View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Stable Matching with Evolving Preferences*

Varun Kanade[1], Nikos Leonardos[2], and Frédéric Magniez[3]

1 **University of Oxford, U.K.**
varunk@cs.ox.ac.uk
2 **University of Athens, Greece**
nikos.leonardos@gmail.com
3 **CNRS and IRIF, Univ Paris Diderot, Sorbonne Paris-Cité, France**
frederic.magniez@cnrs.fr

Abstract

We consider the problem of stable matching with dynamic preference lists. At each time-step, the preference list of some player may change by swapping random adjacent members. The goal of a central agency (algorithm) is to maintain an approximately stable matching, in terms of number of blocking pairs, at all time-steps. The changes in the preference lists are not reported to the algorithm, but must instead be probed explicitly. We design an algorithm that in expectation and with high probability maintains a matching that has at most $O((\log n)^2)$ blocking pairs.

1998 ACM Subject Classification F.1.2 Models of Computation

Keywords and phrases Stable Matching, Dynamic Data

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2016.36

1 Introduction

In the world of massive and distributed data, it is hardly reasonable to assume that data are static. Yet, one must design algorithms that maintain a solution for a given problem that is (approximately) consistent with the requirements, *e.g.,* a permutation that is almost sorted. Thus, it is important to design algorithms and data structures that are robust to changes in their input, *i.e.,* they produce an output with some performance guarantee (quickly).

There are a few different dynamic data models that have been considered. The area of dynamic graph algorithms consists of maintaining some property or structure, such as connectivity, matchings, or spanning trees, even when the underlying graphs are changing [3, 11, 10, 5]. Here, it is assumed that the changes to the graph may be *arbitrary*, but are reported to the algorithm; and the focus is on designing data structures and algorithms that adapt efficiently (typically in terms of computational time) to changes in the input. The area of streaming algorithms studies the setting where the data can only be accessed as a *stream* and the focus is on producing the desired output with highly space-efficient procedures (typically poly-logarithmic in the size of the input). In the area of online algorithms, one must design procedures that, even when data is revealed bit by bit, produce an output that is *competitive* with algorithms that see the entire input at once.

Recently, Anagnostopoulos *et al.* [1] proposed the *evolving data model* to take into account the dynamic aspects of massive data. In this model, the changes to the data are not

* Partially supported by the French ANR Blanc project ANR-12-BS02-005 (RDAM) and the European Commission IST STREP project Quantum Algorithms (QALGO) 600700. Most of the work was carried out when V. K and N. L. were at LIAFA (now IRIF). During this time V.K. was supported by the Fondation Sciences Mathématiques de Paris (FSMP).


© Varun Kanade, Nikos Leonardos, and Frédéric Magniez;
licensed under Creative Commons License CC-BY

revealed to the algorithm; instead, an algorithm has query access to the data. However, it is assumed that the changes to the data are *stochastic*, not adversarial. In this setting, the focus is not on computational complexity (which is allowed to be polynomial at each time-step), but query complexity, the number of probes made by the algorithm. Anagnostopoulos *et al.* [1] studied the problem of maintaining the *minimum* element of a permutation and an approximately sorted permutation, motivated by questions such as maintaining high (page)-ranked pages. In their setup, a permutation evolves by choosing a random element and swapping it with an adjacent element. In later work, Anagnostopoulos *et al.* [2] studied evolving graph models and problems such as *s-t* connectivity and minimum spanning tree.

In this work, our aim is to bring this notion to game theory starting from the basic problem of computing a stable matching. In other words, we introduce the notion of evolving agents, who may not report any updates to their strategy (or preferences) without an explicit request. In the stable matching problem, the input consists of two sets A, B of equal size, and for each member a total order (preference) over members in the other set. Given a matching between A and B, a pair (x, y) with $x \in A$ and $y \in B$ is blocking if they prefer each other to their matches. A matching is stable if there are no blocking pairs. Gale and Shapley showed that a stable matching always exists and can be found by an efficient algorithm [4]. We consider the setting where the preference lists *evolve* over time. The preference lists can evolve over time, by swapping adjacent elements. More precisely, while the algorithm can perform one query per time-step, we assume that a total number of α swaps events also occur, where $\alpha = \Theta(1)$ is some fixed parameter, called the *evolution rate*. This assumption is similar to previous works and models the critical regime: with less evolution events the input is basically static, and with more the input evolves too fast. The goal is to maintain a matching that has *few* blocking pairs.

We summarize our results as follows. All three statements hold in expectation and with high probability.

1. Using the results of Anagnostopoulos *et al.* [1] for sorting permutations, we design an algorithm that maintains a matching with at most $O(n \log n)$ blocking pairs, at all time-steps after roughly the first $n^2 \log n$ steps (Theorem 6). Also, we observe that any analysis that uses their method as a black-box, cannot improve on this bound (Remark 7).[1]
2. In a restricted setting, where only one side, say the B side, has evolving preference lists, and if the A side has uniform random permutations as preference lists (known to the central agency), we design an algorithm that maintains a matching with $O(\log n)$ blocking pairs at all time-steps after roughly the first $n \log n$ steps (Theorem 8).
3. Finally, we design an algorithm in the general setting, that maintains a matching with at most $O((\log n)^2)$ blocking pairs at all time-steps after roughly the first $n^2 \log n$ steps (Theorem 14).

2 Preliminaries

In the rest of the paper, $n \geq 1$ denotes an integer parameter and $[n]$ the set of integers $\{1, 2, \ldots, n\}$. For a non-negative random variable X, parametrized by some integer n, we write "$X = O(f(n))$ *in expectation and with high probability*" when for any constant c there exist constants $n_0, c', c'' > 0$ such that $\mathbb{E}[X] \leq c' f(n)$ and $\Pr(X > c'' f(n)) \leq n^{-c}$, for every integer $n \geq n_0$.

[1] We don't rule out the possibility that a more fine-grained analysis of the algorithm may give better bounds; instead we design new algorithms.

2.1 Stable Matching

We only consider the bipartite stable matching problem, also known as stable marriage. There are two sets of players A and B, with $|A| = |B| = n$. Each player $x \in A$ ($y \in B$) holds a permutation of B (A), denoted $\pi_x : B \to [n]$ ($\pi_y : A \to [n]$) indicating their preferences over players in the set B (A). Thus, for $y \in B$, $\pi_x(y)$ denotes the rank of y in x's preference list (where 1 is the highest rank).

Let $M : A \to B$ be a matching (a bijection). A pair (x, y) is said to be *blocking* if $y \prec_{\pi_x} M(x)$ and $x \prec_{\pi_y} M^{-1}(y)$, where $z \prec_\pi z'$ indicates that z is ranked higher than z' according to permutation π (*i.e.,* $\pi(z) < \pi(z')$). In words, x prefers y to $M(x)$ and y prefers x to $M^{-1}(y)$.

A matching M is *stable* if there are no blocking pairs. Then the *stable matching problem* is to find a stable matching given preference lists $\{\pi_z : z \in A \cup B)\}$. Gale and Shapley [4] proved that a stable matching always exists, and gave an algorithm that given the preferences lists as input finds a stable matching in $O(n^2)$ time.

The Gale-Shapley algorithm is simple to describe. Only players in the set A make proposals. Initially all players are *unmatched*. Let M denote a partial matching at some point. If there is an unmatched player $x \in A$, x makes a proposal to $y \in B$, where y is the highest-ranked player in π_x to whom x has not yet proposed. If y is unmatched, or prefers x to $M^{-1}(y)$, then y accepts the proposal and we set $M(x) = y$. In the latter case, the agent previously matched to y, *i.e.,* $M^{-1}(y)$ before M was updated, becomes unmatched once more. Gale and Shapley showed that this algorithm always results in a stable matching.

Wilson [12] studied the problem where all the preference lists are independent and uniformly random permutations; in this case, he showed that the number of proposals made by the Gale-Shapley algorithm is $O(n \log n)$ in expectation and with high probability (see also [9]). In fact, only the proposing side needs to be random in their statement. We provide a proof sketch for completeness.

▶ **Theorem 1** ([12]). *If the permutations $\{\pi_x : x \in A\}$ are chosen randomly, the number of proposals made in the Gale-Shapley algorithm (where only A makes proposals) is $O(n \log n)$ in expectation and with high probability.*

Proof Sketch. Following the proof in [9] (see also [6]), analyze an alternative procedure where every proposal is uniform over the whole of B. If it happens that $x \in A$ proposes to a $y \in B$ that has already rejected x, then a rejection is guaranteed. It is not hard to show that the number of proposals such an algorithm makes stochastically dominates the number of proposals of the classical algorithm. Next, by the method of deferred decisions, fix the randomness in the algorithm only when needed. Then observe that the number of proposals is equal to the number of coupons chosen in the coupon-collector's problem. ◀

2.2 Model for evolving input

A general framework for studying dynamic data was introduced in [1]. Here we are only concerned with evolving preference lists (or permutations). In our model, we consider discrete *time-steps*. In each time-step, the algorithm is competing against *nature* as follows: the algorithm can *query* the input locally, *nature* lets the input evolve according to one or more *evolution events*.

A query to the stable matching input is a triplet $(z, u, v) \in (A \times B^2) \cup (B \times A^2)$ and the answer is whether $\pi_z(u) < \pi_z(v)$. One evolution event consists of the following: pick $z \in A \cup B$ and $i \in [n-1]$ uniformly at random and swap $u = \pi_z^{-1}(i)$ and $v = \pi_z^{-1}(i+1)$ (*i.e.,* set $\pi_z(u) = i + 1$ and $\pi_z(v) = i$).

While the algorithm can perform one query per time-step, α evolution events also occur, where $\alpha \geq 1$ is some integer called the *evolution rate*. We further assume that $\alpha = \Theta(1)$, meaning that evolution events occur basically as often than the algorithm probes. We emphasize that the rate-limiting factor is the queries made by the algorithm. In particular, the algorithm may perform arbitrary (polynomial-time) computations in between time-steps. We are now ready to define our problem:

Evolving Stable Matching (ESM): Given query access to an instance of the stable matching problem with evolution rate $\alpha = \Theta(1)$, maintain a matching that minimizes the number of blocking pairs.

2.3 Sorting evolving permutations

The problem of sorting a single evolving permutation has been already addressed in [1]. In this context, the evolution rate is still constant, but denotes the evolution rate of this single permutation. We will use the algorithm QUICKSORT of [1]. It is simply the randomized version of quicksort which is shown to be robust with respect to an evolving input. The first lemma shows that the running time of quicksort is not affected by evolution events.[2]

▶ **Lemma 2** (Proposition 3 in [1]). *The running time of* QUICKSORT *is* $O(n \log n)$ *in expectation and with high probability, for any rate of evolution when the pairs to be swapped are chosen randomly.*

Second, Lemma 6 in [1] states that QUICKSORT when run on an evolving permutation π, computes a permutation $\tilde{\pi}$ in which every element is approximately sorted. At time-step t, let π^t the denote the current permutation, and $\tilde{\pi}^t$ its approximation computed by the algorithm.

▶ **Lemma 3** (Lemma 6 in [1]). *Let* t *be the time-step of completion of* QUICKSORT*, then given an element* u*, the number of pairs* (u, v) *that the permutations* π^t *and* $\tilde{\pi}^t$ *rank differently is* $O(\log n)$ *in expectation and with high probability.*

In our setting there are $2n$ evolving permutations over some set of n elements. Algorithm 1 simply sorts m (out of $2n$) permutations, denoted by $\pi_1, \ldots, \pi_m$ using QUICKSORT one after another. (We always invoke Algorithm 1 with either n or $2n$ permutations.)

Algorithm 1 : Sequential sorting

```
1: procedure SequentialSort({π_j : j = 1, ..., m}) ▷ Only have query access to input
2:     for j = 1 to m do
3:         π̃_j ← Quicksort(π_j)
4:     return {π̃_j : j = 1, ..., m}
```

Using Lemma 3 (Lemma 6 of [1]) we can argue that Algorithm 1 maintains all permutations approximately sorted. While the evolving rate is still $\alpha = \Theta(1)$, there are now $2n$ evolving permutations, and the total number of evolution events is α per time-step.

[2] We remark that Anagnostopoulos *et al.* [1] use 'whp' to denote events that hold with probability $1 - o(1)$, rather than the stronger notion we use in this paper. However, their proofs for the results used in our paper actually prove the stronger bounds. They have other results that do not satisfy the stronger notion and these are not used in our work.

▶ **Lemma 4.** *Let t be the time-step when Algorithm 1 terminates. Then, for $m \leq 2n$, given any element u and $j \in [m]$, the number of pairs (u, v) that the permutations π_j^t and $\tilde{\pi}_j^t$ rank differently is $O(\log n)$ in expectation and with high probability.*

Proof. Fix some $j \in [n]$. Suppose that $\tilde{\pi}_j^t$ was computed at time-step $t' \leq t$ (the time-step when QUICKSORT for this particular list terminates). By Lemma 3 the statement holds for u at time-step t'. Due to Lemma 2 we have $t - t' = O(n^2 \log n)$ with high probability. During this time, the number of evolution steps that have swapped u with an adjacent element is $O(\log n)$ with high probability. This follows from a balls-and-bins experiment where we throw $O(n^2 \log n)$ balls (corresponding to the evolution steps) into $m(n-1)$ bins (corresponding to the adjacent pairs). It is known (see Exercise 3.1 in [9]) that in this particular case the number of balls in every bin is of the order of its mean with high probability. Therefore, during this time, at most $O(\log n)$ more elements may be swapped with u. ◀

2.4 Chernoff Bound with dependent variables

We will require the following extension of the Chernoff bound. It follows from (the more general) Theorem 3.8 in [8].

▶ **Theorem 5.** *For $i \in [n]$, let Y_i be a random variable over some set $\mathcal{Y}_i$ and X_i be a Boolean random variable. For any $y \in \prod_{i=1}^{n} \mathcal{Y}_i$ and $k \in [n]$, let $E_k(y)$ denote the event $Y_1 = y_1, \ldots, Y_k = y_k$. Suppose $\mathbb{P}[X_k = 1 | E_{k-1}(y)] \leq p$, for all y and k as above. Then, for any $t \geq 0$,*

$$\mathbb{P}\Big[\sum X_k \geq pn + t\Big] \leq \exp\Big(-\frac{3t^2}{6pn + 2t}\Big).$$

3 Two simple cases

In this section we present two simple arguments in two different settings. First, we consider how the original Gale-Shapley algorithm performs when run, without any modification, on lists produced by running quicksort on the evolving input. We present a simple analysis showing a bound of $O(n \log n)$ on the number of blocking pairs, which is better than the trivial bound of order n^2. Next, we analyze the Gale-Shapley algorithm when evolution events only occur on one side and the preference lists are uniformly random permutations; in particular the preference lists on the A side are chosen uniformly at random, and the preference lists on the B-side are subject to evolution events.[3] We present a simple analysis showing an $O(\log n)$ bound on the number of blocking pairs for this special case.

3.1 A simple algorithm

Our first algorithm ignores evolution of preference lists and runs the standard Gale-Shapley algorithm to produce a matching. More specifically, it first obtains the preferences lists for all $2n$ agents using the QUICKSORT algorithm of [1] (*i.e.,* using Algorithm 1) and then produces a matching by running the Gale-Shapley algorithm on these lists (ignoring the fact that these lists are only *approximately* correct).

[3] Note that after sufficiently many time-steps (though still polynomial) the evolution events ensure that all permutations are uniformly random. This follows from analyzing the mixing time of the corresponding Markov chain over permutations. See for example the book [7].

We show that this simple algorithm maintains a matching with at most $O(n \log n)$ blocking pairs. Note that the number of blocking pairs is trivially at most n^2. We further argue that improving the bound would require new ideas that either go around Lemma 4 (Lemma 6 of [1]) or improve the analysis in a substantial way.

Algorithm 2 runs in perpetuity. The matching M is maintained as the output until the new matching based on the newly sorted preference lists can be computed.

Algorithm 2 : Simple dynamic stable matching

1: **while** TRUE **do**
2: $\{\tilde{\pi}_z \ : \ z \in A \cup B\} \leftarrow$ SEQUENTIALSORT$(\{\pi_z \ : \ z \in A \cup B\})$ ▷ Calling Algorithm 1
3: **return** Gale-Shapley matching M on the (approximately) sorted lists $\{\tilde{\pi}_z : z \in A \cup B\}$

▶ **Theorem 6.** *For a sufficiently large constant c_0 and any time-step $t \geq c_0 n^2 \log n$, Algorithm 2 maintains a matching with $O(n \log n)$ blocking pairs in expectation and with high probability.*

Proof. We consider the number of blocking pairs at time-step $T \leq t$ when the current matching M was computed. In the following discussion, we use the following notation: for $x \in A, y \in B$, if $M(x) = y$, then $M(y) = x$ (rather than M^{-1}). At time-step T, for each $z \in A \cup B$, define the indicator function $I_z(w)$ to be 1 when $M(z) \prec_{\tilde{\pi}_z} w$ and $w \prec_{\pi_z} M(z)$ and 0 otherwise. (We don't explicitly use superscripts on the preference lists π as time-step T is fixed until specified otherwise.) By Lemma 4, in expectation and with high probability,

$$\sum_w I_z(w) = \left|\{w : w \prec_{\pi_z} M(z) \text{ and } M(z) \prec_{\tilde{\pi}_z} w\}\right| = O(\log n), \tag{1}$$

If a pair (x, y) is blocking at time-step T, then $x \prec_{\pi_y} M(y)$ and $y \prec_{\pi_x} M(x)$. Assume $x \prec_{\tilde{\pi}_y} M(y)$ and $y \prec_{\tilde{\pi}_x} M(x)$. Since $y \prec_{\tilde{\pi}_x} M(x)$, x must have proposed to y at some point during the execution of the Gale-Shapley algorithm. By the properties of the Gale-Shapley algorithm, y should have been matched to an element of A with rank according to $\tilde{\pi}_y$ at least as high as the rank of x in $\tilde{\pi}_y$. This contradicts $x \prec_{\tilde{\pi}_y} M(y)$. It follows that either $M(y) \prec_{\tilde{\pi}_y} x$ or $M(x) \prec_{\tilde{\pi}_x} y$. Define $U(x, y)$ to be 1 when (x, y) is blocking and 0 otherwise. We have argued that

$$U(x, y) \leq I_x(y) + I_y(x).$$

By the union bound, Equation 1 holds for every $z \in A \cup B$ with high probability. Summing over all pairs (x, y) and applying the union bound again

$$\sum_{x,y} U(x, y) \leq \sum_{x,y} I_x(y) + I_y(x) = \sum_x \Big(\sum_y I_x(y)\Big) + \sum_y \Big(\sum_x I_y(x)\Big) = O(n \log n),$$

in expectation and with high probability.

Next, let $t' = t - T$. We need to account for blocking pairs that may have arisen during these t' time-steps. First, we observe that by Lemma 2 and union bound, with high probability $t' = O(n^2 \log n)$ (as otherwise another matching more up-to-date than the one at time-step T would be available). During the t' time-steps from T to t, evolution may create a blocking pair only if the swap decreases the rank of $M(z)$ in π_z, for some $z \in A \cup B$. Therefore, each step of the evolution introduces—independently—a new blocking pair with

A-side lists π

1	3	2	1	4	5	6	7
2	4	3	2	5	6	7	1
3	5	4	3	6	7	1	2
4	6	5	4	7	1	2	3
5	7	6	5	1	2	3	4
6	1	7	6	2	3	4	5
7	2	1	7	3	4	5	6

B-side lists π

1	6	7	1	5	4	3	2
2	7	1	2	6	5	4	3
3	1	2	3	7	6	5	4
4	2	3	4	1	7	6	5
5	3	4	5	2	1	7	6
6	4	5	6	3	2	1	7
7	5	6	7	4	3	2	1

A-side lists $\tilde{\pi}$

1	1	2	3	4	5	6	7
2	2	3	4	5	6	7	1
3	3	4	5	6	7	1	2
4	4	5	6	7	1	2	3
5	5	6	7	1	2	3	4
6	6	7	1	2	3	4	5
7	7	1	2	3	4	5	6

B-side lists $\tilde{\pi}$

1	1	7	6	5	4	3	2
2	2	1	7	6	5	4	3
3	3	2	1	7	6	5	4
4	4	3	2	1	7	6	5
5	5	4	3	2	1	7	6
6	6	5	4	3	2	1	7
7	7	6	5	4	3	2	1

Figure 1 Instances π and $\tilde{\pi}$ demonstrating tightness of Algorithm 2.

probability at most $\alpha/(n-1)$. The expected number of blocking pairs introduced is therefore at most $O(n \log n)$ for $\alpha = \Theta(1)$ (and assuming $t' = O(n^2 \log n)$). The result follows by a simple application of the Chernoff and union bounds. ◀

▶ Remark 7. We present a pair of instances to the stable matching problem with preference lists π_z and $\tilde{\pi}_z$ for $z \in A \cup B$, for which the conclusion of Lemma 4 is satisfied, *i.e.,* for any z the number of pairs (i, j) that are ordered differently in π_z and $\tilde{\pi}_z$ are $O(\log n)$. We then show that a matching that is stable with respect to $\tilde{\pi}$ as preference lists has $\Omega(n \log n)$ blocking pairs with respect to π. Thus, it follows that using the QUICKSORT algorithm of Anagnostopoulos *et al.* [1] and its analysis as a blackbox will not result in a stronger result than the one provided in Theorem 6.

First, we define the preference lists $\tilde{\pi}_z$ for $z \in A \cup B$. Let $A = B = [n]$. Then for $x \in A$, the preference list (ranking) $\tilde{\pi}_x$ is defined as $x, x+1, \ldots, n, 1, 2, \ldots, x-1$. On the other hand for $y \in B$, the preference list (ranking) $\tilde{\pi}_y$ is defined as $y, y-1, \ldots, 1, n, n-1, \ldots, y+1$. The rankings π_z, $z \in A \cup B$, are now defined as follows: let k be some parameter, π_z simply as the elements at rank 1 and k swapped. Figure 1 shows an example with $n = 7$ and $k = 3$. Clearly, when $k = \Theta(\log n)$, π_z and $\tilde{\pi}_z$ satisfy the conclusion of Lemma 4. Yet, it is easy to see that $M(i) = i$ is a stable matching for the preference lists $\tilde{\pi}_z$, $z \in A \cup B$, and for this matching with respect to the preference lists π_z, $z \in A \cup B$, every pair (i, j) with $0 < j - i < k$ is a blocking pair.

3.2 One-sided evolution

In this section we analyze how the Gale-Shapley algorithm performs when the initial preference lists are random, but there is no evolution on the lists of the elements in A. Furthermore, we are going to assume that the algorithm knows each permutation in $\{\pi_x : x \in A\}$. We call this setting *one-sided evolution.*

In this setting, the standard Gale-Shapley algorithm is implemented (basic pseudocode is shown in Algorithm 3). Note that the only time the preference lists on the B-side are used is in Line 11. Thus, it is only for these steps that we need to query the input (since the preference lists on the A-side are known to the algorithm). Thus, the number of queries made by the algorithm is bounded by the number of proposals. It was already observed that the number of proposals made in a random instance of the stable matching problem is $O(n \log n)$. The actual algorithm keeps implementing the Gale-Shapley algorithm from scratch after completion. The matching from the previous completed run is used as the current matching. We prove the following result for the one-sided evolution setting.

Algorithm 3 Gale-Shapley Algorithm

```
M ← φ                                                    ▷ Initialize empty matching
for x ∈ A do
    new_match ← False
    p ← x
    while new_match = False do
        y ← first as yet unproposed as per π_p
        if M(y) not yet set then
            M(p) ← y
            M(y) ← p
            new_match = True
        else if p ≺_{π_y} M(y) then                      ▷ y prefers p to M(y)
            p′ ← M(y)
            M(y) ← p
            M(p) ← y
            p ← p′
return M
```

▶ **Theorem 8.** *For a sufficiently large constant c_0 and any time-step $t \geq c_0 n \log n$, the Gale-Shapley algorithm (repeatedly run and using the matching of the last completed run as the output) under one-sided evolution maintains a matching with at most $O(\log n)$ blocking pairs in expectation and with high probability.*

Proof. To prove the bound on the number of blocking pairs, call an evolution event on y's list *critical* if it involves the *then* match of y. Suppose that after the algorithm terminates, $y \in B$ is involved in k blocking pairs $(x_1, y), \ldots, (x_k, y)$. We observe that each one of the $x_1, \ldots, x_k$ was involved in at least one critical evolution step. To see this note that if (x, y) is blocking, then x proposed to y during the execution of the algorithm and got rejected subsequently (because π_x didn't change and $y \prec_{\pi_x} M(x)$). But since at the end of the execution it forms a blocking pair, it must ranked higher than $M(y)$. This is only possible if x was swapped with the *then* match of y in some evolution event during the execution of the algorithm.

Given this observation, we estimate the number of blocking pairs by estimating the number of critical evolution steps. Note that an evolution step is critical with probability at most $2\alpha/(n-1)$ (at most $2n$ out of $n(n-1)$ pairs involve the matching). Let T be a random variable equal to the number of proposals before the algorithm outputs a matching and label the corresponding time-steps as $1, 2, \ldots, T$. For each step k, let X_k be a Boolean random variable that is equal to 1 if at the time-step labeled k some evolution event was critical.

First note that from a coupon-collecting argument as in Theorem 1 it follows that $T = O(n \log n)$ in expectation and with high probability. This is because for that argument the distribution of $\{\pi_y : y \in B\}$ is irrelevant and $\{\pi_x : x \in A\}$ being random permutations suffices. Therefore, we may fix an appropriately large constant C so that $T \leq Cn \log n$ with high probability and let $m = Cn \log n$. As noted above, at any given step and given any information from the previous steps, an evolution step is critical with probability at most $2\alpha/(n-1)$; thus, by Theorem 5,

$$\mathbb{P}\Big[\sum_{k=1}^{m} X_k > 2C \log n\Big] = O(n^{-C}),$$

for sufficiently large C. We have

$$\mathbb{P}\Big[\sum_{k=1}^{T} X_k > 2C\log n\Big] \leq \mathbb{P}\big[T > Cn\log n\big] + \mathbb{P}\Big[\Big(\sum_{k=1}^{T} X_k > 2C\log n\Big)\wedge\big(T \leq Cn\log n\big)\Big]$$
$$\leq \mathbb{P}\big[T > Cn\log n\big] + \mathbb{P}\Big[\sum_{k=1}^{m} X_k > 2C\log n\Big].$$

By the observation at the beginning of this paragraph the claimed bound holds with high probability. The bound on the expectation follows by noting that there can be at most n^2 blocking pairs.

Finally, note that there will be at most $O(n\log n)$ time-steps before a new matching is computed. As in the final paragraph in the proof of Theorem 6, one can show that evolution cannot produce more than $O(\log n)$ blocking pairs in these many steps. ◀

4 General Case: Improved algorithm

We now consider the general setting where the preference lists on both sides may be evolving. We present a modified version of the Gale-Shapley algorithm that takes advantage of Lemma 4 (Lemma 6 of [1]) and maintains a stable matching with at most $O((\log n)^2)$ blocking pairs. The algorithm consists of two separate processes that run in an interleaved fashion: the *sorting process* on even time-steps and the *matching process* on odd ones. The sorting process is basically a call to SequentialSort($\{\pi_x \mid x \in A\}$) that produces *approximately sorted* preference lists on the A side, $\{\tilde{\pi}_x \mid x \in A\}$. The algorithm runs in perpetuity, in the sense that as soon as it terminates it restarts, though the copies $\{\tilde{\pi}_x \mid x \in A\}$ from the previous execution are retained to be used by the stable matching process. Initially, the $\tilde{\pi}_x$ are set to be random permutations, thus, for the first $O(n^2\log n)$ steps, until one run of the sorting process is complete, the matching output by the algorithm will be garbage.

Algorithm 4 : Interleaving Sorting and Matching

```
for t = 1, 2, ... do
    if t is Even then
        Perform query for Algorithm 1
    else if t is Odd then
        Perform query for Algorithm 5
```

We note that what is counted here is only time-steps; making queries is the bottleneck. Additional computations required by the algorithm can be performed in between time-steps at no additional cost.

The sorting process performs queries only during even steps and its purpose it to keep the preference list of each $x \in A$ approximately sorted, where by approximately sorted we meant that the conclusion of Lemma 4 holds.

The matching process performs queries during odd steps. Our stable matching algorithm, which is a variant of the Gale-Shapley algorithm, is presented in Algorithm 5. Note that the $\{\tilde{\pi}_x \mid x \in A\}$ used in Algorithm 5 are *static* and are the output of the latest completed run of Algorithm 1. However, the comparisons made are all on dynamic data. The difference from the standard Gale-Shapley algorithm is that whenever some $x \in A$ is about to make a proposal, first the *best* $y \in B$ among the $O(\log n)$ highest ranked as per the ranking $\tilde{\pi}_x$ that have not yet rejected x is found. Note however, that the best is with respect to the

dynamic (current) preference list π_x (otherwise, it would be trivial since $\tilde{\pi}_x$ is static). This operation is basically the algorithm to find the minimum element, which can be implemented in $O(\log n)$ time using only comparison queries (see Section 3 in [1]). We don't need to use any particular result regarding finding the minimum element; instead, we incorporate the errors that may have occurred while finding the minimum due to the dynamic nature of the data, into our stable matching analysis directly.

Algorithm 5 : Modified Gale-Shapley Algorithm

1: $M \leftarrow \emptyset$
2: **for** $x \in A$ **do**
3: `new_match` $\leftarrow$ FALSE
4: $p \leftarrow x$
5: **while** `new_match` $=$ FALSE **do**
6: $S \leftarrow C \log n$ `highest-ranked, not-yet-proposed elements in` B `per` $\tilde{\pi}_x$
7: $y \leftarrow$ `best`(S) ▷ Best with respect to dynamic π_x
8: **if** $M(y)$ `not yet set` **then**
9: $M(p) \leftarrow y$
10: $M(y) \leftarrow p$
11: `new_match` $\leftarrow$ TRUE
12: **else if** $p \prec_{\pi_y} M(y)$ **then**
13: $p' \leftarrow M(y)$
14: $M(y) \leftarrow p$
15: $M(p) \leftarrow y$
16: $p \leftarrow p'$
17: **return** M

We first describe the high-level idea of the proof. The sorting process needs $O(n^2 \log n)$ comparisons with high probability. The approximations $\{\tilde{\pi}_x : x \in A\}$ of $\{\pi_x : x \in A\}$ that are being computed are used by the modified Gale-Shapley algorithm. By Lemma 4 we are able to claim that for any element u in the preference list of any $x \in A$, the number of pairs (u, v) that are ordered differently in $\tilde{\pi}_x$ and π_x are $O(\log n)$. Therefore, when x is about to propose it suffices to look among $O(\log n)$ elements in $\tilde{\pi}_x$ to find the y to which the proposal will be made. Since the matching process is expected to make $O(n \log n)$ proposals, it is expected to require $O(n(\log n)^2)$ comparisons. It turns out that, during these steps, evolution creates a blocking pair with probability at most α/n. Therefore we expect $O((\log n)^2)$ blocking pairs.

We now provide the details of the proof. In order to bound the number of blocking pairs, it is crucial that during the matching process not too many queries are made, or alternatively that not too many proposals are made. We therefore need an analog of Theorem 1. To apply the coupon-collecting argument from the proof of that theorem we prove the following lemma.

▶ **Lemma 9.** *Provided π_x was chosen uniformly at random at time-step 0 and only comparison queries are made, the element y chosen at line 7 of Algorithm 5 is a random element from the subset of B to which x has not by that point made any proposals.*

Proof. The proof of the lemma relies on the fact that the dynamic quicksort algorithm used to obtain $\tilde{\pi}_x$ for $x \in A$ and the procedure used to find the best element in line 7 of Algorithm 5 only use comparison queries.

Let π_x be the preference list of x before the first comparison is queried. Fix an arbitrary

sequence of evolution steps that occurs during the computation of y. Suppose that given these choices of nature, $y = \pi_x(k)$. Then, given the same evolution steps, for any other permutation π'_x, $y = \pi'_x(k)$. Since π_x is a random permutation, $y = \pi_x(k)$ is a random element of σ_x. ◀

▶ Remark 10. We remark that the requirement on the implementation of QUICKSORT and `best` using only comparison queries is necessary and the lemma does not hold for an arbitrary algorithm.

▶ **Lemma 11.** *The number of proposals during one execution of the matching process is $O(n \log n)$ in expectation and with high probability.*

Proof. Suppose x proposes to y and at that point k elements of B are unmatched. Note that it must be the case that x has not proposed to any of these elements (otherwise, they would not be currently unmatched). Thus, by the previous lemma, each of these k elements receives a proposal with probability at least $1/n$. The stated bound follows from the analysis of coupon collector's problem as in Theorem 1. ◀

As in the one-sided setting, the analysis will rely on estimating the occurrence of a specific kind of critical evolution steps. In the present case the definition of a critical evolution step is a little more involved than its one-sided counterpart.

▶ **Definition 12.** An evolution event on the preference list π_z of $z \in A \cup B$ is *critical* if one of the following holds:

1. An evolution event involves a swap of the *then* match of z, $M(z)$.
2. If $z \in A$, the evolution event involves swapping the *then* best element as per π_z to which z has not yet proposed.

The following lemma establishes the link between the critical evolution steps and the number of blocking pairs.

▶ **Lemma 13.** *Assume that for the duration of one run of the matching process, the preference lists $\{\pi_z \mid z \in A \cup B\}$ satisfy the conditions of Lemma 4, and suppose that (x, y) is a blocking pair with respect to the returned matching. Then there was a critical evolution event on the preference list of at least one of x and y during the execution of the matching process.*

Proof. First consider the case that x proposed to y during the execution of the matching process. It follows that y rejected x at some point in favor of some other element. When x was rejected, the *then* $M(y)$ satisfies $M(y) \prec_{\pi_y} x$. Subsequently, $M(y)$ may change but has to become better, unless there was a swap that involves the *then* $M(y)$, which is a critical event on π_y. Since, we know that in the end $x \prec_{\pi_y} x'$, where x' is the final match of y, there must have been some evolution event where x was swapped with the *then* match of y. Thus, by part 1 of Definition 12, a critical evolution event occurred on π_y.

On the other hand, suppose x never proposed to y, and let y' be the final match of x. Suppose that when x proposed to y', $y \prec_{\pi_x} y'$; it follows that `best` on line 7 of Algorithm 5 failed to return the best element to which x had not yet proposed. Since, we are assuming that $\tilde{\pi}_x$ is a sufficient approximation of π_x, it must be because the actual *best* element was swapped at least once while `best` was being executed. Thus, by part 2 of Definition 12, a critical evolution event occurred on π_x. Finally, if when x proposed to y', it was the case that $y' \prec_{\pi_x} y$, but (x, y) is a blocking pair, it must be that y' was involved in a swap subsequently leading to a critical event involving x's the *then* match. ◀

▶ **Theorem 14.** *Provided the initial preference lists are drawn randomly,*[4] *for all $z \in A \cup B$, for a sufficiently large constant c_0 and any time-step $t \geq c_0 n^2 \log n$, Algorithm 5 maintains a matching with at most $O((\log n)^2)$ blocking pairs in expectation and with high probability.*

Proof. As a result of Lemma 13, we can estimate the number of blocking pairs by estimating the number of critical evolution steps. Let T be a random variable equal to the number of queries of Algorithm 5 before it outputs a matching and label the corresponding time-steps as $1, 2, \ldots, T$. For each step k, let X_k be a Boolean random variable that is equal to 1 if at the time-step labeled k the evolution was critical. Furthermore, denote by $\mathcal{E}$ the event that during these T time-steps the lists were approximately sorted. By Lemma 4, the event $\mathcal{E}$ occurs with high probability.

By Lemma 11 it follows that $T = O(n(\log n)^2)$ in expectation and with high probability, since we are wasting $O(\log n)$ queries per proposal. Therefore, we may fix an appropriately large constant C so that $T \leq Cn(\log n)^2$ with high probability and let $m = Cn(\log n)^2$. Note that—given any history of evolution steps—an evolution step is critical with probability at most $O(\alpha/n)$, since for each $z \in A \cup B$ there is a constant number of elements that evolution has to swap in order to be critical. Thus, by Theorem 5,

$$\mathbb{P}\Big[\sum_{k=1}^{m} X_k > 2C(\log n)^2\Big] = O(n^{-C}),$$

for sufficiently large C. We have

$$\begin{aligned}\mathbb{P}\Big[\sum_{k=1}^{T} X_k > 2C(\log n)^2\Big] &\leq \mathbb{P}\big[T > Cn(\log n)^2\big] + \mathbb{P}[\bar{\mathcal{E}}] \\ &\quad + \mathbb{P}\Big[\Big(\sum_{k=1}^{T} X_k > 2C(\log n)^2\Big) \wedge \big(T \leq Cn(\log n)^2\big) \wedge \mathcal{E}\Big] \\ &\leq \mathbb{P}\big[T > Cn(\log n)^2\big] + \mathbb{P}[\bar{\mathcal{E}}] + \mathbb{P}\Big[\sum_{k=1}^{m} X_k > 2C(\log n)^2\Big].\end{aligned}$$

It follows that the claimed bound holds with high probability. The bound on the expectation follows by noting that there can be at most n^2 blocking pairs. ◀

Acknowledgments. For earlier discussions, F.M. would like to thank Marcos Kiwi who, among other things, introduced him to the line of works on evolving data and shared preliminary thoughts on the possibility of computing stable matchings in this context. We also thank a reviewer for (minor) corrections in our main theorem.

References

1 A. Anagnostopoulos, R. Kumar, M. Mahdian, and E. Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011. Selected Papers from 36th International Colloquium on Automata, Languages and Programming. doi:10.1016/j.tcs.2010.10.003.

[4] This is not actually required, since after sufficiently long (though still polynomial) time, all the preference lists will be close to random due to a mixing time argument on the set of permutations.

2 A. Anagnostopoulos, R. Kumar, M. Mahdian, E. Upfal, and F. Vandin. Algorithms on evolving graphs. In *Proc. of 3rd Innovations in Theoretical Computer Science*, 2012.

3 D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, 1999.

4 D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

5 M. Gupta and R. Peng. Fully dynamic (1+ e)-approximate matchings. In *Proc. of 54th IEEE Foundations of Computer Science*, pages 548–557, Oct 2013. doi:10.1109/FOCS.2013.65.

6 D. Knuth. *Stable Marriage and Its Relation to Other Combinatorial Problems: An Introduction to the Mathematical Analysis of Algorithms*. CRM proceedings & lecture notes. American Mathematical Society, 1997.

7 David Asher Levin, Yuval Peres, and Elizabeth Lee Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009.

8 C. McDiarmid. Concentration. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, volume 16 of *Algorithms and Combinatorics*, pages 195–248. Springer Berlin Heidelberg, 1998. doi:10.1007/978-3-662-12788-9_6.

9 R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge International Series on Parallel Computation. Cambridge University Press, 1995.

10 O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proc. of 45th ACM Symposium on Theory of Computing*, pages 745–754, 2013. doi:10.1145/2488608.2488703.

11 K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proc. of 42nd ACM Symposium on Theory of Computing*, pages 457–464, 2010. doi:10.1145/1806689.1806753.

12 L. B. Wilson. An analysis of the stable marriage assignment algorithm. *BIT Numerical Mathematics*, 12(4):569–575, 1972.