Matching with our Eyes Closed

Gagan Goel
Google Research
New York, USA
gagangoel@google.com

Pushkar Tripathi
Georgia Institute of Technology
Atlanta, USA
pushkar.tripathi@gatech.edu

Abstract—Motivated by an application in kidney exchange, we study the following query-commit problem: we are given the set of vertices of a non-bipartite graph G. The set of edges in this graph are not known ahead of time. We can *query* any pair of vertices to determine if they are adjacent. If the queried edge exists, we are *committed* to match the two endpoints. Our objective is to maximize the size of the matching.

This restriction in the amount of information available to the algorithm constraints us to implement myopic, greedy-like algorithms. A simple deterministic greedy algorithm achieves a factor 1/2 which is tight for deterministic algorithms. An important open question in this direction is to give a randomized greedy algorithm that has a significantly better approximation factor. This question was first asked almost 20 years ago by Dyer and Frieze [9] where they showed that a natural randomized strategy of picking edges uniformly at random doesn't help and has an approximation factor of 1/2 + o(1). They left it as an open question to devise a better randomized greedy algorithm. In subsequent work, Aronson, Dyer, Frieze, and Suen [2] gave a different randomized greedy algorithm and showed that it attains a factor 0.5 + ε where ε is 0.0000025.

In this paper we propose and analyze a new randomized greedy algorithm for finding a large matching in a general graph and use it to solve the query commit problem mentioned above. We show that our algorithm attains a factor of at least 0.56, a significant improvement over 0.50000025. We also show that no randomized algorithm can have an approximation factor better than 0.7916 for the query commit problem. For another large and interesting class of randomized algorithms that we call vertex-iterative algorithms, we show that no vertex-iterative algorithm can have an approximation factor better than 0.75.

Keywords-Matching; Randomized Algorithms; Combinatorial Optimization

I. Introduction

The matching problem has been a corner-stone of combinatorial optimization and has received considerable attention starting from the work of Jack Edmonds [10]. The problem finds applications in diverse fields such as online advertising and several barter exchange settings [20], [25]–[29]. Even though efficient combinatorial algorithms for the maximum matching problem are known [10] [22], these algorithms require complete information about the underlying graph and rely on building global combinatorial structures such as *blossoms*.

However this may not be feasible in some settings where the underlying graph may be hidden and the algorithm is constrained to make decisions based solely on locally available information. For example consider the kidney exchange problem - Often patients with a kidney disease have a family member who is willing to donate his/her kidney. Unfortunately, these donors are sometimes blood-type incompatible. To solve this problem, a kidney exchange is performed in which patients swap their incompatible donors to get a compatible donor. Owing to the cost involved in medical tests, incentive issues, and due to ethical concerns, it is desired that an exchange is performed whenever the test indicates that the exchange is possible. To increase the efficiency of the kidney exchange program, it is important to match the maximum number of compatible patient-donor pairs. Refer to [27]–[29] and references there-in for more details.

Formally this problem can be modeled as a maximum matching problem in a general non-bipartite graph where each patient-donor pair represents a node of the graph and an edge between two nodes indicate if an exchange is possible. For every pair $(u, v) \in V \times V$ we are *not* told a priori whether there is an edge connecting these vertices, until we *probe/scan* this pair. If we scan a pair of vertices and find that there is an edge connecting them, we are constrained to *pick* this edge and in this case both u and v are removed from the graph. However, if we find that u and v are not connected by an edge, they continue to be available to be matched in the future. The goal is to maximize the number of vertices that get matched. This problem is known as the *query-commit* problem. Various variants of this model have received considerable attention [5] [3] [23] recently.

It is easy to see that any algorithm that probes all permissible edges is a $1/2$ approximation since we are guaranteed to pick a maximal matching which is at least half as big as the largest matching in the underlying graph. One can also construct examples to show that no deterministic algorithm can do strictly better than $1/2$ on all instances. Thus our only hope is to seek a randomized algorithm that beats the barrier of $1/2$ in expectation.

Arguably the most natural randomized algorithm is the one that tests all vertex pairs in a random order. However this algorithm can have an approximation factor as bad as $1/2 + o(1)$. For example consider a graph over n vertices, such that half the vertices form a clique, while for the other

half of the vertices, each one of them is adjacent to a unique vertex in the clique. The above naive randomized algorithm would attain a factor of only $1/2 + \log n/n$ for such graphs.

Hence, to get a better approximation factor, one could possibly use correlated randomness (as opposed to sampling edges at random in every step) to determine the order in which the edges should be scanned. This idea was first used by Aronson, Dyer, Frieze, and Suen in [2] where they gave the following randomized algorithm for the maximum matching problem - Pick a vertex at random and match it to one of its unmatched neighbors uniformly at random. They showed that this algorithm does marginally better than 0.5 and attains a factor of 0.50000025. Note that this algorithm can be simulated in the query-commit model by first picking a random vertex u and then scanning pairs (u, v) for all unmatched vertices $v \in V$ in a random order, until u gets *matched* or it is established that u cannot be matched with the remaining unmatched vertices.

A. Our Results

From the above discussion it is clear that choosing the correct correlated randomization to determine the order in which the edges are scanned is critical to get a good approximation factor. In this work, we propose and analyze the following randomized algorithm (referred to as SHUF-FLE): Shuffle the vertices according to a uniformly random permutation ρ and iterate through them one at a time. If the current vertex is already matched then ignore it else scan edges incident to it in the order dictated by ρ until it gets matched or there are no vertices left. Then proceed to the next vertex.

Note that our algorithm differs from the algorithm given by [2] in the sense that after picking a random vertex u, we scan all the edges (u, v) for $v \in V$ in a *specific order* that is given by the randomization used to pick the vertex u itself, instead of scanning them in a random order, as suggested by [2].

In this paper, we show the following results:

(a) SHUFFLE attains a factor of 0.56 for the query-commit problem, and thus improves upon the 0.50000025-factor algorithm given by Aronson, Dyer, Frieze, and Suen in [2].

(b) There exists a family of graphs for which SHUFFLE attains a factor no better than 0.727.

(c) We show that no randomized algorithm can attain a factor better than 0.7916 for the query-commit problem.

(d) We also show a tighter bound for a large class of algorithms called vertex iterative (VI) algorithms. Both SHUFFLE and the algorithm in [2] fall in this class. A VI algorithm considers the vertices one at a time and for every vertex probes edges incident on it until it gets matched. A vertex may also choose to "give-up"

in which case it plays no further part in the algorithm and we move on to the next vertex. The order in which the vertices are scanned, and the sequence in which we scan the edges may be determined adaptively i.e. it can depend on the past outcomes. For this general class of algorithms we show that no randomized algorithm can attain a factor better than 0.75.

B. Related Work

As mentioned earlier, the query-commit model essentially limits us to implement myopic Greedy-like algorithms. Since the greedy algorithms are easy to implement and can be easily adapted to different environments, for instance distributed setting, they are quite prevalent in practice. As a result, there is some interest in designing better randomized greedy algorithms that outperform the 0.5 approximation of a deterministic greedy algorithm. Dyer and Frieze [9] studied the greedy algorithm where an edge is picked uniformly at random among the unmatched vertices. They showed that for general graphs it doesn't give any improvement asymptotically, and for sparse graphs it significantly improves the approximation factor. Later, [2] gave a different randomized greedy algorithm that achieves a factor 0.50000025 for general graphs.

The RANKING algorithm of Karp, Vazirani, and Vazirani [16] and the subsequent work of [14], [18] can be thought of as randomized greedy algorithms for bi-partite graphs. The former giving an approximation factor of $1 - 1/e$, and the latter 0.69. In other related work, designing fast algorithms for finding approximate maximum matchings has also received considerable attention recently [8], [24], [30]. However all these algorithms explicitly exploit the edge structure of the graph and are not applicable in our setting.

Some other examples of using randomization to give simple efficient algorithms include the seminal work of Karger [15] where randomization was used to derive a simple and efficient algorithm for min-cut. More recently Costello et. al. [6] considered randomized versions of the Johnson's algorithm [13] for the satisfiability problem and showed how the simple greedy algorithm can be tweaked through randomization to do better than $2/3$.

Stochastic Kidney Exchange Problem: In the stochastic setting of the query commit problem, we are given probabilities p_{uv} for the existences of every edge (u, v). Recently Molinaro and Ravi [23] gave an optimal approximation algorithm for sparse graphs in this setting. Prior to this Chen et al. [5] considered a generalization of this model. In their model they associate a *patience parameter* with every vertex $v \in V$ $t(v)$ bounding the maximum number of failed probes v is willing to participate in. After $t(v)$ failed attempts, vertex v would leave the system, and would not be considered for any further matches. They gave a $1/4$ competitive algorithm for this problem.

The results of [5] were later extended to the weighted setting by Bansal et al. [3] who used a linear program to bound the performance of the optimal algorithm and gave a $1/4$ competitive algorithm for the general case, and a $1/3$ competitive algorithm for the special case of bipartite graphs.

In a recent paper, Costello et al. [7] studied another variant of the stochastic kidney exchange model omitting the patience parameter and gave upper and lower bounds for the same.

C. Technical Contributions

Conquering Non-monotonicity: A vast body of prior work [1] [4] [11] [14] [16] [18] on online allocation problems has relied on a crucial property called *monotonicity*. This property is formally defined in Section II-B, but intuitively it states that adding new vertices can not cause one of its previously matched neighbors to get unmatched. This property holds for bipartite graphs for most natural algorithms like RANKING, but unfortunately as shown by the simple example in Figure 1, it does not hold for all graphs in general. Observe that at the heart of the example is an odd cycle which are not present in bipartite graphs.

Thus the lack of monotonicity is a massive blow in the quest for an algorithm for the query-commit problem. We overcome this obstacle by defining another property called *stability*. This property states that adding a vertex can only alter the matching found by SHUFFLE by at most a single augmenting path. Using this observation we show that every time a vertex does not conform to the monotonicity property in SHUFFLE it results in a certain types of *good-events*. Finally we bound the approximation factor for the algorithm in terms of the number of these good events.

Strongly Factor Revealing Family of Linear Programs: In our analysis we prove several combinatorial lemmas to lower bound the performance of SHUFFLE. We then coalesce them into a large linear program parameterized by the size of the input, that lower bounds the approximation factor. However the linear program is quite complicated and not amenable to traditional means of analysis that are used to study factor revealing linear programs (refer to [21] [17] [12] [19]). Our analysis is based on a technique recently introduced in [18]. In this method we use these parameterized LPs to derive a new family of LPs (strongly factor revealing family) each of which lower bounds the approximation factor SHUFFLE for *any* given input size. Thus solving any large enough instance from this family serves to bound the performance of SHUFFLE.

Upper Bound using Yao's Lemma [31]: We consider a general class of algorithms called vertex iterative algorithms and prove a lower bound on the performance of any such algorithm using Yao's Lemma [31]. This entails finding the best deterministic algorithm for an appropriately chosen distribution over the inputs. Unfortunately it turns out to be quite difficult to characterize the optimal deterministic algorithm for the distribution that we consider in our proof. Instead we define a class of fictitious deterministic algorithms called *revealing-algorithms* and show that they perform at least as well as any deterministic vertex iterative algorithm. Then to analyze the optimal revealing-algorithm for our distribution we define another randomized revealing algorithm and argue that its expected performance is at least as good as the best deterministic revealing algorithm. Finally we bound to performance of this randomized algorithm to complete our proof.

II. PRELIMINARIES

A. Problem Statement

Let $G(V, E)$ be a non-bipartite graph where $|V| = n$. We wish to analyze the performance of the following algorithm for the query commit problem.

SHUFFLE Algorithm
1) Choose a uniformly random permutation ρ of the vertices.
2) Apply permutation ρ to V - thereby assigning each vertex a priority or rank.
3) Process the vertices one at a time in the increasing order of the rank. If the vertex under consideration, say vertex u, is already matched then ignore it, else scan edges incident to it from the unmatched vertices in an increasing order of their rank until vertex u gets matched. Then proceed to the next vertex.

We will use u, v to denote the vertices of V and $s, t \in [n]$, to index the vertices in V. Let ALG(G) be the expected size of the matching returned by the above algorithm and let OPT(G) be the largest matching in G. We say that SHUFFLE attains a factor of α if $ALG(G) \geq \alpha |OPT(G)|$ for every graph G.

We also use the notion of time: time t will denote the event when the tth vertex in V is considered by the algorithm. Define Ω_V to be the set of all permutations of V. For any permutation ρ of the vertices, we will use $\rho(t)$ to denote the vertex at the tth position in ρ and $\rho^{-1}(u)$ to denote the position of the vertex u in ρ. For vertices u, v we say u is above v in ρ if $\rho^{-1}(u)$ is less than $\rho^{-1}(v)$. We can similarly define the notion of a vertex u being below another vertex v. For any permutation $\rho \in \Omega_V$ let $\rho[u \rightsquigarrow s]$ denote the permutation obtained by moving u to position s keeping the order of the other vertices unchanged.

For any $t \in [n]$ define x_t to be the probability that the vertex $\rho(t) \in V$ at position t gets matched in SHUFFLE, where the probability is taken over the random choices of ρ. We will use $Shuffle(\rho)$ to denote an invocation of SHUFFLE with ρ as the permutation chosen in the first step. In general for any algorithm $\mathcal{A}$ and graph G we will use $\mathcal{A}(G)$ to denote the performance of $\mathcal{A}$ on G.

B. Properties of the SHUFFLE Algorithm

In this section we will introduce some terminology that would be central to our analysis.

Monotonicity Property: Consider $u = \rho(t)$ that is unmatched by $Shuffle(\rho)$. Suppose v is another vertex that is adjacent to u, i.e. $uv \in E$, then v is surely matched to a vertex above u. The monotonicity property strengthens this observation, by allowing us to translate u to any of the n positions in ρ. Concretely, we say a vertex u satisfies the monotonicity property with respect to v for a given permutation ρ if,

1) $uv \in E$
2) u is unmatched in $Shuffle(\rho)$
3) v is matched in $Shuffle(\rho)$
4) v is matched above position t in $Shuffle(\rho[u \rightsquigarrow s])$ for every $s \in [n]$

Non-monotone Event: Unfortunately some vertices may not satisfy the monotonicity property for all choices of s (in the fourth point above). This prompts us to define a non-monotone event. For a given permutation ρ, a vertex $u = \rho(t)$ is involved in a non-monotone event with respect to another vertex v and position s if the following conditions are met.

1) $uv \in E$
2) u is unmatched in $Shuffle(\rho)$
3) v is matched in $Shuffle(\rho)$
4) v is matched below position t or unmatched in $Shuffle(\rho[u \rightsquigarrow s])$

That is, promoting u in ρ causes one of its previously matched neighbors to match lower or to get unmatched. We use $\Gamma^v(s, t, \rho)$ to be the indicator variable for this event. For illustration, consider the example shown in Figure 1.
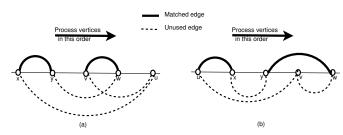

Figure 1: Non-monotone event generated by translating u

The matched edges are shown in bold while the other edges are indicated by dashed lines. Here we consider two permutations $\rho = [x, y, v, w, u]$ (Figure 1a) and $\rho' = [u, x, y, v, w]$ (Figure 1b). Note that moving u to the start of the permutation ρ causes one of its previously matched neighbors v to get unmatched in $Shuffle(\rho')$. In this case we say u has *generated* a non-monotone event by moving to position 1, i.e., $\Gamma^v(1, 5, \rho) = 1$ for the above example.

Observe that $\{x, y, w, v, u\}$ form a cycle with odd number of vertices. In fact every non-monotone can be associated with an odd cycle. This explains why such events do not occur when the underlying graph is bipartite.

Stability Property: At a high level this property states that for any $\rho \in \Omega_V$ the execution of $Shuffle(\rho)$ does not change dramatically by altering the position of a single vertex in ρ. For any graph $H(V, E)$ consider $\rho \in \Omega_V$. For an arbitrary vertex $u \in V$ let $\rho' = \rho[u \rightsquigarrow s]$. Consider the execution of $Shuffle$ for graph H and for any $v \in V$, define $M(v, \rho)$ to be the set of matched edges when v was considered in $Shuffle(\rho)$. Similarly let $M(v, \rho')$ be the set of matched edges when v was considered by $Shuffle(\rho')$. The stability property is summarized by the following claim that can be proved by induction.

Claim 1. *For all $v \in V - u$, the symmetric difference of $M(v, \rho)$ and $M(v, \rho')$ has at most one component, i.e., it is either empty or a path or a cycle.*

Good-Events: We will use a notion of good-events to quantify our algorithm's improvement over the greedy algorithm that matches exactly half the vertices in the worst case. A pair of vertices that are matched to each other in the optimal solution would correspond to some good-events if they both are also matched (even if not to each other) in an execution of the algorithm. We will attribute any such pair of vertices to two types of good events that are defined below.

Type 1 good-event: For any given permutation $\rho \in \Omega_V$ under consideration, we say a good-event of type 1 happens at position t if both $\rho(t)$ and its partner(if any) in OPT(G) are matched by $Shuffle(\rho)$. We will use $Good_1(t, \rho)$ as the indicator variable for this event.

Type 2 good-event: The second type of good event is defined in a slightly indirect manner. For a permutation $\rho \in \Omega_V$ we say a good event of type 2 happens at position t if the following conditions are met

1) $u = \rho(t)$ is matched in $Shuffle(\rho)$. Let w be the vertex that it is matched to.
2) If w^* is the partner (if any) of w in OPT(G), then w^* is matched in $Shuffle(\rho)$.
3) Consider ρ' that is produced by deleting u from ρ and keeping the relative order of all other vertices the same. Then, both w and w^* are matched to *each other* in $Shuffle(\rho')$.

From the above definitions it is clear that type 2 good events are a subset of type 1 good events. Also note that if OPT(G) matches w to w^*, and both the vertices are matched in our algorithm for a given permutation, then we generate two good events of type 1, one at the position of w and other at the position of w^*. On the other hand, one can show that we will generate at most one good event of type 2, which will be at the position of the vertex u, where u is matched to the vertex that is first to be matched among w and w^*.

III. Analysis of the Shuffle Algorithm

We divide the analysis into two parts. In the first part, we use combinatorial arguments to set up equations relating the variables defined in Section II-B. In the second part of the analysis we use these equations to give a lower bound on the performance of Shuffle through a strongly factor revealing LP.

Intuitively, the input space of graphs has two types - graphs that have *few* non-monotone events(such as bipartite graphs) and those that have *large* number of non-monotone events. In the first case one can leverage the monotonicity property to show that the algorithm has a good approximation factor. The analysis for the second type of graphs is little more involved and relies on the observation that if there are a large number of good-events then we get a good approximation factor. To show that there are non-negligible number of good-events we prove a structural lemma - that forms the core of our arguments - to lower bounds the number of good-events in terms of the number of non-monotone events. To the best of our knowledge, this is the first work that relates good-events to non-monotone events for the matching problems (and, in general, allocation problems), and uses it to show a non-trivial improvement over the standard greedy algorithm.

A. Combinatorial Arguments

We begin by proving that the worst examples for Shuffle must have a perfect matching (for the proof, refer to the full version).

Lemma 1. *We may assume without loss of generality that the graphs for which* Shuffle *attains the worst factor has a perfect matching.*

Throughout the rest of the paper we will assume that the given graph G has a perfect matching. Furthermore, for any $u \in V$, we will use u^* to denote its match in OPT(G) we will refer to u^* (resp. u) as the *partner* of u (resp. u^*) with respect to OPT(G). Also for brevity we will use $\Gamma(s, t, \rho)$ to denote $\Gamma^{u^*}(s, t, \rho)$ where $u = \rho(t)$. Thus $\Gamma(s, t, \rho)$ refers to the non-monotone event where translating an unmatched vertex $u = \rho(t)$ to position s in ρ causes its partner u^* (which was previously matched above t) to get unmatched or to match below t.

The following observation follows easily from the definitions of x_t.

Observation 1. $\forall t \in [n] \; : \quad x_t \geq x_{t+1}$

The next lemma is similar in spirit to Observation 1 above.

Lemma 2. *For any* $s, t \in [n]$ *such that* $s < t$,

$$\sum_{\rho \in \Omega_V} \Gamma(s, t, \rho) \leq \sum_{\rho \in \Omega_V} \Gamma(s, t+1, \rho)$$

Proof: Consider $\rho \in \Omega_V$ such that we generate a non-monotone event $\Gamma(s, t, \rho)$ by promoting $u = \rho(t)$ to

position s. Let $\rho' = \rho[u \rightsquigarrow t + 1]$. Since u is unmatched at position t in ρ it will remain unmatched at position $t + 1$ in $Shuffle(\rho')$. Also translating u from position $t + 1$ to s in ρ' will generate a non-monotone event $\Gamma(s, t + 1, \rho')$. Thus we can set up an injective map from non-montone events of the form $\Gamma(s, t, .)$ to non-monotone events of the form $\Gamma(s, t + 1, .)$. This suffices to prove the above claim. ∎

The following lemma relates x_t and $\Gamma(s, t, \rho)$.

Lemma 3. *For every* $s, t \in [n]$,

$$|\Omega_V|(1 - x_t) \geq \sum_{\rho \in \Omega_V} \Gamma(s, t, \rho)$$

Proof: The total number of times the vertex at position t remains unmatched is given by $|\Omega_V|(1 - x_t)$. Since a necessary requirement(by property 2 in the definition) of a having a non-montone event $\Gamma(s, t, \rho)$ is that $\rho(t)$ is unmatched in $Shuffle(\rho)$, the number of such non-monotone events is at most $|\Omega_V|(1 - x_t)$. ∎

Lemma 4 follows from similar arguments as above.

Lemma 4. *For every* $t \in [n]$,
1) $|\Omega_V| x_t \geq \sum_{\rho \in \Omega_V} Good_1(t, \rho)$
2) $|\Omega_V| x_t \geq \sum_{\rho \in \Omega_V} Good_2(t, \rho)$

We will now present three bounds on ALG(G). The first follows trivially from the definition of x_t.

Observation 2.

$$ALG(G) = \sum_{t \in [n]} x_t$$

The next two bounds proved in Lemmas 5 and 6 are slightly more involved and illustrate the role of good-events (defined in Section II-B) in attaining a factor better than 0.5.

Lemma 5.

$$ALG(G) = \frac{n}{2} + \frac{\sum_t \sum_{\rho \in \Omega_V} Good_1(t, \rho)}{2|\Omega_V|}$$

Proof: By Lemma 1 we may assume, without loss of generality, that the underlying graph has a perfect matching. Any algorithm(including Shuffle) that returns a maximal matching would surely match at least one of w and w^* for any vertex $w \in V$ and therefore matches at least $n/2$ vertices.

To find the total number of vertices that get matched we also need to add the number of pairs (w, w^*) such that both w and w^* are matched. Since each such pair contributes twice to the summation $\sum_t \sum_{\rho \in \Omega_V} Good_1(t, \rho)$, the expected number of vertices matched by Shuffle is given by $\frac{n}{2} + \frac{\sum_t \sum_{\rho \in \Omega_V} Good_1(t, \rho)}{2|\Omega_V|}$. ∎

Lemma 6.

$$ALG(G) \geq \frac{n}{2} + \frac{\sum_t \sum_{\rho \in \Omega_V} Good_2(t,\rho)}{|\Omega_V|}$$

Proof: As before by Lemma 1 we may assume, that the underlying graph has a perfect matching. Thus any greedy algorithm(including SHUFFLE) matches at least $n/2$ vertices. To bound the total number of vertices that get matched we also need to add the number of pairs (w, w^*) such that both w and w^* are matched. We do this by constructing a map from good-events of type 2 into the set of such pairs.

Consider a good-event $Good_2(t,\rho)$ and let $u = \rho(t)$. Let w be the vertex that u is matched to in $Shuffle(\rho)$. By definition of a good-event of type 2, w^* is also matched in $Shuffle(\rho)$. Thus each type 2 good-event maps to a pair (w, w^*) such that both w and w^* are matched. Also observe that this map is one-one, i.e. given a pair (w, w^*) that lies in the range of this map there is exactly one good-event of type 2 that could be its preimage. This is because if we define $\hat{w} \in \{w, w^*\}$ to be the first among w and w^* to get matched in $Shuffle(\rho)$ then the neighbor of $\hat{w}$ in $Shuffle(\rho)$ uniquely defines the vertex u that if deleted would cause w and w^* to be matched. ∎

The following lemma illustrates the connection between non-montone events and good-events. It is the pivotal technical lemma in the analysis. Finally in Lemma 8 we relate the number of non-monotone events bounds to the total number of miss events.

Lemma 7. *For every $t \in [n]$,*

$$\sum_{\rho \in \Omega_V} \Gamma(t,n,\rho) \leq \sum_{\rho \in \Omega_V} Good_2(t,\rho) + \frac{n \sum_{s \leq t} \sum_{\rho \in \Omega_V} Good_1(s,\rho)}{t}$$

Proof: The proof of this Lemma relies on setting up two maps that associate good-events (of type 1 or 2) with every non-monotone event. Let $\rho \in \Omega_V$ such that $\Gamma(t,n,\rho) = 1$. Let $u = \rho(n)$ and $\rho' = \rho[u \rightsquigarrow t]$. Clearly u is matched by $Shuffle(\rho')$, let $w \neq u^*$ be the vertex that it is matched to. Let w^* be the partner for w in $OPT(G)$. We consider the following two cases.

Case 1: Both w and w^* were matched to each other in $Shuffle(\rho)$.

Since we have a non-monotone event when we move u to position t, w^* should surely be matched in $Shuffle(\rho')$. Thus in this case moving u to position t in ρ generates a good-event of type 2 at position t in $Shuffle(\rho')$. Thus in this case, every non-monotone event $\Gamma(t,n,\rho)$ generates exactly one good event of type 2. This is represented by the first term on the right-hand side of the above inequality.

Case 2: w and w^* were not matched to each other in $Shuffle(\rho)$.

Suppose we move w^* around in ρ', i.e. consider $\rho'' = \rho'[w^* \rightsquigarrow s]$ for $s \in [n]$ then we have the following two claims. The first claim says that w would always stay matched, while the second claim says that for at least(the top) t positions of w^* both w and w^* will get matched. We defer the proofs to the appendix.

Claim 2. *w is matched in $Shuffle(\rho'[w^* \rightsquigarrow s])$ for all choices of $s \in [n]$.*

Claim 3. *w^* is matched by $Shuffle(\rho'[w^* \rightsquigarrow s])$ for all choices of $s \leq t$, $s \in [n]$.*

By Claim 2 w stays matched for all positions of w^*. Of the n positions that w^* can take, it too gets matched for at least t positions. Thus we generate at least t good-events of type 1 (for the top t positions of w^*) by moving w^* over ρ' from every non-monotone event $\Gamma(t,n,\rho)$.

However, we may over count while accounting for good-events since the same type 1 good-event can be generated from multiple non-montone events. Let us fix t for which we are writing the equation. Consider a good-event $Good_1(s,\hat{\rho})$ generated as defined above. Let $u = \hat{\rho}(t)$ and $w^* = \hat{\rho}(s)$. Let us count the number of non-montone events $\Gamma(t,n,\rho)$(characterized by permutation ρ) that may generate this good-event. The positions of all vertices except w^* and u are predetermined by their positions in $\hat{\rho}$. Now u should surely be at position n in ρ, but the position of w^* remains ambiguous. It may be at any of the n positions in ρ. Thus in the worst case there may be n ways to choose ρ such that the non-montone event $\Gamma(t,n,\rho)$ generates the good-event $Good_1(s,\hat{\rho})$. Thus every non-monotone event generates t good-events of type 1 and each such good-event may be generated by as many as n non-monotone events. Substituting the variables gives us the second term in the above inequality. ∎

Lemma 8. *For every $t \in [n]$,*

$$n(1 - x_t) \leq \sum_{s \leq t} x_s + \frac{\sum_{s \leq t} \sum_{\rho \in \Omega_V} \Gamma(s,t,\rho)}{|\Omega_V|}$$

Proof: We will prove this lemma by defining a function that maps a miss event at position t to n events that are either match events above t or to a non-monotone events above t. Consider $\rho \in \Omega_V$ such that $u = \rho(t)$ is unmatched. Consider $Shuffle(\rho[u \rightsquigarrow s])$ for all values of $s \in [n]$ and suppose we analyse what happens to u^* in each of these events. It will either continue to be matched above position t or it may get matched below position t(or get unmatched). In the former case we generate a match event above t, while in the latter we generate a non-monotone event at s, $\Gamma(s,t,\rho)$. One can check that the set of events generated for distinct miss events are disjoint.

There are $|\Omega_V|(1 - x_t)$ miss events in all and the total number of match events above t is given by $|\Omega_V| \sum_{s \leq t} x_s$.

Using the above map we have

$$n|\Omega_V|(1 - x_t) \leq |\Omega_V| \sum_{s \leq t} x_s + \sum_{s \leq t} \sum_{\rho \in \Omega_V} \Gamma(s, t, \rho)$$

Dividing throughout by $|\Omega_V|$ proves the lemma. ∎

In the next section we will demonstrate how the above structural Lemmas can be used to show that SHUFFLE attains a factor of at least 0.560.

B. Strongly Factor Revealing Linear Program

Let us define g_t to be the probability of a type 1 good event at position t, i.e. $g_t = \frac{\sum_{\rho \in \Omega_V} Good_1(t, \rho)}{|\Omega_V|}$. Similarly define $h_t = \frac{\sum_{\rho \in \Omega_V} Good_2(t, \rho)}{|\Omega_V|}$ and $\gamma_{s,t} = \frac{\sum_{\rho \in \Omega_V} \Gamma(s, t, \rho)}{|\Omega_V|}$. In light of the above definitions the results of Section III-A can be reformulated as the following linear program.

LP(n) : minimize $\dfrac{ALG}{n}$

Subject to

$$ALG = \sum_{t \in [n]} x_t \qquad \text{Observation 2}$$

$$ALG = \frac{n}{2} + \frac{1}{2} \sum_t g_t \qquad \text{Lemma 5}$$

$$ALG \geq \frac{n}{2} + \sum_t h_t \qquad \text{Lemma 6}$$

$$\gamma_{t,n} \leq h_t + \frac{n \sum_{s \leq t} g_s}{t} \qquad \forall t \in [n] \qquad \text{Lemma 7}$$

$$n(1 - x_t) \leq \sum_{s \leq t} x_s + \sum_{s \leq t} \gamma_{s,t} \quad \forall t \in [n] \qquad \text{Lemma 8}$$

$$1 - x_t \geq \gamma_{s,t} \qquad \forall s, t \in [n] \qquad \text{Lemma 3}$$

$$x_t \geq x_{t+1} \qquad \forall t \in [n] \qquad \text{Observation 1}$$

$$\gamma_{s,t} \leq \gamma_{s,t+1} \qquad \forall s, t \in [n] \qquad \text{Lemma 2}$$

$$x_t \geq g_t \qquad \forall t \in [n] \qquad \text{Lemma 4}$$

$$x_t \geq h_t \qquad \forall t \in [n] \qquad \text{Lemma 4}$$

$$0 \leq x_t, \gamma_{s,t}, g_t, h_t \leq 1 \qquad \forall s, t \in [n]$$

Here we are trying to minimize the approximation factor of the algorithm subject to the constraints derived in Section III-A. The following lemma follows immediately.

Lemma 9. *LP(n) lower bounds the performance(approximation factor) of* SHUFFLE *on any graph having n vertices.*

Table I gives the optimal value of LP(n) for different choices of the parameter n.

There are two main approaches towards rigorously estimating approximation factors using factor revealing LPs. The first method relies on analytically solving the given LP for an arbitrary sized input and then arguing about the optimality of the solution (refer to [21] [17]). The second technique relies on observing patterns in the dual for the

n	Optimal Value of LP(n)	Factor = LP(n)/n
20	11.124	0.5562
50	27.913	0.5582
100	55.899	0.5588
200	111.870	0.5593
300	157.879	0.5596
400	224.001	0.560

Table I: Optimal Values of LP(n)

given LP and using these to guess a near optimal dual for an arbitrary program (e.g. [12] [19]). This approach is usually lossy and fails to attain the optimal approximation factor. Both of these approaches are infeasible in our context since the linear program (and its dual) at hand is quite complex and not amenable to analytical study. Therefore, in order to rigorously establish that SHUFFLE does attain a factor of at least 0.560 we will employ a technique inspired by the recent work by [18].

In this technique we construct a family of progressively stronger linear programs (called a *strongly factor-revealing family*) such that the solution to any of these programs lower bounds the approximation ratio of the algorithm. Towards this end we will in fact prove that the optimal value of $LP(k)$ for any small constant k lower bounds the optimal value of $LP(n)$ for an any choice of $n >> k$. Thus our family of strongly factor-revealing linear programs is simply $\{LP(1), LP(2) \cdots LP(k)\}$. Concretely, we will show LP-OPT$(k) \leq$ LP-OPT(n), where LP-OPT(n) is the optimal value of $LP(n)$. Refer to the full version of the paper for the complete proof.

Lemma 10. *For any fixed constant k dividing n,*

$$LP\text{-}OPT(k) \leq LP\text{-}OPT(n)$$

Lemma 10 above together with Table 1 suffices to prove the following theorem.

Theorem 11. SHUFFLE *attains a factor of at least 0.560 in expectation.*

We also show the following theorem to bound the tightness of our analysis. We defer the proof to the full version.

Theorem 12. *There exists a graph for which* SHUFFLE *attains a factor of at most 0.727.*

IV. UPPER BOUNDS

In Theorem 13 we provide a bound on the performance of any randomized algorithm for the query commit problem. The proof can be found in the full version.

Theorem 13. *No randomized algorithm can attain a factor better than $19/24 = 0.7916$ for the query commit problem.*

Next we will give an improved upper bound for a broad class of randomized algorithms called *vertex-iterative* algorithms. This class includes the SHUFFLE algorithm as well as the algorithm in [2].

We call a randomized algorithm for the query-commit problem to be *vertex-iterative*(VI) if it iterates through the vertices one at a time in a possibly adaptive sequence and for every vertex scans the edges incident on it in an arbitrary(adaptive) order until it gets matched. A vertex may also choose to scan only a subset of edges incident on it before it decides to "give-up", i.e., after which this vertex will play no further role in the algorithm. For this class of algorithms we prove the following theorem.

Theorem 14. *No randomized* VI *algorithm can attain a factor better than* 0.75 *for the query commit problem.*

Proof:

We use Yao's Lemma [31].

Lemma 15. *[Yao's Lemma] The expected worse-case performance of the optimal randomized algorithm for the query-commit problem is upper bounded by the expected performance for the optimal deterministic algorithm for any given distribution over input graphs.*

From Yao's Lemma above, we have to come up with a distribution over input graphs and bound the performance of the optimal deterministic VI algorithm for this distribution. We will consider the following family $\mathcal{I}$ of n vertex graphs - For any instance $I \in \mathcal{I}$, the vertex set can be divided into two equal parts C(Clique) and P(Pendant). The vertices in C form a clique while every vertex in P is adjacent to a unique vertex in C. For any vertex $u \in C$, let $u^* \in P$ be the unique pendant vertex adjacent to it. An example is shown in Figure 2. We will assume uniform distribution over graphs in $\mathcal{I}$. Alternately, our distribution may be viewed as the uniform distribution over all relabellings ($\phi : [n] \to [n]$) of vertices of the graph Γ shown in Figure 2.
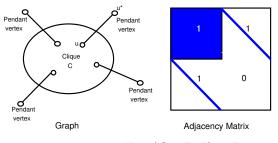

Figure 2: Graph $\Gamma = (C \cup P, E) \in \mathcal{I}$

Unfortunately, it is quite difficult to characterize the optimal deterministic algorithm over graphs in $\mathcal{I}$. Instead we will devise a class of hypothetical algorithms, called *revealing algorithms* whose performance is no-worse than that of any deterministic VI algorithm and then bound their factor. This class of algorithms is hypothetical in the sense that it assumes limited access to the underlying graph $I = (C \cup P, E)$. It uses a notion of *active* and *inactive*

vertices and at the start of the algorithm all vertices are marked to be active.

A revealing algorithm is a VI algorithm that proceeds in several phases. We begin every phase by choosing an arbitrary active vertex say v. If $v \in P$ then it gets matched to its unique neighbor in $v^* \in C$ and we terminate the phase by marking both v and v^* as being inactive. Otherwise if $v \in C$ then it starts querying edges incident to other active vertices (according to a possibly adaptive strategy). If we query an edge incident to a vertex $u^* \in P$ then the u^* *reveals* its unique neighbor $u \in C$ and gets matched to it and both u and u^* are marked as inactive. However if v queries another active vertex $w \in C$, then v and w get matched, as required in the query-commit model, and $\{v, w, v^*, w^*\}$ are marked to be inactive. v may also choose to give-up after trying a few edges, at which point it is marked as inactive (we also mark $v^* \in P$ as inactive at this point). The phase ends when v gets marked as inactive and it is not considered any further by the algorithm. The algorithm terminates when all vertices are marked to be inactive.

Let $\mathcal{O}$ be the optimal deterministic VI algorithm for instances drawn from $\mathcal{I}$. In Lemma 16, we show that there exists a deterministic revealing-algorithm does at least as well as $\mathcal{O}$. We defer the proof to the appendix.

Lemma 16. *There exists a deterministic revealing-algorithm that does at least as well as* $\mathcal{O}$ *on every instance* $I \in \mathcal{I}$.

Corollary 17 follows from Lemma 16.

Corollary 17. *For an instance drawn uniformly at random from* $\mathcal{I}$ *the expected performance of the optimal deterministic revealing-algorithm is at least as good as the expected performance of the optimal deterministic* VI *algorithm.*

Let $\mathcal{A}$ be the optimal deterministic revealing-algorithm for graphs drawn from $\mathcal{I}$. By Lemma 15 and Corollary 17 we only need to bound performance of $\mathcal{A}$. To get a handle on this we define a randomized revealing algorithm Random-Reveal(RR) and then in Lemma 18 we show that the expected performance of RR for the graph Γ shown in Figure 2 is (approximately) equal to the expected performance of $\mathcal{A}$ on graphs drawn uniformly at random from $\mathcal{I}$. Thus we can use the analysis of RR (done in Lemma 19) to upper bound the performance of any VI algorithm.

RR is defined as follows - For any given instance $I = (C \cup P, E)$ we choose a random permutation ρ of $C \cup P$. Then we process the vertices one at a time according to ρ. If the current vertex is inactive then it is ignored else we query edges incident on it in random order until it gets matched. The algorithm terminates when there are fewer than $n^{0.9}$ active vertices.

Remark 1. *The modified termination condition changes the performance of the algorithm by only a negligible factor and is used to simplify the proof.*

Lemma 18. *For the graph Γ shown in Figure 2*

$$\mathbb{E}_{\rho}\left[RR(\Gamma)\right] = \mathbb{E}_{I \in \mathcal{I}}\left[\mathcal{A}(I)\right]$$

Proof: Without loss of generality we may assume that $\mathcal{A}$ is a *greedy* algorithm in the sense that it will never cause a vertex to give-up until it gets matched or has exhausted all its options. The lemma can be proved by establishing the following fact which follows by induction and using the observation that at the end of any phase, the induced graph on the set of active vertices is uniformly distributed over graphs in $\mathcal{I}$ but defined over smaller number of vertices.

Fact: The distribution of the induced graphs over unmatched vertices at the end of the kth phase is identical for both RR and $\mathcal{A}$ for every choice of $k \in [n]$. ∎

Now we are left to analyze the performance of RR on Γ to complete the proof of Theorem 14. We do this in the following lemma.

Lemma 19. *For the graph Γ shown in Figure 2*

$$\mathbb{E}_{\rho}\left[RR(\Gamma)\right] = 0.75$$

Proof: Note that Γ has a unique optimal solution where all n vertices get matched, namely $\{ww^* \mid \forall w \in C\}$. We will argue that for an arbitrary phase, the expected number of vertices that get matched is $3/4$ of the expected number of vertices that are labeled as inactive during the phase. This will establish the above lemma.

For a given phase let v be the active vertex chosen by RR at the start of this phase. The proof for this case is based on the following three simple claims whose proofs can be found in the full version.

Claim 4. $Pr\left[v \in C\right] = Pr\left[v \in P\right]$

Claim 5. *If $v \in P$, then two vertices get matched by RR in the current phase and exactly these two vertices are also labeled to be inactive.*

Claim 6. *If $v \in C$, then 4 vertices get matched in expectation and on an average we label 6 more vertices as being inactive in this phase.*

By Claims 4, 5, 6 we see that the the expected number of vertices that get matched in any phase is given by $2(0.5) + 4(0.5) = 3$ while the expected number of vertices that are marked as inactive is $2(0.5) + 6(0.5) = 4$. Thus RR attains a factor of $3/4$. ∎

References

[1] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. *SODA*, 2010.

[2] Jonathan Aronson, Martin Dyer, Alan Frieze, and Stephen Suen. Randomized greedy matching. ii. *Random Struct. Algorithms*, 6:55–73, January 1995.

[3] Nikhil Bansal, Anupam Gupta, Jian Li, Julian Mestre, Viswanath Nagarajan, and Atri Rudra. When lp is the cure for your matching woes: improved bounds for stochastic matchings. In *Proceedings of the 18th annual European conference on Algorithms: Part II*, ESA'10, pages 218–229, 2010.

[4] B. Birnbaum and C. Mathieu. On-line bipartite matching made simple. 2008.

[5] Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '09, pages 266–278, 2009.

[6] Kevin Costello, Asaf Shapira, and Prasad Tetali. Randomizing two derandomized greedy algorithms. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '11, 2011.

[7] Kevin Costello, Prasad Tetali, and Pushkar Tripathi. Stochastic matching with commitment. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '12, 2012.

[8] Ran Duan and Seth Pettie. Approximating maximum weight matching in near-linear time. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, pages 673–682, Washington, DC, USA, 2010. IEEE Computer Society.

[9] Martin Dyer and Alan Frieze. Randomized greedy matching. *Random Structures and Algorithms*, 2(1):29–45, 1991.

[10] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[11] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA*, pages 982–991, 2008.

[12] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 731–740, New York, NY, USA, 2002. ACM.

[13] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[14] Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching in the unknown distributional model. In *STOC*, pages 106–117, 2011.

[15] David R. Karger. Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 21–30, 1993.

[16] R.M. Karp, U.V. Vazirani, and V.V. Vazirani. An optimal algorithm for online bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990.

[17] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Allocating online advertisement space with unreliable estimates. In *Proceedings of the 8th ACM conference on Electronic commerce*, EC '07, pages 288–294, New York, NY, USA, 2007. ACM.

[18] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: An approach based on strongly factor-revealing lps. In *STOC*, pages 117–126, 2011.

[19] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Improved approximation algorithms for metric facility location problems. In *In Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 229–242, 2002.

[20] Aranyak Mehta and Vahab Mirrokni. Online ad serving : Theory and practice. Tutorial, 2011.

[21] Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. In *FOCS*, 2005.

[22] Silvio Micali and Vijay V. Vazirani. An o(sqrtve) algoithm for finding maximum matching in general graphs. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 17 –27, oct. 1980.

[23] Marco Molinaro and R. Ravi. The query-commit problem. Manuscript, http://arxiv.org/abs/1110.0990, 2011.

[24] Seth Pettie and Peter Sanders. A simpler linear time $2/3 - \epsilon$ approximation to maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004.

[25] L.F. Ross, D.T. Rubin, M. Siegler, M.A. Josephson, J.R. Thistlethwaite, and E.S.Woodle. Ethics of a paired-kidney-exchange program. *The New England Journal of Medicine*, 336:1752–1755, 1997.

[26] L.F. Ross, D.T. Rubin, M. Siegler, M.A. Josephson, J.R. Thistlethwaite, and E.S. Woodle. The case for a living emotionally related international kidney donor exchange registry. *Transplantation Proceedings*, 18:5–9, 1986.

[27] Tayfun Sonmez Roth, Alvin E. and M. Utku Unver. A kidney exchange clearinghouse in new england. *American Economic Review, Papers and Proceedings*, 95:376–380, May 2005.

[28] Tayfun Sonmez Roth, Alvin E. and M. Utku Unver. Pairwise kidney exchange. *Journal of Economic Theory*, 125:151–188, December 2005.

[29] Tayfun Sonmez Roth, Alvin E. and M. Utku Unver. Kidney paired donation with compatible pairs(letter to the editor). *American Journal of Transplantation*, 7, December 2007.

[30] Doratha E. Drake Vinkemeier and Stefan Hougardy. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Trans. Algorithms*, 1:107–122, July 2005.

[31] A. C. Yao. Probabilistic computations: towards a unified measure of complexity. *FOCS*, pages 222–227, 1977.

APPENDIX

Proof of Claim 2

Proof: Since u is unmatched in $Shuffle(\rho)$ and it is adjacent to w(by our assumption), w must be matched in $Shuffle(\rho)$. Let v be the vertex that w gets matched to. Since we are in Case 2 of Lemma 7, $v \neq w^*$. Next let us look at $Shuffle(\rho')$; v must available when w gets matched to u. Thus when w gets matched in $Shuffle(\rho)$ it has at least two options i.e. u and v.

Finally lets consider $\rho'' = \rho'[w^* \rightsquigarrow s]$. By the stability property (Claim 1) the matchings produced by $Shuffle(\rho')$ and $Shuffle(\rho'')$ at the time when w is being considered differ by an augmenting path. Thus the number of options available to w in $Shuffle(\rho')$ and $Shuffle(\rho'')$ can differ by at most 1. Recall that both u and $v \neq w^*$ were available to w in $Shuffle(\rho')$. So at least one of them should still be available in $Shuffle(\rho'')$. Therefore w will get matched in $Shuffle(\rho'')$. ∎

Proof of Claim 3

Proof: Let us consider two cases.
Case 1: $\rho'^{-1}(w) > t$. Since u gets matched to w in $Shuffle(\rho')$, w should be unmatched before time t. Note that the execution of $Shuffle(\rho'[w^* \rightsquigarrow s])$ and $Shuffle(\rho')$ would be identical until time s. Thus when we consider w^* in $Shuffle(\rho'[w^* \rightsquigarrow s])$, w is still unmatched. Therefore w^* will surely get matched by $Shuffle(\rho'[w^* \rightsquigarrow s])$.

Case 2: $\rho'^{-1}(w) \leq t$. If possible let w^* not get matched in $Shuffle(\rho'[w^* \rightsquigarrow s])$. Let $r = \rho'^{-1}(w)$. In this case $Shuffle(\rho')$ and $Shuffle(\rho'[w^* \rightsquigarrow s])$ are identical until time r. Therefore both u and w^* must be unmatched when we consider w in $Shuffle(\rho'[w^* \rightsquigarrow s])$. Since $s \leq t$, w would choose w^* instead of u which is a contradiction. ∎

Proof of Lemma 16

Proof: Consider the execution of $\mathcal{O}$ on any instance $I = (C \cup P, E)$. We can simulate the execution of $\mathcal{O}$ on I by a revealing algorithm $\mathcal{A}$. The only possible difficulty could be if $\mathcal{O}$ scans an edge incident on a vertex that is already marked to be inactive by $\mathcal{A}$. This step is not allowed in $\mathcal{A}$, and cannot be simulated. In this case the current vertex in our simulation immediately gives up and we proceed to the next phase. One can check that under this simulation $\mathcal{A}$ produces a matching that is at least as large as the one returned by $\mathcal{O}$. ∎