Edge Weighted Online Windowed Matching

Itai Ashlagi Maximilien Burq Chinmoy Dutta Patrick Jaillet
Chris Sholley Amin Saberi*

Abstract

Motivated by applications from ride-sharing and kidney exchange, we study the problem of matching agents who arrive at a marketplace over time and leave after d time periods. Agents can only be matched while they are present in the marketplace. Each pair of agents can yield a different match value, and the planner's goal is to maximize the total value over a finite time horizon.

First we study the case in which vertices arrive in an adversarial order. We provide a randomized $1/4$-competitive algorithm building on a result by Feldman et al. [2009b] and Lehmann et al. [2006]. We extend the model to the case in which departure times are drawn independently from a distribution with non-decreasing hazard rate, for which we establish a $1/8$-competitive algorithm.

When the arrival order is chosen uniformly at random, we show that a batching algorithm, which computes a maximum-weighted matching every $(d+1)$ periods, is 0.279-competitive.

1 Introduction

Traffic congestion is a severe problem in metropolitan areas around the world. A resident in Los Angeles is estimated to lose around $6,000 per year due to spending extra hours in traffic (Economist 2014). A couple of ways to relieve congestion are pricing [Vickrey, 1965] and carpooling, and online platforms and other technological advances are now available to assist with these tasks [Ostrovsky and Schwarz, 2018].[1]

Online ride-sharing platforms now offer the option to share rides which has an immediate impact on reducing congestion. Passengers who share rides also share the cost thereby paying a lower price for their rides and can further use high occupancy lanes. However, a passenger may also experience additional waiting, detours, and less privacy. Facing these trade-offs, ride-sharing platforms seek to increase the volume of shared rides, which will in turn help in reducing congestion.

This paper is concerned with the problem of matching passengers for sharing rides (carpooling in the context of an on-demand ridesharing platform where passengers request rides in real-time and are willing to share the ride with other passengers). We present and study a stylized graph-theoretic matching problem that captures the following three key features faced by ride-sharing platforms. First is spatial; the farther away two passengers' pickup locations or the more mismatch

*Ashlagi: Stanford, iashlagi@stanford.edu. Burq: MIT, mburq@mit.edu. Dutta: Lyft, cdutta@lyft.com. Jaillet: MIT, jaillet@mit.edu. Saberi: Stanford, saberi@stanford.edu, Shollet: Lyft, chris@lyft.com

[1]See Ostrovsky and Schwarz [2018] for a discussion about the complementarities between autonomous vehicles, carpooling and pricing.

between their routes, the higher the disutility from being matched. Second is temporal; ride-sharing platforms offer passengers the option of waiting for a few minutes, during which time they are eligible to be matched to one another. Third, the platform faces uncertainty about future demand, and needs to commit to matching decisions within the waiting period that the passengers have agreed to with imperfect knowledge of future matching opportunities.

The spatial aspect is captured by modeling passengers as vertices of a graph. Time is discrete and one vertex arrives at each time period. An edge of the graph has a non-negative weight which is the utility from matching the two passenger-vertices. A vertex cannot match more than d periods after its arrival; after d units of time the vertex becomes *critical* and departs. It is useful to interpret d as a service quality set by the platform: it is the maximum waiting period for a passenger to get matched and if the platform is unable to match a given passenger within d units of time since arrival, then the passenger must be assigned to a single ride.

The goal is to find a weighted matching with a large total weight in an online manner. This means that the decision for every vertex has to be made no later than d periods after its arrival (this differs from the classic online bipartite matching literature, in which $d = 0$). There is no a priori information about edge weights and the underlying graph may be arbitrary and hence *non-bipartite*.

Contributions

Our first results are given for the setting in which the vertices arrive in an **adversarial order**. We introduce for this setting a $1/4$-competitive algorithm, termed *Postponed Greedy* (PG). We further show that no algorithm achieves a competitive ratio that is higher than $1/2$.

The key idea behind PG is to look at a virtual bipartite graph, in which each vertex is duplicated into a "buyer" and a "seller" copy. We enforce that the seller copy does not match before the vertex becomes critical. This enables us to postpone the matching decision until we have more information about the graph structure and the likely matchings. We then proceed in a manner similar to Feldman et al. [2009b]: tentatively match each new buyer copy to the seller that maximizes its margin, i.e., the difference between the weight of the edge with the seller and the value of the seller's current match.

We extend the model to the case where the departure of vertices are determined stochastically. We show that when the departure distribution is memoryless and realized departure times are revealed to the algorithm just when becoming critical, one can adapt the PG algorithm to achieve a competitive ratio of $1/8$. It is worth noting that when departure times are chosen in an adversarial manner no algorithm can achieve a constant competitive ratio.

Next we study the setting, in which vertices arrive in a **random order**. We analyze a *batching* algorithm which, every $d + 1$ time steps, computes a maximum weighted matching among the last $d + 1$ arrivals (batching-like algorithms are commonly used in ride-sharing platforms).[2] Vertices that are left unmatched are discarded forever (again, one can interpret discards as single rides). We show that when the number of vertices is sufficiently large, batching is 0.279-competitive.

The analysis proceeds in three steps. First, we show that the competitive ratio is bounded by the solution to a graph covering problem. Second, we show how a solution for small graphs can be extended to covers for larger graphs. Finally, we establish a reduction that allows us to consider only a finite set of values for d. We conclude with a computer-aided argument for graphs in the

[2]Batching is also used in many kidney exchange platforms (from personal communication).

finite family.

Related literature

There is a growing literature related to ride-sharing. Santi et al. [2014] finds that about 80% of rides in Manhattan could be shared by two passengers. Many studies focus on rebalancing or dispatching problems without pooling, e.g., Pavone et al. [2012], Zhang and Pavone [2014], Santi et al. [2014], Spieser et al. [2016], Banerjee et al. [2018]. Alonso-Mora et al. [2017] studies real-time high-capacity ride-sharing. They do not consider, however, a graph-theoretic online formulation for matching rides.

This paper is closely related to the online matching literature. In the classic problem, introduced in Karp et al. [1990], the graph is bipartite with vertices on one side waiting, while others are arriving sequentially and have to be matched *immediately* upon arrival. This work has numerous extensions, for example to stochastic arrivals and in the adwords context Mehta et al. [2007], Goel and Mehta [2008], Feldman et al. [2009a], Manshadi et al. [2011], Jaillet and Lu [2013]. See Mehta [2013] for a detailed survey. Our work contributes to this literature in three ways. First and foremost, our graph can be non-bipartite, which is the case in applications such as ride-sharing and kidney exchange. Second, all vertices arrive over time and remain for some given time until they are matched or hit their deadline and depart. Third, we provide algorithms that perform well on edge-weighted graphs. Closely related is Huang et al. [2018, 2019], which studies a similar model to ours in the non-weighted case, but allow departure times to be adversarial.

Several papers consider the problem of dynamic matching in the edge-weighted case. Feldman et al. [2009b] find that in the classic online bipartite setting, no algorithm achieves a constant approximation. They introduce a *free disposal* assumption, which allows to discard a matched vertex in favor of a new arriving vertex. They find, based on an algorithm by Lehmann et al. [2006], that a greedy algorithm that matches a vertex to the highest marginal vertex, is 0.5-competitive. We build on this result for a special class of bipartite graphs. In the adversarial setting, Emek et al. [2016], Ashlagi et al. [2017a] study the problem of minimizing the sum of distances between matched vertices and the sum of their waiting times. In their model no vertex leaves unmatched. Few papers consider the stochastic setting [Baccara et al., 2015, Ozkan and Ward, 2016, Hu and Zhou, 2016]. These papers find that some waiting before matching is beneficial for improving efficiency.

Related to our work are some works on job or packet scheduling. Jobs arrive online to a buffer, and reveal upon arrival the deadline by which they need to be scheduled. The algorithm can schedule at most one job per time and the value of scheduling a job is independent of the time slot. Constant approximation algorithms are given by Chin et al. [2006] and Li et al. [2005].

Finally, there is a growing literature that focuses on dynamic matching motivated by kidney exchange [Ünver, 2010, Anderson et al., 2015, Dickerson et al., 2013, Ashlagi et al., 2017b]. These papers focus mostly on random graphs with no weights. Closer to our paper is Akbarpour et al. [2017], which finds that in a sparse random graph, knowledge about the departure time of a vertex is beneficial and matching a vertex only when it becomes critical performs well. Our work differs from these papers in two ways: we consider the edge-weighted case, and, we make no assumption on the graph structure.

2 Model

Consider a weighted graph G with n vertices indexed by $i = 1, \dots n$. Vertices arrive sequentially over n periods and let $\sigma(i)$ denote the arrival time of vertex i. Let $v_{ij} \geq 0$ denote the weight on the undirected edge (i, j) between vertices i and j.

For vertices i and j with $\sigma(i) < \sigma(j)$, the weight v_{ij} on the edge between i and j is observed only after vertex j has arrived.

For $d \geq 1$, the **online graph with deadline** d, denoted by $G_{d,\sigma}$, has the same vertices as G, and the edge between i and j in G exists if an only if $|\sigma(i) - \sigma(j)| \leq d$. We say that i becomes **critical** at time $\sigma(i) + d$, at which time the online algorithm needs to either match it and collect the associated edge weight, or let it remain unmatched.

We will consider two settings regarding how arrivals are generated. In the Adversarial Order (AO) setting, we assume that $\sigma(i) = i$. In the Random Order (RO) setting, we assume that σ is sampled uniformly at random among all possible permutations S_n of $[1, n]$.

The goal is to find an online algorithm that generates a matching with high total weight. More precisely, we seek to design a randomized online algorithm that obtains in expectation a high fraction of the expected maximum-weight of a matching over $G_{d,\sigma}$.

To illustrate a natural tradeoff, consider the example in Figure 1 for $d = 1$. At time 2, the planner can either match vertices 1 and 2 or let vertex 1 remain unmatched. This simple example shows that no deterministic algorithm can obtain a constant competitive ratio. Furthermore, no algorithm can achieve a competitive ratio higher than $1/2$.

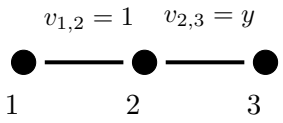

Figure 1: Let $d = 1$. Therefore, there is no edge between vertices 1 and 3. The algorithm needs to decide whether to match 1 with 2 and collect $v_{1,2}$ without knowing y.

3 Adversarial order (AO) of arrivals

The example in Figure 1 illustrates a necessary condition for the algorithm to achieve a constant competitive ratio: with some probability, vertex 2 needs to forgo the match with vertex 1. We ensure this property by assigning every vertex to be either a *seller* or a *buyer*. We then prevent sellers from matching before they become critical, while we allow buyers to be matched at any time.

It will be useful to first study a special case, in which the underlying graph G is bipartite, with sellers on one side and buyers on the other, and in the online graph a buyer and a seller cannot match if the buyer arrives before the seller. For such online graphs we show that a greedy algorithm given by Feldman et al. [2009b] is 0.5-competitive. We then build on this algorithm to design a randomized $1/4$-competitive algorithm for arbitrary graphs.

3.1 Bipartite constrained online graphs

Let G be a bipartite graph and σ be the order of arrivals. The online graph $G_{d,\sigma}$ is called **constrained bipartite** if for every seller s and every buyer b, there is no edge between s and b if $\sigma(b) < \sigma(s)$, i.e. b and s cannot match if b arrives before s.

Consider the following greedy algorithm, which attempts to match buyers in their arriving order. An arriving buyer b is matched to the seller with the highest marginal value if the marginal value is positive. If the seller is already matched to another buyer b', b' becomes unmatched and never gets matched again. Formally:

Algorithm 1 Greedy algorithm [Feldman et al., 2009b]

- Input: constrained bipartite graph, $G_{d,\sigma}$.
- For each arrival $i = 1, \ldots, n$:
  - If i is a seller, initialize $p(i) = 0$, and $m(s) = null$.
  - If i is a buyer:
    - Set $s = \text{argmax}_{s' \in S}\{v_{is'} - p(s')\}$, where S denote the set of sellers already arrived.
    - If $v_{is} - p(s) > 0$, set $m(s) = i$ and set $p(s) = v_{is}$.
- When a seller s becomes critical: match it to $b = m(s)$ if $m(s) \neq null$.

[Feldman et al. [2009b]] The greedy algorithm is 0.5-competitive for online bipartite constrained graphs.

Feldman et al. [2009b] prove that this algorithm is 0.5-competitive for an online matching problem with *free disposal*. In their setting all seller exists and buyer arrive one at a time. The algorithm provides the same guarantees for constrained bipartite graph since, by construction, there is no harm in assuming that all sellers exist rather than arriving over time. The key behind the proof is that the value $p(s)$ function for each seller s is submodular. In fact the result is a special case of a result by Lehmann et al. [2006], who study combinatorial auctions with submodular valuations.

3.2 Arbitrary graphs

In this section we extend the greedy algorithm for constrained bipartite graphs to arbitrary graphs. A naive way to generate a online constrained bipartite graph from an arbitrary one is to randomly assign each vertex to be either a seller or a buyer, independently and with probability $1/2$. Then only keep the edges between each buyer and all the sellers who arrived before her. Formally:

The naive greedy algorithm is $1/8$-competitive for arbitrary online graphs.

Observe that for vertices i, j with $\sigma(i) < \sigma(j)$, edge (i, j) in the original graph remains in the generated constrained bipartite graph with probability $1/4$ (if i is a seller and j is a buyer). We then use proposition 3.1 to prove that naive greedy is $1/8$-competitive.

One source of inefficiency in the naive greedy algorithm is that the decision whether a vertex becomes a seller or a buyer is done independently at random and without taking the graph structure into consideration. We next introduce the *Postponed Greedy* algorithm that defers these decisions as long as possible in order to construct the constrained bipartite graph more carefully.

Algorithm 2 Naive Greedy

- Input: an online graph with deadline d, $G_{d,\sigma}$.
- For each vertex $t = 1, \ldots, n$:

  Toss a fair coin to decide whether i is a *seller* or a *buyer*. Construct the online constrained bipartite graph $\tilde{G}(d, \sigma)$ by keeping only the edges between each buyer and the sellers who arrived before her.

- Run the Greedy algorithm on $\tilde{G}(d, \sigma)$.

When a vertex k arrives, we add two copies of k to a virtual graph: a seller s_k and a buyer b_k. Let S_t and B_t be the set of sellers and buyers at arrival time t. On arrival, seller s_k does not have any edges, and buyer b_k has edges towards any vertex $s_l \in S_k$ with value $v_{l,k}$. Then we run the greedy algorithm with the virtual graph as input. When a vertex k becomes critical, s_k becomes critical in the virtual graph, and we compute its matches generated by greedy.

Both s_k and b_k can be matched in this process. If we were to honor both matches, the outcome would correspond to a 2-matching, in which each vertex has degree at most 2. Now observe that because of the structure of the constrained bipartite graph, this 2-matching does not have any cycles; it is just a collection of disjoint paths. We decompose each path into two disjoint matchings and choose each matching with probability 1/2.

In order to do that, the algorithm must determine, for each original vertex k, whether the virtual buyer b_k or virtual seller s_k will be used in the final matching. We say that k is a *buyer* or *seller* depending on which copy is used. We say that vertex k is *undetermined* when the algorithm has not yet determined which virtual vertex will be used. When an undetermined vertex becomes critical, the algorithm flips a fair coin to decide whether to match according to the buyer or seller copy. This decision is then propagated to the next vertex in the 2-matching: if k is a *seller* then the next vertex will be a *buyer* and vice-versa. That ensures that assignments are correlated and saves a factor 2 compared to uncorrelated assignments in the naive greedy algorithm.

Theorem 3.1. *The postponed greedy (PG) algorithm is* $1/4$*-competitive for arbitrary online graphs.*

Proof. Fix a vertex k, and denote $p^f(s_k)$ to be the final value of its virtual seller s_k's match. If k's status is a seller in step (2.c.i), then we collect $p^f(s_k)$. Note that this happens with probability exactly $1/2$ for every k.

$$\text{PG} = \mathbb{E}\left[\sum_{k \text{ is a seller}} p^f(s_k)\right] = \frac{1}{2}\sum_{k\in[1,T]} p^f(s_k).$$

For a virtual buyer b arriving at time t, let $q(b) = \max_{s\in S_t} v_{sb} - p(s)$ be the margin for b in step (1.d). Note that every increase in a virtual seller's price corresponds to a virtual buyer's margin. Using the notation $S = \cup_t S_t$ and $B = \cup_t B_t$, this implies that $\sum_{s\in S} p^f(s) = \sum_{b\in B} q(b)$.

The dual of the offline matching problem linear programs can be written as:

Algorithm 3 Postponed Greedy (PG)

- Input: an online graph with deadline d, $G_{d,\sigma}$.
- Process events at time t in the following way:
  1. *Arrival of a vertex k:*
     (a) Set k's status to be *undetermined*.
     (b) *Add a virtual seller:* $S_t \leftarrow S_{t-1} \cup \{s_k\}$, $p(s_k) \leftarrow 0$ and $m(s_k) = \emptyset$.
     (c) *Add a virtual buyer:* $B_t \leftarrow B_{t-1} \cup \{b_k\}$.
     (d) *Find a virtual seller for the virtual buyer:* $s = \text{argmax}_{s' \in S_t} v_{s',b_k} - p(s')$.
     (e) *Match if marginal utility is positive:* If $v_{s,b_k} - p(s) > 0$, then tentatively match b_k to s by setting $m(s) \leftarrow b_k$ and $p(s) \leftarrow v_{s,b_k}$.
  2. *Vertex k becomes critical:*
     (a) *Proceed if no match found:* If $m(s_k) = \emptyset$, do nothing.
     (b) *Match in the virtual graph:* If $m(s_k) = b_l$. Set $S_t \leftarrow S_t \setminus \{s_k\}$, and $B_t \leftarrow B_t \setminus \{b_l\}$.
     (c) If k's status is *undetermined*, w.p $1/2$ set it to be either *seller* or *buyer*.
        i. *If k is a seller:* finalize the matching of k to l and collect the reward $v_{k,l}$. Set l's status to be a buyer.
        ii. *If k is a buyer:* Set l's status to be a seller.

$$\begin{aligned} \text{minimize} \quad & \sum_{k \in [1,T]} \lambda_k \\ \text{subject to} \quad & v_{kl} \leq \lambda_k + \lambda_l \quad \forall (k,l) \text{ s.t. } |k-l| \leq d \\ & \lambda_k \geq 0. \end{aligned} \tag{Offline Dual}$$

Let i and $j > i$ be two vertices with $j - i \leq d$. When j arrives, we have $q(b_j) \geq v_{ij} - p(s_i)$. Together with the fact that $p(s)$ increases over time, this implies that $\{p^f(s_k) + q(b_k)\}_{k \in [1,T]}$ is a feasible solution to (Offline Dual).

We can conclude that OFF $\leq \sum_k p^f(s_k) + q(b_k) = 2 \sum_k p^f(s_k) = 4\text{PG}$.

□

3.3 Lower bounds

Claim 3.2. *When the input is a constrained bipartite graph:*

- *No deterministic algorithm can obtain a competitive ratio above* $\frac{\sqrt{5}-1}{2} \approx 0.618$.
- *No randomized algorithm can obtain a competitive ratio above* $\frac{4}{5}$.

Proof. **Deterministic algorithm:** Consider the example on the left of Figure 2. When seller 1 becomes critical, the algorithm either matches her with buyer 3, or lets 1 departs unmatched. The

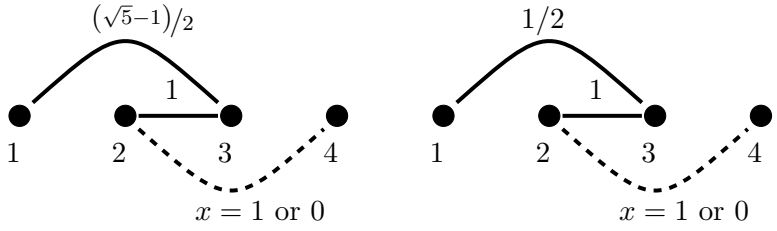

Figure 2: Bipartite graph where $S = \{1, 2\}$ and $B = \{3, 4\}$, with $d = 2$: vertex 1 becomes critical before 4 arrives. The adversary is allowed to choose edge $(2, 4)$ to be either 1 or 0. Left: instance for the deterministic case. Right: instance for the randomized case.

adversary then chooses x accordingly. Thus the competitive ratio cannot exceed:

$$\max\left(\min_{x\in\{0,1\}} \frac{\frac{\sqrt{5}-1}{2}+x}{\rho(x)}, \min_{x\in\{0,1\}} \frac{1}{\rho(x)}\right) = \frac{\sqrt{5}-1}{2}.$$

Where $\rho(x) = \max(\frac{\sqrt{5}-1}{2} + x, 1)$.

Randomized algorithm: Consider the example on the right of Figure 2. Similarly to the deterministic case, when seller 1 becomes critical, the algorithm decides to match her with 3 with probability p. The adversary then chooses x accordingly. Thus the competitive ratio cannot exceed:

$$\max_{p\in[0,1]} \min_{x\in\{0,1\}} \frac{p(1/2+x)+(1-p)}{\max(1/2+x, 1)} = 4/5.$$

□

Next we show that our analysis for PG is tight.

Claim 3.3. *There exists a* constrained *bipartite graph for which PG is* $1/(4-2\epsilon)$ *-competitive.*

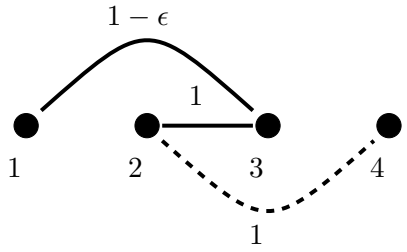

Figure 3: Bipartite graph where $S = \{1, 2\}$ and $B = \{3, 4\}$, with $d = 2$: vertex 1 becomes critical before 4 arrives. Dotted edges represent edges that are not know to the algorithm initially.

Proof. Consider the input graph in Figure 3. Vertex 2 will be temporarily matched with 3, and vertex 1 will depart unmatched. When 2 becomes critical, with probability 1/2, she will be determined to be a *buyer* and will depart unmatched. Therefore the PG collects in expectation 1/2 while the offline algorithm collects $2 - \epsilon$.

□

4 Random permutation (RP) of arrivals

In some cases, the vertices can be assumed to come from a distribution that is unknown to the online algorithm. One way to model this is to assume that the adversary chooses the underlying graph, but that the vertices arrive in random order.

4.1 The batching algorithm

The *batching* algorithm computes a maximum-weight matching every $d+1$ time steps. Every vertex in the matching is then matched, and all other vertices in the batch are discarded.

Theorem 4.1. *Batching is* $(0.279 + O(1/n))$*-competitive.*

The proof of Theorem 4.1 works in three steps. In a first step, we reduce the analysis of the competitive ratio of *Batching* to a graph covering problem. More precisely, we show that it is enough to cover C_n^d, the cycle with n vertices to the power d, with ensembles of cliques. Second, we show how a cover for small n can be extended to any n at the cost of a small rounding error. Finally, we establish a reduction that allows us to consider only a finite set of values for d. We conclude with a computer-aided argument for graphs in the finite family.

4.1.1 Reducing to a graph theoretic problem

There is no harm in assuming that the underlying graph G is a complete. Recall that S_n is the set of all permutations over integers $1, ..., n$. For any deadline d and any arrival sequence $\sigma \in S_n$, we define the *path* graph $P_n^d(\sigma)$ with edge-weight $v_{ij} = 1$ if $|\sigma(i) - \sigma(j)| \leq d$, and $v_{ij} = 0$ otherwise.[3]

Note that every batch in the algorithm has $d + 1$ vertices except the last batch which may have fewer vertices. Let $b_i(\sigma, d)$ be the batch of vertex i under permutation σ and batch size $d+1$: $b_i(\sigma, d)$ is the unique integer such that $(d+1)(b_i-1) < \sigma(i) \leq (d+1)b_i$. We define the *batched* graph $B_n^d(\sigma)$ with edge-weight $v_{ij} = 1$ if i and j are in the same batch (i.e. $b_i(\sigma, d+1) = b_j(\sigma, d+1)$), and $v_{ij} = 0$ otherwise.[4]

For any $n \geq d \geq 1$, denote C_n^d to be the n-cycle to the power d.

[Graph operations] For any two graphs H and H' with vertices $1, ..., n$ and respective edge weights v_{ij}, v'_{ij}, we define the following:

(i) The linear combination $aH + bH'$ denotes the graph with edge weights $av_{ij} + bv'_{ij}$,

(ii) The product $H * H'$ denotes the graph with edge weights $v_{ij} * v'_{ij}$, and

(iii) We say that H is a *cover* of H' if for all i, j, $v_{i,j} \geq v'_{ij}$.

[3] Note that $P_n^d(\sigma)$ corresponds to the path $(\sigma(1), \sigma(2)), (\sigma(2), \sigma(3)), ..., (\sigma(n-1), \sigma(n))$ taken to the power d.

[4] Note that $B_n^d(\sigma)$ is a collection of disjoint $(d+1)$-cliques.

For any graph H, let $m(H)$ denote the value of a maximum-weight matching over H. Observe that when the arrival sequence is σ, the graph $P_n^d(\sigma) * G = G(d, \sigma)$ and therefore the offline algorithm collects $m(P_n^d(\sigma) * G)$. Note that the online algorithm collects $m(B_n^d(\sigma) * G)$.

Remark 4.2. *Observe that for any graphs H, H', G and any $a, b \in \mathbb{R}$, we have:*

- $m(aH + bH') \leq am(H) + bm(H')$.
- *If H is a cover of H', then, $m(H * G) \geq m(H' * G)$.*

[Periodic permutation] For $p < n$ such that p divides n, we say that a permutation $\sigma \in S_n$ is p-periodic if for all $i \in [1, n-p]$, $\sigma(i+p) \equiv \sigma(i) + p \mod n$.

We say that a permutation σ is periodic if there exists p such that σ is p-periodic.

[(α, d)-cover] Let F be an unweighted graph with n vertices. We say that a set of permutations $\{\sigma_1, ..., \sigma_K\} \in S_n$ forms an (α, d)-cover of F if there exist values $\lambda_1, ..., \lambda_K \in [0, 1]$ such that:

(i) $\sum_{k \leq K} \lambda_k B_n^d(\sigma_k)$ is a cover of F.

(ii) $\sum_{k \leq K} \lambda_k = \alpha$.

We say that an (α, d)-cover is p-periodic if for all k, σ_k is p-periodic.

The next proposition will allow us to abstract away from the weights that are chosen by the adversary. For any graph H, we denote by H_{ij} the weight v_{ij} in H. If there exists an (α, d)-cover of C_n^d, then batching is $1/\alpha$-competitive.

Proof. Let *id* be the identity permutation over n vertices. Let $\{\sigma_1, ..., \sigma_K\}$ be an (α, d)-cover of C_n^d. Fix an arrival sequence $\sigma \in S_n$. We first claim that $\{\sigma_1 \circ \sigma, ..., \sigma_K \circ \sigma\}$ is an (α, d)-cover of $P_n^d(\sigma)$.

For any $\sigma \in S_n$, let us denote $\beta_{i,j}(\sigma)$ and $\rho_{i,j}(\sigma)$ to be the weights of edge (i, j) in $B_n^d(\sigma)$ and $P_n^d(\sigma)$ respectively. Consider $(i, j) \in P_n^d(\sigma)$: $|\sigma(i) - \sigma(j)| \leq d$:

$$\begin{aligned}
\sum_k \lambda_k \beta_{i,j}(\sigma_k \circ \sigma) &= \sum_k \lambda_k \mathbb{I}[b_i(\sigma_k \circ \sigma, d) = b_j(\sigma_k \circ \sigma, d)] \\
&= \sum_k \lambda_k \mathbb{I}[b_{\sigma(i)}(\sigma_k, d) = b_{\sigma(j)}(\sigma_k, d)] \\
&\geq \rho(\mathrm{id})_{\sigma(i),\sigma(j)} = 1,
\end{aligned}$$

where the last inequality is implied by the fact that $\{\sigma_1, ..., \sigma_K\}$ is an (α, d)-cover of C_n^d and therefore of $P_n^d(\mathrm{id})$. Therefore the claim holds using remark 4.2.

Denote by BAT the value collected by the batching algorithm and OFF the value collected by the offline algorithm. Observe that

$$\begin{aligned}
\mathrm{OFF} &= \frac{1}{n!} \sum_{\sigma \in S_n} m(P_n^d(\sigma) * G) \\
&\leq \frac{1}{n!} \sum_{\sigma \in S_n} \sum_k \lambda_k m(B_n^d(\sigma_k \circ \sigma) * G) \\
&= \frac{1}{n!} \sum_k \lambda_k \sum_{\sigma' \in S_n} m(B_n^d(\sigma') * G) \\
&= \alpha \mathrm{BAT},
\end{aligned}$$

where we used the change of variable $\sigma' = \sigma_k \circ \sigma$ and the fact that the application $\mathcal{A}_k : \sigma \mapsto \sigma_k \circ \sigma$ is a bijection.

□

We have reduced the analysis of Batching to a graph-theoretic problem without edge weights. In what follows, we will show that we can reduce the problem further to find covers of C_n^d for only small values of n and d.

4.1.2 Reducing n: periodic covers.

We now wish to find (α, d)-covers for C_n^d for every n and d. In Proposition 4.1.2, we show that it is sufficient to find periodic covers for small values of n.

Let p be a multiple of $d+1$, and n_1 a multiple of p. Any p-periodic (α, d)-cover of $C_{n_1}^d$ can be extended into an $(\alpha + O(p/n), d)$-cover of C_n^d for any $n \geq n_1$.

Proof when n is a multiple of p. Let $\{\sigma_1, ..., \sigma_K\}$ be a p-periodic (α, d)-cover of $C_{n_1}^d$. We will show that it can be extended into an (α, d)-cover of C_n^d.

Assume for now that n is a multiple of p. Let σ'_k be the p-periodic permutation over $1, ..., n$ such that for all $i \in [1, p]$, $\sigma'_k(i) = \sigma_k(i)$. Take $i', j' \in [1, n]$ such that $|i' - j'| \leq d$. Because $n_1 > p$ is a multiple of p, there exist $i, j \in [1, n_1]$ such that $i \equiv i' \mod p$, $j \equiv j' \mod p$ and $|i - j| \leq d$. By p-periodicity of σ_k and σ'_k, we know that $B_n^d(\sigma'_k)_{i',j'} = B_{n_1}^d(\sigma_k)_{i,j}$. Thus we can conclude that $\{\sigma'_1, ..., \sigma'_K\}$ is an (α, d)-cover of C_n^d. □

Proof in the general case. When n is not a multiple of p, let $v \in [1, p-1]$ be the remainder of the euclidian division of n by p, and u be such that $n = pu + v$. Let $\{\sigma_1, ..., \sigma_K\}$ be a p-periodic (α, d)-cover of $C_{n_1}^d$ with associated weights $\{\lambda_1, ..., \lambda_K\}$. We will show that it can be extended into an $\left(\alpha \left(u/u-2\right), d\right)$-cover of C_n^d.

We set $\widetilde{\sigma}_k$ to be the p-periodic permutation over $1, ..., pu$ such that for all $i \in [1, p]$, $\widetilde{\sigma}_k(i) = \sigma_k(i)$. Let x be an integer in the interval $[1, u+1]$. Define the permutation $\sigma'_{k,x}$ as follows:

$$\sigma'_{k,x}(i) = \begin{cases} \widetilde{\sigma}_k(i) & i \leq px \\ i + (u-x)p & i \in [px+1, px+1+v] \\ \widetilde{\sigma}_k(i-v) & i > px+1+v. \end{cases} \tag{1}$$

Take $i', j' \in [1, px] \cup [px+1+v, n]$ such that $|i' - j'| \leq d$. Because $n_1 > p$ is a multiple of p, there exist $i, j \in [1, n_1]$ such that $i \equiv i' \mod p$, $j \equiv j' \mod p$ and $|i - j| \leq d$. By p-periodicity of σ_k and σ'_k, we know that edge (i', j') is in $B_n^d(\sigma'_k)$ iff (i, j) is in $B_{n_1}^d(\sigma_k)$. Thus we can conclude that $\sum_k \lambda_k B_n^d(\sigma'_{k,x})$ covers edge (i', j') of C_n^d.

Every edge is therefore covered for at least $u-2$ different values of x. Therefore, $\sum_k \sum_x \frac{u}{u-2} \lambda_k B_n^d(\sigma'_{k,x})$ covers C_n^d. This means that $(\sigma_{k,x})_{k,x}$ is an $\left(\alpha \left(u/u-2\right), d\right)$-cover of C_n^d.

□

4.1.3 Reducing d: cycle contraction.

In Proposition 4.1.2, we show that it is enough to find periodic (α, d)-covers of C_n^d for small values of n. Next, we provide a reduction that enables us to consider only a finite set of values for d.

The key idea of the reduction is that we can contract vertices of C_n^d into n/u groups of u vertices. The resulting graph also happens to be a cycle $C_{n/u}^{(d+1)/u}$. In Proposition 4.1.3, we provide a way to expand an $(\alpha, u-1)$-cover on the contracted graph into an (α, d) cover on the original graph.

[Cycle contraction] For any n, d and an integer u which divides n, we define the u-contraction $f_u(C_n^d)$ to be the graph with vertices $a_k = \{uk+1, ..., u(k+1)\}$ for $k \in [0, {}^n/_u - 1]$, and edges (a_k, a_l) if and only if there exist $i \in a_k$ and $j \in a_l$ with an edge (i, j) in C_n^d.

Claim 4.3. *For any d, if $u > 1$ divides $d+1$ and $d+1$ divides n, then $f_u(C_n^d) = C_{n/u}^{(d+1)/u}$.*

Proof. We first prove that $C_{n/u}^{(d+1)/u}$ covers $f_k(C_n^d)$. Fix $k, l \in [0, {}^n/_u - 1]$, and assume that $k < l$. If $|l-k| \leq (d+1)/u$, then let $i = u(k+1)$ and $j = ul+1$. We have $|j-i| = u(l-k-1)+1 \leq d$, thus $(i,j) \in C_n^d$ and $(k,l) \in f_u(C_n^d)$.

Conversely, we now prove that $f_u(C_n^d)$ covers $C_{n/u}^{(d+1)/u}$. If there exist $i \in a_k$ and $j \in a_l$ such that $|j-i| \leq d$, then $u(l-k) \leq ul+1-u(k+1) \leq d+1$ which implies that $(k,l) \in C_{n/u}^{(d+1)/u}$. □

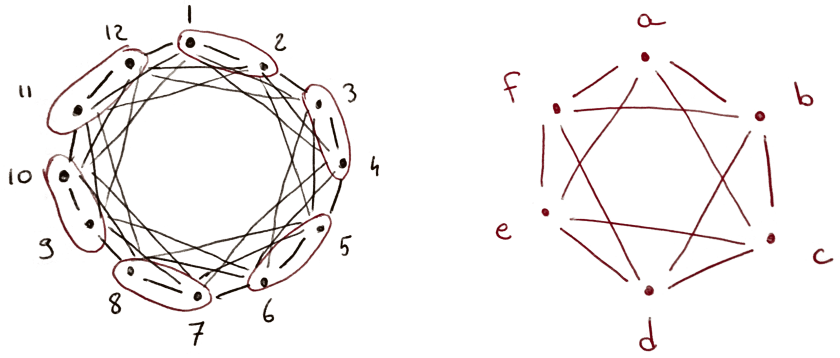

Figure 4: Left: C_{12}^3, with contraction for $u = 2$. Right: Contracted graph $f(C_{12}^3) = C_6^2$ with vertices $a = \{1,2\}$, $b = \{3,4\}$, ... $f = \{11,12\}$.

Fix $d \geq 1$. For $d+1 > k \geq 1$, suppose that there is a periodic $(\alpha, k-1)$-cover of C_{rk}^k.

(i) For any integer r, if k divides $d+1$ then there exists a periodic (α, d)-cover of $C_{r(d+1)}^d$.

(ii) In general, if v is the remainder of the euclidian division of $d+1$ by k, then there exists a periodic $\left(\alpha(1+{}^v/_{d+1-v})^2, d\right)$-cover of $C_{r(d+1)}^d$.

Proof of (i). Suppose that $d+1 = ku$ and suppose that there exists p multiple of $d+1$ such that we have a p-periodic $(\alpha, k-1)$-cover $\{\sigma_1, ..., \sigma_K\}$ of $f_u\left(C_{r(d+1)}^d\right) = C_{rk}^k$. For any permutation $\sigma \in S_{rk}$ we can construct a permutation $\sigma' \in S_{r(d+1)}$ in the following way: if $i \in a_t$ then $\sigma'(i) = \frac{n}{k}\sigma(t) + i$. Because $B_{rk}^k(\sigma_i)$ is a cover of $B_{r(d+1)}^d$, we can conclude that $\sigma'_1, ..., \sigma'_K$ is an (α, d)-cover of $C_{r(d+1)}^d$. □

Proof of (ii). Suppose now that $d+1 = ku+v$ with $1 \leq v < k$. We first select vertices in the following way: select a subset $\Phi \subset [1, d+1]$ of $d+1-v$ vertices uniformly at random. Take $\Delta = \Phi + ku[1, r-1] = \{a + kub | a \in \Phi, b \in [1, r-1]\}$ and note that $|\Delta| = kur$.

We now contract vertices in Δ. This is the same as in Definition 4.1.3: for $t \in [1, u]$, a_t is the set of u smallest vertices of Δ that are not in $a_1 \cup ... \cup a_{t-1}$. Because $d+1-v$ is a multiple of u, we have $a_{i+k} = a_i + (d+1)$. This implies that the contracted graph is $C_{n/u}^{(d+1)/u}$.

Similarly to the proof of case (i), we extend a cover for $C_{n/u}^{(d+1)/u}$ to cover every edge (i,j) for $i, j \in \Delta$. If we sum over all the possible ways to select subset Φ, we note that every edge $(i,j) \in C_{r(d+1)}^{d}$ is covered with probability at least $\left(\frac{d+1-v}{d+1}\right)^2$. □

4.1.4 Final step: Computer-aided proof of factor 2.79

We will now apply Proposition 4.1.2 with $p = 2(d+1)$ and $n_1 = 4(d+1)$. Let Ω_d be the set of $2(d+1)$-periodic permutations of $1, ..., 4(d+1)$. We can find covers for $C_{4(d+1)}^{d}$ using the following linear program:

$$\begin{aligned} \min \quad & \sum_{\sigma \in \Omega_d} \lambda_\sigma \\ \text{s.t.} \quad & \sum_{\sigma \in \Omega_d} \lambda_\sigma \mathbb{I}[b_i(\sigma, d) = b_j(\sigma, d)] \geq 1, \quad \forall (i,j) \in C_{4(d+1)}^{d} \\ & \lambda_\sigma \in \mathbb{R}^+, \qquad\qquad \sigma \in \Omega_d \end{aligned} \tag{LP$_d$}$$

Let α_d be the solution to LP$_d$. Let $\alpha = \sup_{d \geq 1} \alpha_d$. Batching is $(1/\alpha + O(1/n))$-competitive.

Proof. Follows from Propositions 4.1.1 and 4.1.2. □

The Linear program (LP$_d$) has $O(d!)$ variables, and solving it may not be computationally possible when d is large. Using Proposition 4.1.3, we now provide a way to find upper bounds on α_d by solving a different LP on a smaller graph. Recall that Ω_{k-1} is the set of $2k$-periodic permutations of $1, ..., 4k$. We define the problem of finding an $(\alpha, k-1)$-cover of the cycle C_{4k}^{k}.

$$\begin{aligned} \min \quad & \sum_{\sigma \in \Omega_{k-1}} \lambda_\sigma \\ \text{s.t.} \quad & \sum_{\sigma \in \Omega_{k-1}} \lambda_\sigma \mathbb{I}[b_i(\sigma, k-1) = b_j(\sigma, k-1)] \geq 1, \quad \forall (i,j) \in C_{4k}^{k} \\ & \lambda_\sigma \in \mathbb{R}^+, \qquad\qquad \sigma \in \Omega_{k-1} \end{aligned} \tag{LP'$_k$}$$

We denote by α'_k the solution to (LP'$_k$). Solving (LP'$_k$) numerically for $k = 4$ yields $\alpha'_4 \leq 3.17$ (See Table 1). For all $d \geq 52$ Proposition 4.1.3 therefore implies that, $\alpha_d \leq 3.17 * \left(\frac{51}{49}\right)^2 = 3.58$. [5] It remains to check that for all $d \leq 50$, $\alpha_d \leq 3.58$. We either solve (LP$_d$) directly (see left Table 1) or use Proposition 4.1.3 (see right Table 1) to conclude. Observing that $2.79 \leq \frac{1}{3.58}$, this allows us to conclude that Batching is 0.279-competitive, and concludes the proof for Theorem 4.1. □

[5] We note that our methodology can be extended to obtain a better factor. For instance, being able to solve (LP$_d$) for values higher than 50 would lead to a competitive ratio closer to $\frac{1}{3}$.

d	α_d	α'_d
1	2	
2	2.33	4
3	2.5	3.45
4	2.64	3.17
5	2.71	3.15
6	2.75	3.12
7	2.79	3.09
8	2.83	3.08
9	2.99*	3.07
10	3.2*	3.20*
11	3.11*	3.153*
12		3.264*
13	3.23*	3.318*

d	$\alpha_d \leq$	k used
17	3.58	4
19	3.48	6
23	3.44	11
29	3.31	7
31	3.36	5
37	3.30	6
41	3.31	5
43	3.24	7
47	3.35	9

Table 1: Left: Numerical values for α_d and α'_d for small values of d. Starred elements were solved approximately (are therefore upper bounds on the actual value). Right: Upper bounds for α_d for prime values of d; derived from Proposition 4.1.3 using the following formula: $\alpha_d \leq \alpha_k \left(\frac{d+1-v}{d+1}\right)^2$ for $k \leq d$ and $v = d \bmod k$.

4.2 Lower bound in random order.

No algorithm is more than $\frac{1}{2}$-competitive even under the random arrival order.

Proof. Consider a graph with three vertices $\{1, 2, 3\}$ and $d = 1$, i.e. vertices can only be matched to the ones arriving just before or after them. After the first two arrivals, the online algorithm $\mathcal{A}$ needs to decide whether to match them or let the first arrival leave. Furthermore, it has no information on how $v_{\sigma(1),\sigma(2)}$ compares to the other edge weights. Therefore the decision of whether to match has to be a coin toss. Regardless of whether the algorithm matches the first two or the last two arrivals, its expected reward is $\frac{v_{1,2}+v_{2,3}+v_{3,1}}{3}$. OFF however has an expected reward of $\frac{\max(v_{1,2},v_{2,3})+\max(v_{2,3},v_{3,1})+\max(v_{3,1},v_{1,2})}{3}$. Taking $v_{1,2} = v_{2,3} \to 0$, we get $\mathcal{A} = 1/3$ while OFF $= 2/3$ which concludes the proof. □

5 Extensions

5.1 Stochastic departures in the AO setting

We relax the assumption that all vertices depart after exactly d time steps.

We therefore focus on the stochastic case, in which the departure time d_i of vertex i is sampled independently from a distribution $\mathcal{D}$. We assume that the realizations d_i are only known at the time i becomes critical.

Suppose that there exists $\alpha \in (0, 1)$ such that $\mathcal{D}$ satisfies the property that for all $i < j$,

$$\mathbb{P}[i + d_i \leq j + d_j | i + d_i \geq j] \geq \alpha.$$

Then PG is $\alpha/4$-competitive.

Proof. When a vertex k becomes critical in the original graph, we match it if its status is determined to be *seller*. In that case, we need to ensure that its tentative match b_l is still present. With probability at least α, vertex l is still present, and we collect $p(s_k)$. The rest of the proof follows similarly to that of Theorem 3.1 □

For a memory-less process, conditional on i still being present when j arrives, the probability that i departs first is exactly $1/2$. Therefore the above condition is satisfied with $\alpha = 1/2$. PG is $1/8$-competitive when $\mathcal{D}$ is memory-less.

5.2 Look-ahead under RP setting

We assume now that the online algorithm knows vertices that will arrive in l time steps (and their adjacent edges). We can update the Batching Algorithm in the following way: every $d+l+1$ time steps, compute a maximum-weight matching on both the current vertices and the next l arrivals. Match vertices as they become critical according to the matching, and discard unmatched vertices. Note that this is the same as running Batching when the deadline is $d+l$.

There exists an $(\frac{d+l+1}{l+1}, d+l)$-cover of C_n^d.

Proof. For $k \in [0, d+l]$, let $\sigma_k(i) = i + k \mod n$. Let i, j be such that $|i-j| \leq d$, then $b_i(\sigma_k, d) = b_j(\sigma_k, d)$ for at least $l+1$ different values of k. We can conclude that $\sigma_0, ..., \sigma_{d+l}$ is a $(\frac{d+l+1}{l+1}, d+l)$-cover by taking $\lambda_0 = ... = \lambda_{d+l} = \frac{1}{l+1}$. □

Batching with l-lookahead is $\frac{l+1}{d+l+1}$-competitive when n is large.

5.3 AO setting: alternative to Greedy

Observe that the greedy algorithm discards a buyer that becomes unmatched and does not attempt to subsequently rematch it. In this section, we introduce the Dynamic Deferred Acceptance (DDA) algorithm, which takes as input a constrained bipartite graph and returns a matching (formally presented below). Although the DDA provides the same theoretical guarantees as greedy, we will see in Section 6 that it rationalizes *Re-Optimization* algorithms, which perform very well on practical instances of the carpooling problem.

The main idea is to maintain a tentative maximum-weight matching m at all times during the run of the algorithm. This tentative matching is updated according to an auction mechanism: every seller s is associated with a *price* p_s, which is initiated at zero upon arrival. Every buyer b that that already arrived and yet to become critical is associated with a *profit margin* q_b which corresponds to the value of matching to their most preferred seller minus the price associated with that seller. Every time a new buyer arrives, she bids on her most preferred seller at the current set of prices. This triggers a bidding process that terminates when no unmatched buyer can profitably bid on a seller.

When a seller becomes critical, she is irrevocably matched to her tentative match. A buyer is discarded only if she is unmatched and becomes critical.

At any point t throughout the algorithm, we maintain a set of sellers S_t, a set of buyers B_t, as well as a matching m, a price p_s for every seller $s \in S_t$, and a marginal profit q_b for every buyer $b \in B_t$.

Our algorithm bears similarities to the auction algorithm by Bertsekas [1988]. Prices in this auction increase by ϵ to ensure termination, and optimality is proven through ϵ-complementary

Algorithm 4 Dynamic Deferred Acceptance

- Input: an online graph with deadline d, $G_{d,\sigma}$.
- Process each event in the following way:
  1. *Arrival of a seller s:* Initialize $p_s \leftarrow 0$ and $m(s) \leftarrow \emptyset$.
  2. *Arrival of a buyer b:* Start the following *ascending auction.*
     Repeat
     (a) Let $q_b \leftarrow \max_{s' \in S_t} v_{s',b} - p_{s'}$ and $s \leftarrow \text{argmax}_{s' \in S_t} v_{s',b} - p_{s'}$.
     (b) If $q_b > 0$ then
        i. $p_s \leftarrow p_s + \epsilon$.
        ii. $m(s) \leftarrow b$ (tentatively match s to b)
        iii. Set b to $\emptyset$ if s was not matched before. Otherwise, let b be the previous match of s.
     Until $q_b \leq 0$ or $b = \emptyset$.
  3. *Departure of a seller s:* If seller s becomes critical and $m(s) \neq \emptyset$, finalize the matching of s and $m(s)$ and collect the reward of $v_{s,m(s)}$.

slackness conditions. For the analysis, we consider the limit $\epsilon \to 0$ and assume the auction phase terminates with the maximum weight matching. [6]

The auction phase is always initiated at the existing prices and profit margins. This, together with the fact that the graph is bipartite, ensures that prices never decrease and and marginal profits never increase throughout the algorithm. Furthermore, the prices and marginal profits of the sellers and buyers that are present in the "market" form an optimum dual for the matching linear program.

Consider the DDA algorithm on a constrained bipartite graph.

1. Sellers' prices never decrease, and buyers' profit margins never increase.
2. At the end of every ascending auction, prices of the sellers and the marginal profits of the buyers form an optimal solution to the dual of the matching linear program associated with buyers and sellers present at that particular time.

Maintaining a maximum-weight matching along with optimum dual variables does not guarantee an efficient matching for the whole graph. The dual values are not always feasible for the offline problem. Indeed, the profit margin of some buyer b may decrease after some seller departs the market. This is because b may face increasing competition from new buyers, while the bidding process excludes sellers that have already departed the market (whether matched or not).

DDA is $1/2$-competitive for constrained bipartite graphs.

Proof. The proof follows the primal-dual framework. First, observe that by complementary slackness, any seller s (buyer b) that departs unmatched has a final price $p_s^f = 0$ (final profit margin

[6] One way to formalize this argument is through the Hungarian algorithm Kuhn [1955], where prices are increased simultaneously along an alternating path that only uses edges for which the dual constraint is tight.

$q_b^f = 0$). When a seller s is critical and matches to b, we have $v_{s,b} = p_s^f + q_b^f$. Therefore, *DDA* collects a reward of $\mathcal{A} = \sum_{s \in S} p_s^f + \sum_{b \in B} q_b^f$.

Second, let us consider a buyer b and a seller $s \in [b-d, b)$ who has arrived before b but not more than d steps before. Because sellers do not finalize their matching before they are critical, we know that $s \in S_b$. An ascending auction may be triggered at the time of b's arrival, after which we have: $v_{s,b} \leq p_s(b) + q_b(b) \leq p_s^f + q_b^i$, where the second inequality follows from the definition that $q_b(b) = q_b^i$ and from the monotonicity of sellers' prices (Lemma 5.3). Thus, (p^f, q^i) is a feasible solution to the offline dual problem.

Finally, we observe that upon the arrival of a new buyer, the ascending auction does not change the sum of prices and margins for vertices who were already present:

Claim 5.1. *Consider an arriving buyer b, and let p, q (p', q') be the prices and margins before at the beginning (the end) the ascending auction phase (Step 2(a) in Algorithm 1). Then:*

$$\sum_{s \in S_t} p_s + \sum_{b \in B_t} q_b = \sum_{s \in S_t} p'_s + \sum_{b \in B_t} q'_b. \tag{2}$$

Where S_t and B_t are the set of sellers and buyers present before b arrived.

Proof of Claim 5.1. Because auctions are always started by *buyers*, no previously matched seller will become unmatched. Thus, except for the buyer which started the auction, every increase in a seller's price is exactly matched by a buyer's decrease in margin. □

By applying this equality iteratively after each arrival, we can relate the initial margins q^i to the final margins q^f and prices p^f:

Claim 5.2. $\sum_{s \in S} p_s^f + \sum_{b \in B} q_b^f = \sum_{b \in B} q_b^i$.

This completes the proof of Proposition 3.1 given that the offline algorithm achieves at most:

$$\mathcal{O} \leq \sum_{s \in S} p_s^f + \sum_{b \in B} q_b^i \leq 2\mathcal{A}.$$

□

Proof of Claim 5.2. We iteratively apply the result of Claim 5.1 after any new arrival. Let $\widetilde{S}_t$ (resp. $\widetilde{B}_t$) be the set of sellers (buyers) who have departed, or already been matched before time t. We show by induction over $t \leq T$ that:

$$\sum_{s \in \widetilde{S}_t} p_s^f + \sum_{b \in \widetilde{B}_t} q_b^f + \sum_{s \in S_t} p_s(t) + \sum_{b \in B_t} q_b(t) = \sum_{b \in \widetilde{B}_t} q_b^i + \sum_{b \in B_t} q_b^i. \tag{3}$$

This is obvious for $t = 1$. Suppose that it is true for $t \in [1, T-1]$. Note that departures do not affect (3). If the agent arrivint at $t+1$ is a seller, then for all other sellers s, $p_s(t+1) = p_s(t)$ and for all buyers b, $q_b(t+1) = q_b(t)$, thus (3), is clearly still satisfied. Suppose that vertex $t+1$ is a buyer. Using equation (2), we have:

$$\sum_{s\in S_{t+1}} p_s(t+1) + \sum_{b\in B_{t+1}} q_b(t+1) \tag{4}$$

$$= q_{t+1}(t+1) + \sum_{b\in B_t} q_b(t) + \sum_{s\in S_t} p_s(t) \tag{5}$$

$$= \sum_{b\in B_{t+1}} q_b^i. \tag{6}$$

Note that at time $T+d$, every vertex has departed. Thus, $\widetilde{S}_{T+d} = S$, $\widetilde{B}_{T+d} = B$ and $S_{T+d} = B_{T+d} = \emptyset$. This enables us to conclude our induction and the proof for (3).

□

6 Numerical results

The results from section 5.3 motivate a modified algorithm, termed *Re-Optimization*, in which vertices are no longer separated into buyers and sellers. At each time step, *Re-Opt* re-computes a tentative (non-bipartite) maximum weight matching over all the vertices that are present in the graph. Only vertices that are critical are matched to their tentative match at that time. This modified algorithm does not have theoretical guarantees, but we will show that it performs well on data-driven compatibility graphs.

We compare the *Re-Opt* against three benchmarks that have been previously considered, or that are commonly used in practice:

- The *Greedy* algorithm. The algorithm matches vertices as soon as possible to their available neighbor with the highest value (ties are broken in favor of the earliest arrival).

- The *Batching*(b) algorithm. The algorithm waits k times-steps and then finds a maximum-weight matching. Unmatched vertices are kept in the next batch.[7] We report the best simulation results across parameters a number of batch sizes.

- The Patient algorithm. This algorithm waits until waits until a vertex becomes critical, and matches it to the neighbor with the highest match value (ties are broken in favor of the earliest arrival). This allows to separate the value from knowing the time in which vertices become critical and the value of optimization.

Data

For this numerical experiment, we use a data set of all taxi rides in New York City over an hour time period [8]. For any pair k, l of trips, we can compute the Euclidean distance that would have been traveled had the two passengers taken the same taxi (with multiple stops). The value $v_{k,l}$ represents the "distance saved" by combining the trips.

This enables us to generate a dynamic graph in the following way. For $t \in [1, T]$:

1. Sample with replacement an arrival t from the data set.

[7] See Agatz et al. [2011], Ashlagi et al. [2013] in the case of ride-sharing and kidney exchange respectively.
[8] http://www.andresmh.com/nyctaxitrips/

2. For any vertex l that is present in the pool, compute the value $v_{t,l}$ of matching t to l.

3. Sample a departure time $d_t \sim \mathcal{D}$.

We consider three settings, termed *deterministic* and *exponential* and *uniform* respectively, in which $\mathcal{D}$ is either constant with value d, exponentially distributed with mean d, or uniform $[0, 2d]$. We will report simulation results for d between 50 and 150.

Results

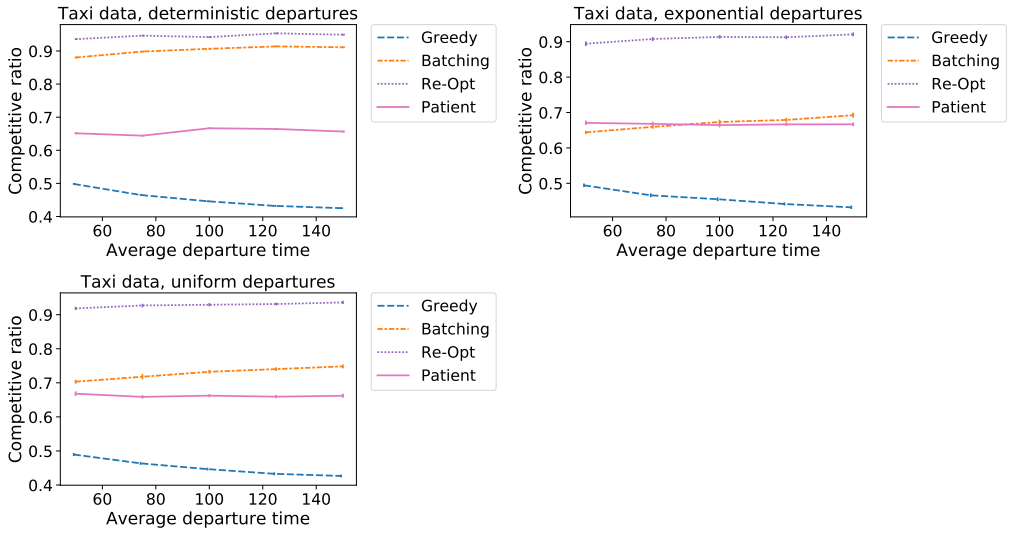

Figure 5: Performance of our 4 algorithms on taxi data (weighted compatibility graph).

In Figure 5, we observe that both the *Patient* and *Batching* algorithms outperform *Greedy* across all three departure settings. Intuitively, having vertices wait until they become critical helps to thicken the market and gives vertices higher valued match options. We notice that when departures are deterministic, *Batching* with the optimal batch size will be almost as efficient as *Re-Opt*. However when the departures are stochastic, there is value in matching vertices as they become critical.

We observe that *Re-Opt* outperforms all other algorithm, although in the cases where the departures are deterministic, *Batching* performs close to *Re-Opt* when the batch size k is carefully chosen. This shows the value of both having information on agents' departure times and also subsequent optimization.

It is important to note that the experiments we ran do not take into account the cost of waiting. We think that a richer model that accounts for this would be an interesting future direction. Two interesting areas for future work include the setting when the information about agent's departure times is uncertain, as well as models that are less restrictive than the adversarial setting (see, e.g., Ozkan and Ward [2016]).

7 Conclusion

This paper introduces a model for dynamic matching, in which all agents arrive and depart after some deadline. Match values are heterogeneous and the underlying graph is non-bipartite. We study online algorithms for two settings, where vertices arrive in an adversarial or random order.

In the adversarial arrival case, we introduce two new 1/4-competitive algorithms when departures are deterministic and known in advance. We also provide a 1/8-competitive algorithm when departures are stochastic, i.i.d, memoryless, and known at the time a vertex becomes critical. Finally we show that no online algorithm is more than 1/2-competitive.

In the random arrival case, we show that a batching algorithm is 0.279-competitive. We also show that with knowledge of future arrivals, its performance guarantee increases towards 1.

Importantly, our model imposes restrictions on the departure process and requires the algorithm to know when vertices become critical. Other than closing the gaps between the upper bound $1/2$ and the achievable competitive ratios, we point out a just a few interesting directions for future research. Our model imposes that matches retain the same value regardless of when they are conducted. An interesting direction is to account for agents' waiting times. A different interesting objective is to achieve both a high total value and a large fraction of matched agents. Finally, it is interesting to consider the stochastic setting with prior information over weights and future arrivals.

References

Niels AH Agatz, Alan L Erera, Martin WP Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro atlanta. *Transportation Research Part B: Methodological*, 45(9): 1450–1464, 2011.

Mohammad Akbarpour, Shengwu Li, and Shayan Oveis Gharan. Thickness and information in dynamic matching markets. 2017.

Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proc Natl Acad Sci USA*, 2017.

Ross Anderson, Itai Ashlagi, David Gamarnik, and Yash Kanoria. A dynamic model of barter exchange. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1925–1933. Society for Industrial and Applied Mathematics, 2015.

I. Ashlagi, P. Jaillet, and V.H. Manshadi. Kidney exchange in dynamic sparse heterogenous pools. *arXiv preprint arXiv:1301.3509*, 2013.

Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 81. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017a.

Itai Ashlagi, Maximilien Burq, Patrick Jaillet, and Vahideh Manshadi. On matching and thickness in heterogeneous dynamic markets. 2017b.

M. Baccara, S. Lee, and L. Yariv. Optimal dynamic matching. Working paper, 2015.

Siddhartha Banerjee, Yash Kanoria, and Pengyu Qian. State dependent control of closed queueing networks. In *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*, pages 2–4. ACM, 2018.

Dimitri P Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of operations research*, 14(1):105–123, 1988.

Francis YL Chin, Marek Chrobak, Stanley PY Fung, Wojciech Jawor, Jiří Sgall, and Tomáš Tichỳ. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4(2):255–276, 2006.

John P Dickerson, Ariel D Procaccia, and Tuomas Sandholm. Failure-aware kidney exchange. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 323–340. ACM, 2013.

Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 333–344. ACM, 2016.

J. Feldman, A. Mehta, V. S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 117–126, 2009a.

Jon Feldman, Nitish Korula, Vahab Mirrokni, S Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *International Workshop on Internet and Network Economics*, pages 374–385. Springer, 2009b.

G. Goel and A. Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 982–991, 2008.

Ming Hu and Yun Zhou. Dynamic type matching. 2016.

Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. How to match when all vertices arrive online. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 17–29. ACM, 2018.

Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2875–2886, 2019.

P. Jaillet and X. Lu. Online stochastic matching: New algorithms with better bounds. *Mathematics of Operations Research*, 39(3):624–646, 2013.

R. M Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing (STOC)*, pages 352–358, 1990.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.

Fei Li, Jay Sethuraman, and Clifford Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 801–802. Society for Industrial and Applied Mathematics, 2005.

V. H. Manshadi, S. Oveis-Gharan, and A. Saberi. Online stochastic matching: online actions based on offline statistics. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1285–1294, 2011.

Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science*, 8(4):265–368, 2013.

Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *Journal of the ACM (JACM)*, 54(5):22, 2007.

Michael Ostrovsky and Michael Schwarz. Carpooling and the economics of self-driving cars. Technical report, National Bureau of Economic Research, 2018.

Erhun Ozkan and Amy R Ward. Dynamic matching for real-time ridesharing. 2016.

Marco Pavone, Stephen L Smith, Emilio Frazzoli, and Daniela Rus. Robotic load balancing for mobility-ondemand systems. *Int J Rob Res*, 2012.

Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H. Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. In *Proc Natl Acad Sci USA*, 2014.

Kevin Spieser, Samitha Samaranayake, Wolfgang Gruel, and Emilio Frazzoli. Shared-vehicle mobility-ondemand systems: A fleet operators guide to rebalancing empty vehicles. In *Transportation Research Board 95th Annual Meeting*, 2016.

M. U. Ünver. Dynamic Kidney Exchange. *Review of Economic Studies*, 77(1):372–414, 2010.

William Vickrey. Pricing as a tool in coordination of local transportation. In *Transportation economics*, pages 275–296. NBER, 1965.

Rick Zhang and Marco Pavone. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. In *Proceedings of Robotics: Science and Systems Conference*, 2014.