_{ij}^*\} \in \arg\min \sum_{j\in S} \frac{1}{w_j} g\left(w_j\left(1-\sum_{i\in N(j)} x_{ij}\right)\right) \text{ s.t.}$$
$$\sum_{j\in N(i)} x_{ij} \le 1 \qquad i \in D_1\ , \qquad (\mathscr{P}^g)$$
$$\sum_{i\in N(j)} x_{ij} \le 1 \qquad j \in S\ ,$$
$$x_{ij} \ge 0 \qquad i \in D_1, j \in S, (i,j) \in E$$

Weighted version

Remark

1. g(.) : R → R can be any differentiable, increasing, and strictly convex function.
2. By picking proper g, the objective here can be Rawlsian social welfar or Nash social welfare
3. All vertices in sub-graph are fully matched, namely, the sum of x^*_{ij} for any j is 1.
4. The x^*_{ij} is the probability induced by the random matching satisfying the *Uniformity I*.

Source: Two-stage Matching and Pricing with Applications to Ride Hailing, Jul 2020

Demand-balanced (DB) Decomposition

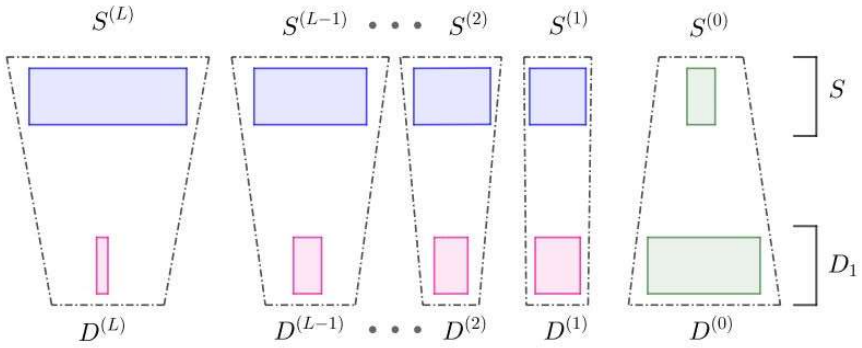


Figure 2 Structural decomposition in Lemma 1; the pairs $(D^{(l)}, S^{(l)})$ are sorted such that $0 = c^{(0)} < c^{(1)} < \ldots < c^{(L)}$. Red dashed-lines are forbidden *non-existent* edges, i.e., any edge between $D^{(l)}$ and $S^{(l')}$ when $l < l'$.

Algorithm 1 WEIGHTED-BALANCED-UTILIZATION

input: bipartite graph $G = (D, S, E)$, non-negative weights $\{w_j\}_{j \in S}$, convex function $g(\cdot)$.

output: bipartite matching M_1 in $G[D_1, S]$, bipartite matching M_2 in $G[\tilde{D}_2, S]$.

Solve convex program $\mathscr{P}^g$ to obtain $\{x^*_{ij}\}$.

Sample matching M_1 with edge marginal probabilities $\{x^*_{ij}\}$.

In the second stage, return the maximum supply-weighted matching M_2 between $\tilde{D}_2$ and the remaining vertices of S.

❑ Demand-balanced (DB) Decomposition

Thm 5 : *It is ¾-competitive against the optimum offline policy.*

Thm 6 : *In the two-stage stochastic matching for maximizing supply efficiency, no policy obtains a competitive ratio better than ¾ against OPT offline, even when all the weights are equal.*

Other Results

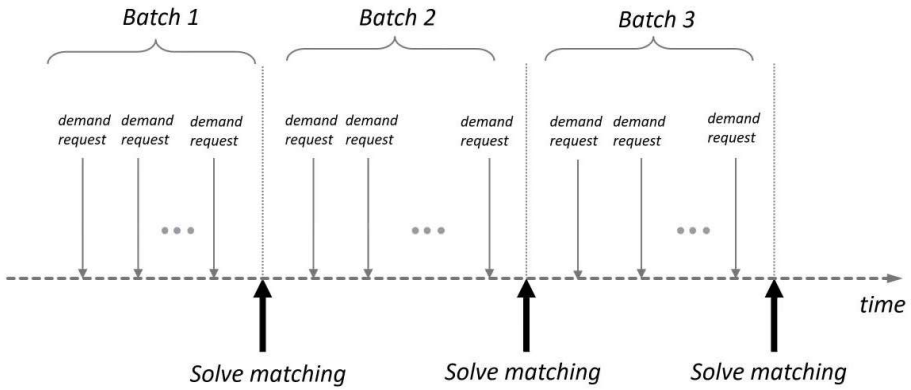
			Lower-bound (previous work)	Upper-bound (this paper)	Lower-bound (this paper)
Two-stage stochastic matching (Section 3)	Optimum offline	Weighted	$(1-\frac{1}{e})$ (*)	$\frac{3}{4}$	$\frac{3}{4}$
		Unweighted	$\frac{2}{3}$ (**)	$\frac{3}{4}$	$\frac{3}{4}$
	Optimum online	Weighted	$(1-\frac{1}{e})$ (*)	FPTAS-hard	$\left(1-\frac{1}{e}+\frac{1}{e^2}\right) \approx 0.767$
		Unweighted	$\frac{2}{3}$ (**)	FPTAS-hard	0.7613

Two-stage stochastic joint matching/pricing for market efficiency (Section 5)	-	$\frac{1}{2}$	$\frac{1}{2}$
Single-stage stochastic joint matching/pricing for demand efficiency (Appendix EC.7)	-	$1-\frac{1}{e}$ (ex-ante)	$1-\frac{1}{e}$
Two-stage stochastic matching with general edge weights (Appendix EC.8)	-	$\frac{1}{2}$	$\frac{1}{2}$

FPTAS - Fully Polynomial Time Approximation Scheme

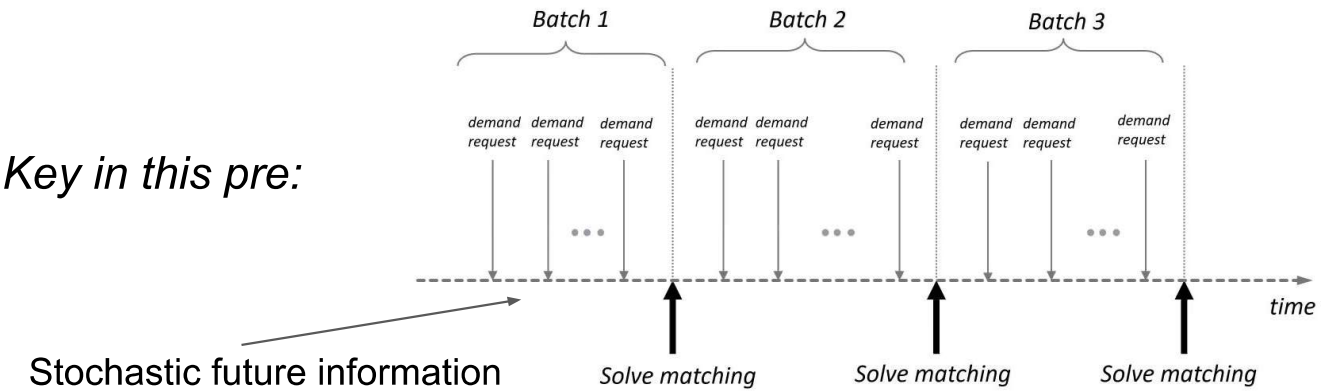
Discussion

Key in this pre:



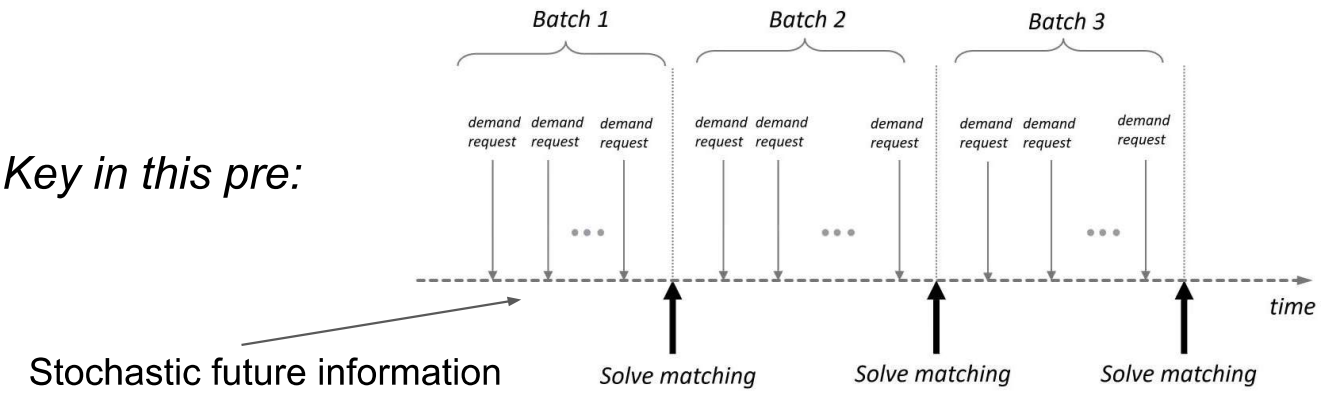
Discussion

Key in this pre:



Discussion

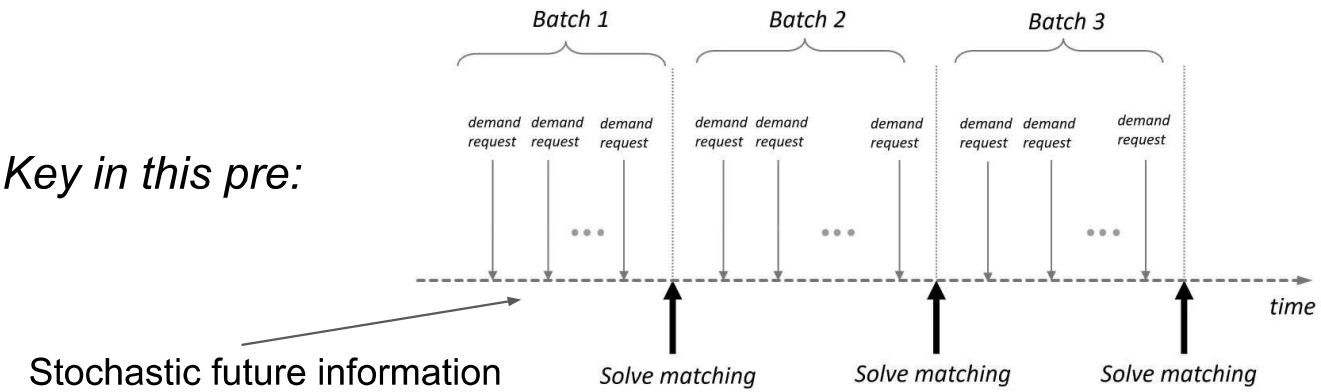
Key in this pre:



1. Most of the algorithms do not take any batch correlation into the account.
   a. How do we solve the unmatched vertices? Include them in the second batch? Shoule we improve the priority of these unmatched vertices in the second batch?
   b. Will the rider does not arrive in the second stage will appear in the second batch? Could we model it?

Discussion

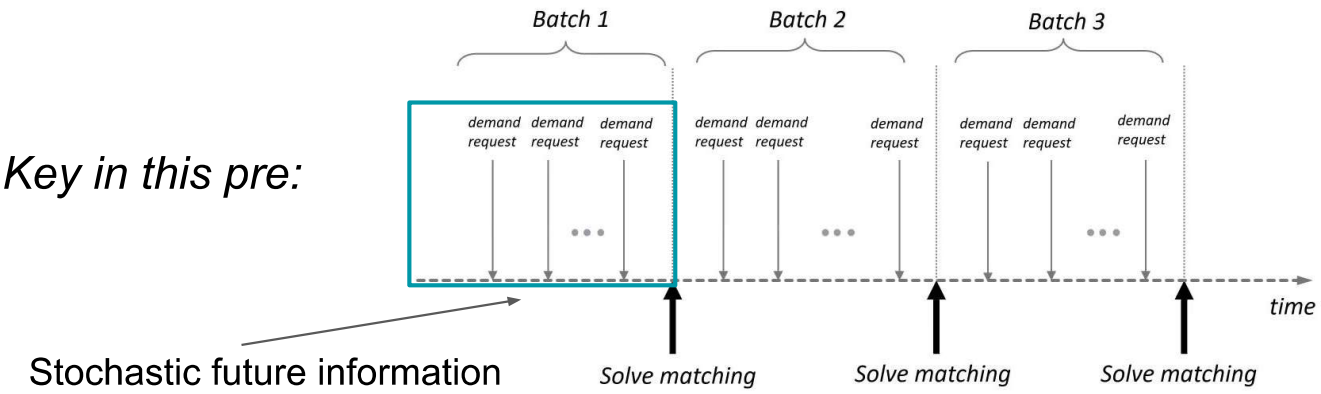
Key in this pre:



1. Most of the algorithms do not take any batch correlation into the account.
   a. How do we solve the unmatched vertices? Include them in the second batch? Shoule we improve the priority of these unmatched vertices in the second batch?
   b. Will the rider does not arrive in the second stage will appear in the second batch? Could we model it?
2. Batching stride?
   a. Currently, we do the matching after a specific time interval T. Then the unmatched vertices in the first batch have wait as least another time interval T to be matched with a driver. Then, could we leverage a dynamic shifting window?

Discussion

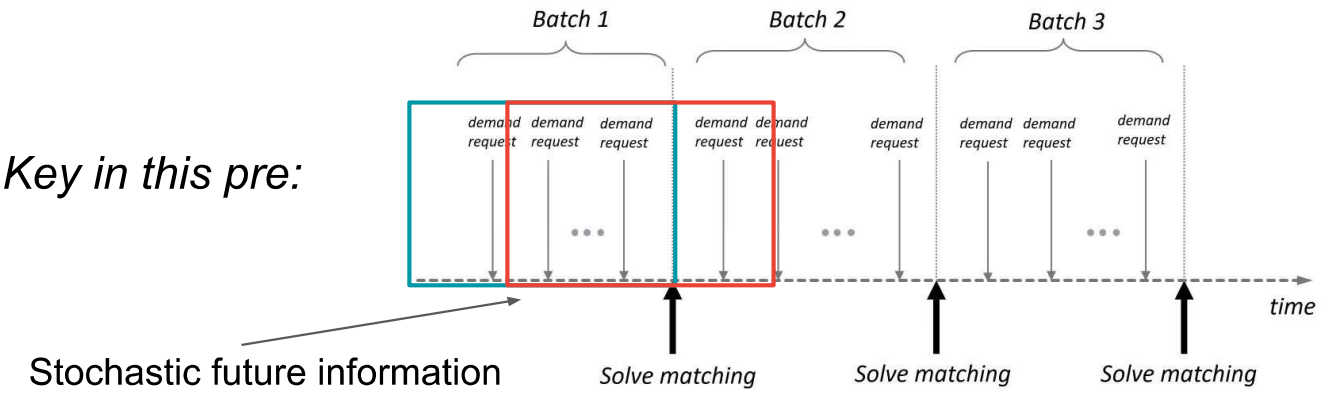
Key in this pre:



1. Most of the algorithms do not take any batch correlation into the account.
   a. How do we solve the unmatched vertices? Include them in the second batch? Shoule we improve the priority of these unmatched vertices in the second batch?
   b. Will the rider does not arrive in the second stage will appear in the second batch? Could we model it?
2. Batching stride?
   a. Currently, we do the matching after a specific time interval T. Then the unmatched vertices in the first batch have wait as least another time interval T to be matched with a driver. Then, could we leverage a dynamic shifting window?

Discussion

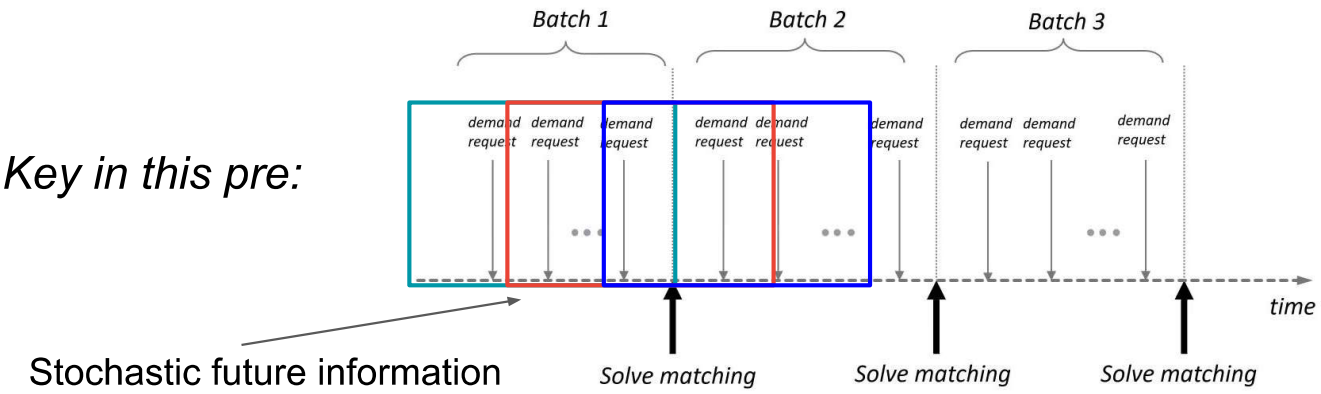
Key in this pre:



1. Most of the algorithms do not take any batch correlation into the account.
   a. How do we solve the unmatched vertices? Include them in the second batch? Shoule we improve the priority of these unmatched vertices in the second batch?
   b. Will the rider does not arrive in the second stage will appear in the second batch? Could we model it?
2. Batching stride?
   a. Currently, we do the matching after a specific time interval T. Then the unmatched vertices in the first batch have wait as least another time interval T to be matched with a driver. Then, could we leverage a dynamic shifting window?

Discussion

Key in this pre:



1. Most of the algorithms do not take any batch correlation into the account.
   a. How do we solve the unmatched vertices? Include them in the second batch? Shoule we improve the priority of these unmatched vertices in the second batch?
   b. Will the rider does not arrive in the second stage will appear in the second batch? Could we model it?
2. Batching stride?
   a. Currently, we do the matching after a specific time interval T. Then the unmatched vertices in the first batch have wait as least another time interval T to be matched with a driver. Then, could we leverage a dynamic shifting window?

❑ Other related work

- Online bipartite allocations
  - RANKING algorithm to the setting where offline vertices are weighted (Aggarwal 2011, Devanur 2013).
  - Inspired by BALANCE algorithm (Golrezaei 2014).

❑ Other related work

- Online bipartite allocations
  - RANKING algorithm to the setting where offline vertices are weighted (Aggarwal 2011, Devanur 2013).
  - Inspired by BALANCE algorithm (Golrezaei 2014).
- Other continuous-time dynamic stochastic matching
  - Vertex arrivals and departures are stochastic and heterogeneous (Aouad, 2020).
  - Heterogeneous supply and demand types, where demand nodes should be matched immediately, but supply nodes can wait for matching until they depart (Özkan, 2020).
  - Unmatched demand and supply will incur waiting or holding costs, and will be carried over to the next period with abandonment (Hu, 2021).
  - Develop a fluid model that approximates the evolution of the stochastic model (Aveklouris, 2021)

❑ Other related work

- Online bipartite allocations
  - RANKING algorithm to the setting where offline vertices are weighted (Aggarwal 2011, Devanur 2013).
  - Inspired by BALANCE algorithm (Golrezaei 2014).
- Other continuous-time dynamic stochastic matching
  - Vertex arrivals and departures are stochastic and heterogeneous (Aouad, 2020).
  - Heterogeneous supply and demand types, where demand nodes should be matched immediately, but supply nodes can wait for matching until they depart (Özkan, 2020).
  - Unmatched demand and supply will incur waiting or holding costs, and will be carried over to the next period with abandonment (Hu, 2021).
  - Develop a fluid model that approximates the evolution of the stochastic model (Aveklouris, 2021)
- Adversarial setting (Emek, 2016; Azar 2018)

❑ Other related work

- Online bipartite allocations
  - RANKING algorithm to the setting where offline vertices are weighted (Aggarwal 2011, Devanur 2013).
  - Inspired by BALANCE algorithm (Golrezaei 2014).
- Other continuous-time dynamic stochastic matching
  - Vertex arrivals and departures are stochastic and heterogeneous (Aouad, 2020).
  - Heterogeneous supply and demand types, where demand nodes should be matched immediately, but supply nodes can wait for matching until they depart (Özkan, 2020).
  - Unmatched demand and supply will incur waiting or holding costs, and will be carried over to the next period with abandonment (Hu, 2021).
  - Develop a fluid model that approximates the evolution of the stochastic model (Aveklouris, 2021)
- Adversarial setting (Emek, 2016; Azar 2018)
- Extend two stage to K-stage (Feng, 2020)

❑ Other related work

- Online bipartite allocations
  - RANKING algorithm to the setting where offline vertices are weighted (Aggarwal 2011, Devanur 2013).
  - Inspired by BALANCE algorithm (Golrezaei 2014).
- Other continuous-time dynamic stochastic matching
  - Vertex arrivals and departures are stochastic and heterogeneous (Aouad, 2020).
  - Heterogeneous supply and demand types, where demand nodes should be matched immediately, but supply nodes can wait for matching until they depart (Özkan, 2020).
  - Unmatched demand and supply will incur waiting or holding costs, and will be carried over to the next period with abandonment (Hu, 2021).
  - Develop a fluid model that approximates the evolution of the stochastic model (Aveklouris, 2021)
- Adversarial setting (Emek, 2016; Azar 2018)
- Extend two stage to K-stage (Feng, 2020)
- Stable matching, e.g., leveraging Gale-Shapley stable marriage problem. (Wang 2018, Fajardo-Delgado 2022).

…

❑ Other related work

- Online bipartite allocations
  - RANKING algorithm to the setting where offline vertices are weighted (Aggarwal 2011, Devanur 2013).
  - Inspired by BALANCE algorithm (Golrezaei 2014).
- Other continuous-time dynamic stochastic matching
  - immediately, but supply nodes can wait for matching until they depart (Özkan, 2020).
  - Unmatched demand and supply will incur waiting or holding costs, and will be carried over to the next period with abandonment (Hu, 2021).
  - Develop a fluid model that approximates the evolution of the stochastic model (Aveklouris, 2021)
- Adversarial setting (Emek, 2016; Azar 2018)
- Extend two stage to K-stage (Feng, 2020)
- Stable matching, e.g., leveraging Gale-Shapley stable marriage problem. (Wang 2018, Fajardo-Delgado 2022).

…

Any question?