Dynamic Maximal Matching in Clique Networks

Minming Li
Department of Computer Science, City University of Hong Kong, Hong Kong

Peter Robinson
School of Computer & Cyber Sciences, Augusta University, GA, USA

Xianbin Zhu
Department of Computer Science, City University of Hong Kong, Hong Kong

Abstract

We consider the problem of computing a maximal matching with a distributed algorithm in the presence of batch-dynamic changes to the graph topology. We assume that a graph of n nodes is vertex-partitioned among k players that communicate via message passing. Our goal is to provide an efficient algorithm that quickly updates the matching even if an adversary determines batches of ℓ edge insertions or deletions. We first show a lower bound of $\Omega\left(\frac{\ell \log k}{k^2 \log n}\right)$ rounds for recomputing a matching assuming an oblivious adversary who is unaware of the initial (random) vertex partition as well as the current state of the players, and a stronger lower bound of $\Omega(\frac{\ell}{k \log n})$ rounds against an adaptive adversary, who may choose any balanced (but not necessarily random) vertex partition initially and who knows the current state of the players. We also present a randomized algorithm that has an initialization time of $O\left(\frac{n}{k} \log n\right)$ rounds, while achieving an update time that that is independent of n: In more detail, the update time is $O\left(\left\lceil \frac{\ell}{k} \right\rceil \log k\right)$ against an oblivious adversary, who must fix all updates in advance. If we consider the stronger adaptive adversary, the update time becomes $O\left(\left\lceil \frac{\ell}{\sqrt{k}} \right\rceil \log k\right)$ rounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases distributed graph algorithm, dynamic network, maximal matching, randomized algorithm, lower bound

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.73

Funding The work described in this paper was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 11213620].

1 Introduction

Designing efficient algorithms for large graphs is crucial for many applications, ranging from transportation networks to protein-interaction networks in biology. Real-world graph data sets are inherently dynamic in the sense that the nodes and edges between them may change over time. While much of the previous literature on dynamic updates in the centralized setting consider only a single update at a time, in a distributed network, each node runs an instance of a distributed algorithm and thus, if multiple updates are happening (almost) simultaneously perhaps affecting different parts of the network, it is desirable to process all of these updates in the same batch to reduce the overall time complexity. In this work, we study the fundamental problem of computing a maximal matching with a distributed algorithm under batch-dynamic edge updates. We point out that, recently, batch-dynamic parallel algorithms have received significant attention, e.g., see [1, 2, 3, 13]. A *matching* M in a graph G is a set of edges without common vertices, and we say that M is a *maximal matching* if no matching in G is a proper superset of M.

The k-Clique Message Passing Model. We assume a communication network that is a clique of k players $P_1, \ldots, P_k$, each of whom is executing an instance of a distributed algorithm. The input is an n-vertex graph, where each vertex is labeled with a unique integer

ID from $[n]$. For technical reasons that will become clear in Section 3, we assume that $k = \Omega(\log^4 n)$, and we are mostly interested in the case where $k \ll n$, which captures the realistic setting that the amount of data exceeds the number of available processing units by far.

The players communicate via message passing over the bidirectional links of the clique network, and we assume that the computation proceeds in *synchronous rounds*: In every round, a player performs some local computation, which may include private random coin flips, and then sends a message of at most $O(\log n)$ bits over each one of its $k-1$ incident communication links. Note that all messages are guaranteed to be received by the end of the same round. Since each link carries at most $O(\log n)$ bits per round, each player can receive at most $O(k \log n)$ bits per round in total from the other players. As we consider distributed algorithms, each player initially only knows part of the input. Here we consider **vertex-partitioning**, which means that each vertex u is assigned to some player P. For each vertex v assigned to P, we say that P *hosts* vertex v. We use the notation $P(v)$ to denote the player hosting v, and use $V(P)$ to denote the set of vertices hosted by P. Note that vertices cannot migrate between players.

Adversary and Input Assignment. The input of a player P consists of the IDs of its hosted vertices $V(P)$ and, for each $v \in V(P)$, player P knows the IDs of v's neighbors and the players to which they were assigned. In this work, we consider two adversarial models: The *adaptive adversary* may fix any *balanced vertex partitioning*, which requires each player to hold at least $\Omega(\frac{n}{k})$ and at most $O(\frac{n}{k} \log n)$ vertices in total. On the other hand, when considering the *oblivious adversary*, we assume *random vertex partitioning*, which means that each vertex is uniformly at random assigned to any player. We point out that randomly partitioning the vertices is a common assumption in real-world graph processing systems, e.g., see [12]. In expectation, each player P obtains a set of $\Theta(n/k)$ vertices, and thus random vertex partitioning yields a balanced partitioning with high probability.

Dynamic Changes, Initialization, and Output. We model dynamical changes in this setting by considering a sequence of updates to the edges of the graph. Each update consists of a *batch* of at most ℓ edge deletions and insertions, chosen by the adversary as follows:

- The adaptive adversary may observe the local memory of the players, which includes the currently computed matching, before choosing the next batch of ℓ edge updates.
- The oblivious adversary must choose the entire sequence of batch updates in advance.

Since we are assuming a distributed algorithm, each player only needs to output the part of the solution relevant to its hosted vertices. In more detail, each player P needs to output every edge $\{u, v\} \in M$, for which it hosts an endpoint, e.g., $u \in V(P)$.

We allow the algorithm to perform some initial computation right after the vertex partitioning and before the very first batch of updates arrives. The number of rounds required for this part defines the *initialization time* of the algorithm. Then, the first batch of updates takes place instantaneously, and every player that hosts a vertex v incident to an added or deleted edge learns about v's new neighborhood. These changes yield a modified graph G'. The algorithm must react by updating M to yield a maximal matching for G'; no further changes to the graph topologies occur until the algorithm has completed its update. Subsequently, the next batch of updates arrives, and so forth.

Update Time. As we are considering randomized algorithms, we give probabilistic guarantees on the (worst case) time complexity measures. When saying that an event $\mathcal{E}$ holds *with high probability (in N)*, for some parameter N, this means that $\mathcal{E}$ occurs with probability at

least $1 - 1/N^c$, where c is a positive constant. If $N = n$, we simply omit N and say that the $\mathcal{E}$ happens with high probability. In particular, we say that an *algorithm has an update time of T with high probability in N*, if the algorithm outputs a maximal matching following a batch of ℓ updates in at most T rounds w.h.p. in N, assuming that the local states of the players correspond to a maximal matching on the graph prior to these updates. Note that for the oblivious adversary, the probability is taken over the random coin flips of the players, as well as the random vertex partitioning. Since a random partition is very likely balanced, it will be sufficient if this bound holds for all roughly balanced partitions.

Relationship to k-Machine Model and Congested Clique. When assuming the oblivious adversary, our model corresponds to the *k-machine model* [22, 17], which is motivated by vertex-centric graph processing frameworks such as Google Pregel [24], Apache Giraph [12], and GraphX [16]. Thus, our results for the oblivious case directly extend to this setting.

For $k = n$ players, our model is equivalent to the congested clique [23] if each player obtains exactly one vertex. Thus our approach also yields a communication-efficient dynamic algorithm for the latter model. We elaborate this point in more detail in Section 4.

Local Memory. Analogously to the congested clique and the k-machine model, we do not impose a restriction on the local memory of the players. Note that this stands in contrast to the popular Massively Parallel Computation (MPC) model of [21], where the memory of each machine (i.e., player) is limited, which in turn limits the amount of information any machine can receive in a single round.

We point out that our algorithm is nevertheless space-efficient, in the sense that each player P_i uses at most $O(\max\{n, |G[V(P_i)]|\} \cdot \log n)$ bits of local memory, where $|G[V(P_i)]|$ is the size of the subgraph induced by P_i's hosted vertices and their incident edges.

1.1 Our Contributions

We present the first bounds for maximal matching in the k-clique message passing model under batch-dynamic updates, where each batch consists of at most ℓ edge additions or deletions. We start by determining lower bounds on the necessary update time of any algorithm:

▶ **Theorem 1.** *Consider any randomized algorithm for maximal matching in the k-clique message passing model that initially constructs a maximal matching (w.h.p.), and is guaranteed to recompute a maximal matching with an update time of T (w.h.p.), assuming ℓ edge-updates per batch, for any $\ell \leqslant n/2k$ and $k \leqslant \sqrt{n}/\log n$. Then, the following hold:*

1. $T = \Omega\left(\frac{\ell \log k}{k^2 \log n}\right)$ *rounds, against an oblivious adversary;*
2. $T = \Omega\left(\frac{\ell}{k \log n}\right)$ *rounds, against an adaptive adversary.*

These results hold for any number of initialization rounds, and even if the players have access to shared randomness.

We point out that there is a naïve way to obtain an update time of $O(\lceil \ell/k \rceil)$ rounds that applies to any problem in this setting, if we allow a prohibitively large initialization time of $O(m/k) = O(n^2/k)$ rounds, for an input graph with m edges. To see why this is the case, observe that every player can learn the entire input graph within $O(m/k)$ rounds, considering that an edge fits into a message of $O(\log n)$ bits and a player can receive $O(k)$

messages per round. Then, upon any batch of ℓ updates, each player simply sends all its updates to every other player, which, by using a simple information dissemination technique (see Lemma 8), takes $O(\ell/k)$ rounds. Since every player knew the entire graph from the initialization phase and has learned about all edge-changes, all players also know the updated graph and thus can locally compute the new solution without further communication. Apart from the slow initialization time, we emphasize that this naïve approach requires each player to have $\Omega(m)$ bits of local memory, which is unrealistic when considering large data sets.

We present a randomized algorithm that has a significantly faster initialization time of $O\big(\frac{n}{k}\log n\big)$ rounds, nearly matching the bounds of Theorem 1 up to a factor of $(k \cdot \log n)$ in the case of an oblivious adversary. For an adaptive adversary, we are able to further narrow the gap between lower and upper bound to just a $(\sqrt{k} \cdot \log k \cdot \log n)$-factor.

▶ **Theorem 2.** *Suppose that the adversary may add or delete up to ℓ edges per batch. There exists a randomized dynamic algorithm for maximal matching that has an initialization time of $O\big(\frac{n}{k}\log n\big)$ rounds with high probability (in n), and each player P_i uses at most $O(\max\{n, |G[V(P_i)]|\} \cdot \log n)$ bits of local memory. The update time T is bounded as follows:*

1. *Oblivious adversary: $T = O\big(\lceil\frac{\ell}{k}\rceil \log k\big)$ rounds w.h.p. (in k), and $T = O\big(\lceil\frac{\ell}{k}\rceil \log n\big)$ rounds w.h.p. (in n).*
2. *Adaptive adversary: $T = O\Big(\lceil\frac{\ell}{\sqrt{k}}\rceil \log k\Big)$ rounds w.h.p. (in n).*

In Section 4, we show that our techniques lead to communication-efficient algorithms in the congested clique, in the sense that the message complexity is proportional to the number of updates.

1.2 Related Work

We start by describing prior work that considers dynamic updates in the k-machine model and the MPC model [21], as these are most closely related to our setting. While the k-machine model has been considered for a wide variety of problems such as PageRank approximation [26] and graph clustering [7], to the best of our knowledge, the only work studying dynamically-changing graphs in the k-machine model is the result of Gilbert and Lu [15] on minimum spanning trees (MSTs). The primary technique used in their work is applying Euler Tours, which facilitates updating an MST in response to edge updates. The (static) maximal matching problem has not been studied in the k-machine model. While the approach of [5] provides a way to translate existing PRAM maximal matching algorithms to the k-machine model, their work crucially relies on a balanced *edge partitioning*, and thus is not applicable to the k-clique message passing model, where we assume *vertex partitioning*.

The paper [20] initiated dynamic problems in the MPC model and presented dynamic MPC algorithms for connectivity, minimum spanning tree and several matching problems. They also explored the relationship between classical dynamic algorithms and dynamic algorithms in the MPC model. Following the dynamic connectivity algorithms in the MPC model under a batch of updates [13], Nowicki and Onak [25] further studied dynamic MPC algorithms for minimum spanning forest, 2-edge connected components, and maximal matching with batch updates.

One may find that k-machine model and the MPC model share some similarities. We should notice that the major difference between the k-machine model and the MPC model is that, in the k-machine model, the total bandwidth is $O(k^2 \log n)$ while the MPC model

has a total bandwidth $\tilde{O}(m)$ where m is the number of edges of the input graph. In the MPC model, each machine can load all its input to other machines in one round, but in the k-machine model, this takes at least $\Omega(S/k)$ rounds where S is an upper bound on the number of edges incident to the vertices hosted by each machine. In [25], their idea to deal with a batch of updates happening on a maximal matching is to reduce it to a maximal matching problem on a graph with vertex cover size at most $O(k)$, which can be solved by static algorithms of maximal matching in [8]. Note that the total bandwidth in the k-clique message passing model is $O(k^2 \log n)$ bits per round, and thus it is unclear how to efficiently run dynamic algorithms designed for the MPC model in our setting.

Since the pioneering work of [10], several advancements have been made in the field of dynamic distributed models, e.g., [6, 9, 11]. Notably, [9] gave dynamic algorithms for maximal matching in the CONGEST model using $O(1)$ amortized time complexity but the worst case time complexity is $O(n)$. Recently, [14] conducted a study on dynamic problems in the CONGEST model and Congested Clique model. They demonstrated that for any problem, there exists a batch-dynamic congested clique algorithm using $O(\lceil \alpha/n \rceil)$ rounds and $O(m \log n)$ bits of auxiliary state when there are α edge label changes in a batch. Also [4] gave distributed dynamic algorithms for some classical symmetry breaking problems and their techniques are majorly based on dynamic algorithms in the centralized settings.

2 A Lower Bound for Batch-Dynamic Maximal Matching

In this section, we prove Theorem 1. We first present the proof for the oblivious adversary, which is technically more involved, due to the fact that the adversary neither controls the input partition nor has any knowledge of the current maximal matching when choosing the updates.

For both cases, oblivious and adaptive adversaries, we consider the following lower bound graph G, formed by $q = n/3$ disjoint line segments $L_1, \ldots, L_q$, and each L_i consists of three nodes $x_{i,1}$, $x_{i,2}$, and x_{i-3} that are connected by a path of length 2; for simplicity, we assume that $n/3$ and n/k are integers. Figure 1a gives an example of this construction. If a vertex u is part of the line segment L_j, we say that $j \in [q]$ is the *index of* u.

2.1 Oblivious Adversary

Initially, the algorithm computes some maximal matching on G. Thus, there exists a set S of $q = n/3$ unmatched vertices in total, and S contains exactly one unmatched vertex per line segment. For a player P_i, we define $U_i \subseteq [q]$ to be the set of corresponding line segment indices of the vertices in S that are hosted by P_i. Let $U'_i \subseteq U_i$ denote the indices of the unmatched vertices at P_i such that, for every $j \in U'_i$, player P_i hosts only the (single) unmatched vertex of line segment L_j, and neither of the other two matched vertices in L_j. We define the event Lrg ("large") as

$$\mathsf{Lrg} = \bigwedge_{i=1}^{k} \left(|U'_i| \geqslant |U_i| - \frac{12n}{k^2} \right), \tag{1}$$

which captures the case when every set U'_i is of sufficiently large size. We now show that Lrg is very likely to occur.[1]

[1] Omitted proofs will appear in the full version of the paper.

▶ **Lemma 3.** *Event* Lrg *occurs with probability at least* $1 - n^{-\Omega(1)}$.

Clearly, there must exist a player P_1 that hosts a set of at least $q/k = n/3k$ unmatched vertices after computing the initial matching, i.e., $|U_1| \geqslant n/3k$, and this invariant remains true even when conditioning on event Lrg. For the remainder of the proof of Theorem 1, we focus on the amount of information learned by P_1 during the course of the update procedure. Define $\mathsf{Lrg}_{U_1'}$ to be the event that U_1' is "large", formally,

$$|U_1'| \geqslant \frac{n}{3k} - \frac{12n}{k^2} \geqslant \frac{n}{6k}. \tag{2}$$

Note that the second inequality holds for sufficiently large n, since $k = \omega(1)$ and $k = o(n)$ by assumption. Clearly, $\mathsf{Lrg}_{U_1'}$ is implied by event Lrg, and hence we immediately obtain the following from Lemma 3:

▶ **Lemma 4.** *Event* $\mathsf{Lrg}_{U_1'}$ *(see Ineq. (2)) occurs with probability at least* $1 - n^{-\Omega(1)}$.

Adversarial Strategy. The adversary samples a set of indices I, by independently including each $i \in [q]$ with probability $\frac{\ell}{n}$, i.e., $|I| = \frac{\ell q}{n} = \ell/3$ in expectation. If the resulting set I has a size greater than ℓ, the adversary simply restarts the sampling process until $|I| \leqslant \ell$. Then, for each $i \in I$, the adversary removes one of the two edges of segment L_i, chosen independently and uniformly at random.

Let $A \subseteq I$ be the set of *affected indices*, which contains every index $i \in I$, for which the adversary deletes the matched edge e of L_i, i.e., e is *not* incident to the unmatched vertex in L_i. We define $A_1 = A \cap U_1'$, i.e., A_1 is the subset of indices i, such that:
1. the only vertex of L_i hosted by player P_1 is the unmatched vertex $v \in L_i$, and
2. the adversary has deleted the matched edge of L_i, which is not incident to v.
Let Bal_{A_1} be the event that $|A_1|$ is "balanced" with respect to U_1', which is true if

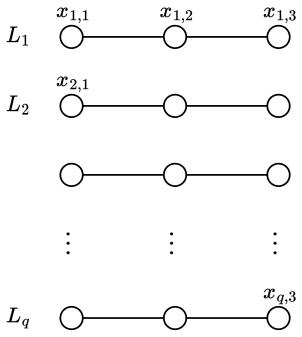
$$|A_1| \in \left[\frac{\ell}{4n}|U_1'|, \frac{4\ell}{n}|U_1'|\right]. \tag{3}$$

▶ **Lemma 5.** *For a given* U_1' *and conditioned on* $\mathsf{Lrg}_{U_1'}$*, event* Bal_{A_1} *occurs, i.e., Eq. 3 holds, with probability at least* $1 - n^{-\Omega(1)}$.

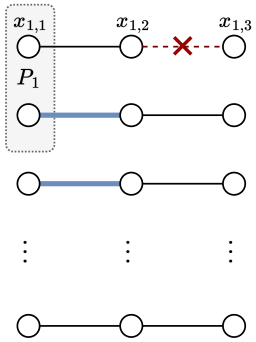
Notation. We slightly abuse notation and write "P_i" to refer to the random variable that represents the local state of player P_i after the adversary removes the edges but before any update-computation of the algorithm has taken place; we assume that the local state also includes the public random bits. We frequently compute the conditional entropy of a random variable X with respect to a random variable Y and some event $Z = z$. To improve readability, we use the notation $\mathbf{H}[X \mid Y, z]$ to mean $\mathbf{H}[X \mid Y, Z = z]$, i.e., the conditional entropy of X with respect to random variable Y conditioned on event $Z = z$.

▶ **Lemma 6.** $\mathbf{H}\Big[A_1 \;\Big|\; |A_1| = a, P_1 = p_1, U_1' = u_1', \mathsf{Bal}_{A_1}, \mathsf{Lrg}_{U_1'}\Big] = \Omega\left(\frac{\ell}{k} \cdot \log_2\left(\frac{n}{\ell}\right)\right)$

Proof. Define $p(a_1) = \mathbf{Pr}\Big[A_1 = a_1 \;\Big|\; a, p_1, u_1', \mathsf{Bal}_{A_1}, \mathsf{Lrg}_{U_1'}\Big]$. To obtain a bound on $p(a_1)$ that holds for any given a_1, we need to estimate the number of possible choices for A_1 under the given conditioning. Since $A_1 \subseteq U_1'$ and $|A_1| = a$, there are $\binom{|u_1'|}{a}$ possible ways for choosing



(a) The input graph consists of $q = n/3$ line segments, each of length 2.



(b) The thick blue edges show the maximal matching after removing the (matched) edge $\{x_{1,2}, x_{1,3}\}$. Player P_1 hosts $x_{1,1}$ and $x_{2,1}$.

Figure 1 The lower bound graph construction. After the adversary deletes the matched edge $\{x_{1,2}, x_{1,3}\}$ in the graph shown in Figure 1b, player P_1, who is unaware of this update, must learn about it from another player to know that it needs to output $\{x_{1,1}, x_{1,2}\}$ as part of the matching.

A_1. Recall that the adversary chooses the indices in A (and hence also A_1) independently from the vertex partitioning, which tells us that A_1 has uniform probability over the $\binom{|u_1'|}{a}$ possible choices. We have

$$\begin{aligned}
\log_2 \frac{1}{p(a_1)} &\geqslant \log_2 \binom{|u_1'|}{a} \\
(\text{since } \binom{n}{k} \geqslant \left(\frac{n}{k}\right)^k) \quad &\geqslant a \cdot \log_2\left(\frac{|u_1'|}{a}\right) \\
(\text{since } \mathsf{Bal}_{A_1} \text{ implies } a \in [(\ell/4n)|u_1'|, (\ell/n)|u_1|']) \quad &\geqslant \frac{\ell}{4n}|u_1'| \cdot \log_2\left(\frac{n}{\ell}\right) \\
(\text{since } \mathsf{Lrg}_{U_1'} \text{ implies } |u_1'| \geqslant 6n/k) \quad &= \Omega\left(\frac{\ell}{k} \cdot \log_2\left(\frac{n}{\ell}\right)\right). \qquad (4)
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathbf{H}\left[A_1 \;\middle|\; a, p_1, u_1', \mathsf{Bal}_{A_1}, \mathsf{Lrg}_{U_1'}\right] &= \sum_{a_1} p(a_1) \cdot \log_2 \frac{1}{p(a_1)} \\
(\text{by } (4)) \quad &\geqslant \Omega\left(\frac{\ell}{k}\log\left(\frac{n}{\ell}\right)\right) \sum_{a_1} p(a_1) = \Omega\left(\frac{\ell}{k}\log\left(\frac{n}{\ell}\right)\right).
\end{aligned}$$

◀

We now combine Lemmas 4, 5, and 6 to bound the initial uncertainty about the set A_1 from the point of view of player P_1. For an event $\mathcal{E}$, we use $\mathbf{1}_{\mathcal{E}}$ to denote the indicator random variable that is 1 if and only if $\mathcal{E}$ occurs. Since conditioning on a random variable cannot increase the entropy, we have

$$
\begin{aligned}
\mathbf{H}[A_1 \mid P_1] &\geqslant \mathbf{H}\Big[A_1 \;\Big|\; P_1, U_1', \mathbf{1}_{\mathsf{Lrg}_{U_1'}}\Big] \\
&\geqslant \mathbf{Pr}\Big[\mathsf{Lrg}_{U_1'}\Big] \cdot \mathbf{H}\Big[A_1 \;\Big|\; P_1, U_1', \mathsf{Lrg}_{U_1'}\Big] \\
\text{(by Lem. 4)} \quad &\geqslant \tfrac{1}{2} \cdot \mathbf{H}\Big[A_1 \;\Big|\; P_1, U_1', \mathsf{Lrg}_{U_1'}\Big] \\
&= \tfrac{1}{2} \cdot \sum_{u_1} \mathbf{Pr}\Big[U_1 = u_1' \;\Big|\; \mathsf{Lrg}_{U_1'}\Big] \cdot \mathbf{H}\Big[A_1 \;\Big|\; P_1, U_1' = u_1', \mathsf{Lrg}_{U_1'}\Big] \\
&\geqslant \tfrac{1}{2} \sum_{u_1'} \mathbf{Pr}\Big[u_1' \;\Big|\; \mathsf{Lrg}_{U_1'}\Big] \cdot \mathbf{H}\Big[A_1 \;\Big|\; P_1, \mathbf{1}_{\mathsf{Bal}_{A_1}}, u_1', \mathsf{Lrg}_{U_1'}\Big] \\
&\geqslant \tfrac{1}{2} \sum_{u_1'} \Big(\mathbf{Pr}\Big[u_1' \;\Big|\; \mathsf{Lrg}_{U_1'}\Big] \cdot \mathbf{Pr}\Big[\mathsf{Bal}_{A_1} \;\Big|\; u_1', \mathsf{Lrg}_{U_1'}\Big] \\
&\qquad\qquad \cdot \mathbf{H}\Big[A_1 \;\Big|\; P_1, \mathsf{Bal}_{A_1}, u_1', \mathsf{Lrg}_{U_1'}\Big]\Big) \\
\text{(by Lem. 5)} \quad &\geqslant \tfrac{1}{2} \cdot \tfrac{1}{2} \sum_{u_1'} \mathbf{Pr}\Big[u_1' \;\Big|\; \mathsf{Lrg}_{U_1'}\Big] \cdot \mathbf{H}\Big[A_1 \;\Big|\; P_1, \mathsf{Bal}_{A_1}, u_1', \mathsf{Lrg}_{U_1'}\Big] \\
&\geqslant \tfrac{1}{4} \sum_{u_1'} \mathbf{Pr}\Big[u_1' \;\Big|\; \mathsf{Lrg}_{U_1'}\Big] \cdot \mathbf{H}\Big[A_1 \;\Big|\; P_1, |A_1|, \mathsf{Bal}_{A_1}, u_1', \mathsf{Lrg}_{U_1'}\Big] \\
&= \tfrac{1}{4} \sum_{u_1'} \mathbf{Pr}\Big[u_1' \;\Big|\; \mathsf{Lrg}_{U_1'}\Big] \cdot \sum_{a,p_1} \Big(\mathbf{Pr}\Big[|A_1| = a, P_1 = p_1 \;\Big|\; \mathsf{Bal}_{A_1}, u_1', \mathsf{Lrg}_{U_1'}\Big] \\
&\qquad\qquad \cdot \mathbf{H}\Big[A_1 \;\Big|\; a, p_1, u_1', \mathsf{Bal}_{A_1}, \mathsf{Lrg}_{U_1'}\Big]\Big) \\
\text{(by Lem. 6)} \quad &= \Omega\Big(\frac{\ell}{k} \cdot \log_2\Big(\frac{n}{\ell}\Big)\Big) \sum_{u_1'} \mathbf{Pr}\Big[u_1' \;\Big|\; \mathsf{Lrg}_{U_1'}\Big] \cdot \sum_{a,p_1} \mathbf{Pr}\Big[a, p_1 \;\Big|\; \mathsf{Bal}_{A_1}, u_1', \mathsf{Lrg}_{U_1'}\Big] \\
&= \Omega\Big(\frac{\ell}{k} \cdot \log_2\Big(\frac{n}{\ell}\Big)\Big), \qquad (5)
\end{aligned}
$$

where the final step follows because both of the sums over the probabilities evaluate to 1.

Let random variable Out_1 be the edges that are output by P as part of the maximal matching after the update is complete, and let $\mathbf{1}_{\text{Corr}}$ be the indicator random variable that the algorithm succeeds, which equals 1 with high probability according to the premise of the theorem.

▶ **Lemma 7.** $\mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1] = O(1)$.

Proof. By the chain rule of conditional entropy, we have

$$
\begin{aligned}
\mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1] &\leqslant \mathbf{H}[A_1, \mathbf{1}_{\text{Corr}} \mid \mathsf{Out}_1, P_1] \\
&= \mathbf{H}[A_1 \mid \mathbf{1}_{\text{Corr}}, \mathsf{Out}_1, P_1] + \mathbf{H}[\mathbf{1}_{\text{Corr}} \mid \mathsf{Out}_1, P_1] \\
\text{(since } \mathbf{H}[\mathbf{1}_{\text{Corr}} \mid \mathsf{Out}_1, P_1] \leqslant 1) \quad &\leqslant \mathbf{H}[A_1 \mid \mathbf{1}_{\text{Corr}}, \mathsf{Out}_1, P_1] + 1 \\
&= \mathbf{Pr}[\mathbf{1}_{\text{Corr}} = 1]\, \mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1, \mathbf{1}_{\text{Corr}} = 1] \\
&\qquad + \mathbf{Pr}[\mathbf{1}_{\text{Corr}} = 0]\, \mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1, \mathbf{1}_{\text{Corr}} = 0] + 1. \\
&\leqslant \mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1, \mathbf{1}_{\text{Corr}} = 1] \\
&\qquad + \mathbf{Pr}[\mathbf{1}_{\text{Corr}} = 0]\, \mathbf{H}[A_1 \mid \mathbf{1}_{\text{Corr}} = 0] + 1. \\
\text{(since } \mathbf{Pr}[\mathbf{1}_{\text{Corr}} = 0] \leqslant 1/n) \quad &\leqslant \mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1, \mathbf{1}_{\text{Corr}} = 1] + \frac{1}{n}\, \mathbf{H}[A_1 \mid \mathbf{1}_{\text{Corr}} = 0] + 1. \\
&\leqslant \mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1, \mathbf{1}_{\text{Corr}} = 1] + O(1), \qquad (6)
\end{aligned}
$$

where the final step follows since $\mathbf{H}[A_1 \mid \mathbf{1}_{\text{Corr}} = 0] = O(n)$.

Conditioned on the event $\mathbf{1}_{\text{Corr}} = 1$, player P_1 must output the edge incident to every previously unmatched vertex $v \in L_j$, for all $j \in A_1$; see Figure 1b on page 7 for an example. This means that P_1 must have learned the entire set of indices in A_1 by the time the algorithm completes its update, which implies that

$$\mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1, \mathbf{1}_{\text{Corr}} = 1] = 0, \tag{7}$$

and completes the proof of Lemma 7. ◀

Let Π be the transcript of messages received by player P_1 during the update. Note that Out_1 is a function of Π and the local state of P_1, which includes the public randomness. By the data-processing inequality, it follows that

$$\begin{aligned} \mathbf{I}[\,\Pi : A_1 \mid P_1] &\geqslant \mathbf{I}[\,\mathsf{Out}_1 : A_1 \mid P_1] \\ &= \mathbf{H}[A_1 \mid P_1] - \mathbf{H}[A_1 \mid \mathsf{Out}_1, P_1] \\ \text{(by Ineq. (5) and Lem. 7)} \quad &\geqslant \Omega\left(\frac{\ell}{k}\log\left(\frac{n}{\ell}\right)\right) - O(1) \\ &= \Omega\left(\frac{\ell}{k}\log k\right), \end{aligned} \tag{8}$$

where the final step follows since $\ell \leqslant n/2k$, according to the premise of the theorem. In each round, player P_1 can receive at most $c \log n$ bits over every one of its $k-1$ communication links, for some constant $c \geqslant 1$. Thus there are

$$2^{c \log n} + 1 \leqslant 2^{c \log n + 1}$$

possible messages (including the empty message) that P_1 may receive from a distinct player P_j in a single round. Let $\mathcal{M}_r$ be the concatenation of the messages that P_1 receives in a given round r from the $k-1$ other players. By the above derivation, there are at most $2^{c(k-1)\log n + 1}$ possible values for $\mathcal{M}_r$. Let T denote the worst case update time of player P_1, and note that $\Pi = \mathcal{M}_1 \ldots \mathcal{M}_T$. It follows that there are at most $2^{c(k-1)T\log n + 1}$ possible values for Π. The entropy $\mathbf{H}[\Pi]$ is maximized when Π is uniformly distributed, i.e.,

$$\mathbf{H}[\Pi] \leqslant \log_2 2^{c(k-1)T\log n + 1},$$

and (8) implies that

$$c(k-1)T\log n + 1 \geqslant \mathbf{H}[\Pi] \geqslant \mathbf{I}[\,\Pi : A_1 \mid P_1] = \Omega\left(\frac{\ell}{k}\log k\right).$$

This shows that $T = \Omega\left(\frac{\ell \log k}{k^2 \log n}\right)$, as required.

2.2 Extending the Proof to the Adaptive Adversary

We now consider the adaptive adversary, who not only determines the initial (balanced) vertex partitioning, but is also aware of the current state of the nodes, which, in particular includes the computed maximal matching. As before, we focus on the graph G that consists of a collection of q disjoint line segments $L_1, \ldots, L_q$, where each $L_i = (x_{i,1}, x_{i,2}, x_{i,3})$ forms a path of length 2 on vertices $x_{i,1}$, $x_{i,2}$, and $x_{i,3}$; see Figure 1a on page 7. To simplify the presentation, we assume that $\frac{q}{n/k} = \frac{k}{3}$ is an integer. Conceptually, we view the graph as an $(q \times 3)$-size checkerboard B where $x_{i,j}$ is located at coordinate (i, j). It is possible to *tile*

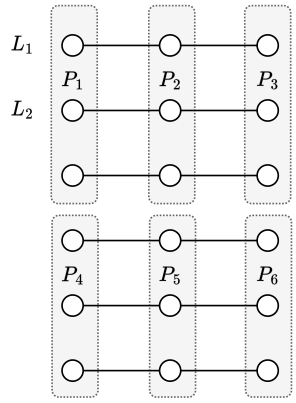


Figure 2 The initial vertex partition ("tiling") of the input graph chosen by the adaptive adversary. Each shaded area corresponds to the vertices hosted by one player.

B, i.e., cover all spaces without any overlaps by using k dominos $D_1, \ldots, D_k$, each of size $(n/k) \times 1$, see, e.g., [27]. The set of coordinates of B covered by D_i represents the vertices assigned to player P_i, which results in a balanced vertex partition, where each player obtains n/k vertices. Figure 2 gives an example of such a tiling.

This ensures the following property:

▶ **Fact 1.** *Every player hosts at most* 1 *vertex per line segment.*

Observe that any maximal matching must leave exactly one vertex per line segment unmatched. Similarly to the oblivious case, there must exist a player P who hosts a set S_P of at least $n/3k$ unmatched vertices. A crucial difference, however, is that the adaptive adversary knows P and S_P when selecting the edge updates. We define I_P to be the set of indices of the line segments that have a vertex in S_P, and note that Fact 1 implies that $|I_P| \geqslant n/3k$.

Then, the adversary samples a set $A_P \subseteq I_P$ of size ℓ uniformly at random, and deletes the matched edge $e_i \in L_i$ from the graph, for each line segment L_i with an index $i \in A_P$. As player P hosts no vertex incident to e_i, it is unaware of these deletions unless it communicates with the other players. Observe that there are

$$\binom{|I_P|}{\ell} \geqslant \binom{n/3k}{\ell}$$

choices for A_P, all of which are equally likely. It follows that

$$\begin{aligned}
\mathbf{H}[A_P \mid P] &\geqslant \log_2 \binom{n/3k}{\ell} \\
&\geqslant \log_2 \left(\frac{n/3k}{\ell}\right)^{\ell} \\
&= \ell \cdot \log_2 \frac{n}{3k\,\ell} \\
\text{(since } \ell \leqslant n/2k\text{)} \quad &= \Omega(\ell).
\end{aligned}$$

Similarly to the analysis for the oblivious adversary we obtain that, conditioned on the output of P, the entropy of A_P becomes close to 0. Thus it follows that player P needs to learn at least $\Omega(\ell)$ bits of information while executing the update algorithm, which takes $\Omega\left(\frac{\ell}{k \log n}\right)$ rounds.

3 An Algorithm for Batch-Dynamic Maximal Matching

In this section, we describe an algorithm that satisfies the bounds claimed in Theorem 2, and prove its correctness.

We first describe some information dissemination tools that our algorithm makes frequent use of. In particular, the next lemma provides a technique for efficiently spreading a set of messages to all players.

▶ **Lemma 8** (Spreading). *Suppose that there are N tokens, each of size $\Theta(\log n)$ bits, located at arbitrary players. There exists a deterministic algorithm Spreading such that each player can receive all N tokens in $O\left(\lceil \frac{N}{k} \rceil\right)$ rounds.*

Proof. We describe the Spreading algorithm and argue the claimed time complexity bound. Suppose player P_i holds a set of x_i tokens initially, and recall that $\sum_{i=1}^{k} x_i = N$.

1. **Redistribution:** P_i splits this set into batches of size exactly $k-1$ (with the possible exception of the last batch). Then, in parallel for each $j \in [k-1]$, player P_i sends the j-th token of the first batch to the player that it is connected to via its j-link. Afterwards, it proceeds with the second batch and so forth. All players perform this in parallel until all of their batches of size exactly $k-1$ have been processed. This process completes in $O(N/k)$ rounds.
   At this point, each player P_i has $x'_i \leqslant k-2$ tokens left that it has not yet redistributed. Next, the players exchange the counts $x'_1, \ldots, x'_k$ which takes $O(1)$ rounds. Every player P_a fixes some arbitrary order $m^a_1, \ldots, m^a_{x'_a}$ of its tokens, and, for every $j \in [x'_a]$, locally computes the *rank of token* m^a_j, denoted by $r(m^a_j)$, as

$$r(m^a_j) = \sum_{i=1}^{a-1} x'_i + j.$$

   It is straightforward to verify that this results in a unique rank for all tokens of all players. Then, P_a sends the token with rank r to player $P_{(r \bmod k)+1}$. Since $x'_a \leqslant k-2$, all remaining tokens can be sent in parallel in just a single round. This ensures that every player P_i holds a set T_i of at most $O(N/k)$ tokens locally.
2. **Dissemination:** P_i sequentially broadcasts every token in its set T_i to all players. After $O(N/k)$ rounds, every player is guaranteed to have received every token. ◀

3.1 Initialization

The initialization phase of our randomized algorithm simply computes a maximal matching on the input graph. As a maximal matching in a graph G is equivalent to a maximal independent set (MIS) in its line graph $\mathcal{L}_G$, a common approach is to simulate an MIS algorithm on $\mathcal{L}_G$. However, in contrast to the congested clique model, the overhead of sending $O(m)$ messages for simulating $\mathcal{L}_G$ is too expensive in our setting, as the total available bandwidth in a single round is only $O(k^2 \log n)$ bits. Instead, we employ the maximal matching algorithm of Israeli and Ittai [19]. As we also leverage this algorithm for handling the updates under an oblivious adversary, we describe its implementation in the k-clique message passing model in more detail.

Every player keeps track of a Boolean variable $\mathsf{matched}_u$, initialized to 0, for each of its hosted vertices u, which indicates if an edge incident to u is included in the matching. We now describe a single iteration of the algorithm of [19]:

1. **Edge Sampling Step**: Every player P uniformly at random chooses an incident edge $e = \{u, v\}$, for each of its vertices u, and sends a $\langle \mathsf{marked}, e \rangle$ message to player $P(v)$ via Spreading. Conceptually, we consider the marked edge $e' = \{u, v'\}$ to be a directed edge $e' = (u, v')$ (pointing from u to v') and use $\vec{G}$ to denote the directed graph consisting of the sampled edges of all players. After this step, every player P knows the subgraph of $\vec{G}$ induced by the marked edges that are incident to its own vertices.
2. **Reduce In-degrees Step**: Player P chooses a random incoming edge $e' = (v', u) \in E(\vec{G})$ for each of its vertices u (if any), and sends a $\langle \mathsf{selected}, e' \rangle$ message to player $P(v')$ via Spreading. Let $\bar{G}$ be the *undirected* graph that only consists of all edges e, for which a $\langle \mathsf{marked}, e \rangle$ as well as a $\langle \mathsf{selected}, e \rangle$ message was sent.
3. **Match-up Step:** Next, each player P samples, for each of its vertices u in $\bar{G}$, an incident edge $e'' = \{u, v''\} \in E(\bar{G})$ uniformly at random, and sends $\langle \mathsf{request}, e'' \rangle$ to $P(v'')$. If P also receives a $\langle \mathsf{request}, e'' \rangle$ message from $P(v'')$, then it sets $\mathsf{matched}_u \leftarrow 1$ (and $P(v'')$ will set $\mathsf{matched}_{v''} \leftarrow 1$ in turn).
4. **Pruning Step**: Consider an edge $e = \{u, v\}$ that was matched in the previous step, and suppose that the ID of u is smaller than the ID of v. We say that player $P(u)$ is *responsible* for e. Finally, we need to compute the residual graph that does not include any edges to already matched nodes. To this end, each player P creates a message $\langle \mathsf{matched}, e \rangle$, for every edge e for which it is responsible, and this message is sent to every other player via Spreading,. After these exchanges, the players have learned about all edges that have at least one matched endpoint, which they discard from the graph. The players locally check whether there are any edges remaining and, if so, proceed to the next iteration. The maximal matching is simply the union over the matchings obtained in the individual phases.

The next lemma quantifies the performance of this algorithm in the k-clique message passing model:

▶ **Lemma 9.** *Consider an n'-node graph G' in which the vertices are partitioned among k players, either randomly or in a balanced manner. Then, there exists a maximal matching algorithm that terminates in $O\Big(\Big\lceil \frac{n'}{k} \Big\rceil \log n'\Big)$ rounds with high probability (in n').*

Our algorithm ensures that the following invariant holds after the initialization phase and every time the algorithm has completed the computation of the new matching following a batch of edge updates:

▶ **Invariant 1.** *Every player knows, for each of its hosted vertices v, the matched edge incident to v (if any), and, for each neighbor u of v, it knows whether u is matched.*

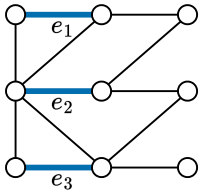
By instantiating Lemma 9 with the initial input assignment, i.e., $n' = n$, we immediately obtain the following:

▶ **Lemma 10.** *The initialization time is $O\big(\frac{n}{k} \log n\big)$ rounds with high probability and Invariant 1 holds.*

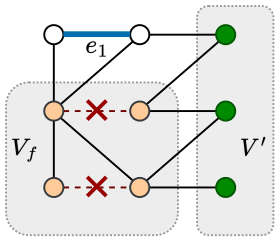
Recall that a batch of updates may consist of ℓ_1 edge deletions and ℓ_2 edge insertions, where $\ell_1 + \ell_2 \leqslant \ell$. Conceptually, our algorithm treats this as two separate update steps with batches of size ℓ_1 and ℓ_2, respectively. This allows us to assume that each batch contains only one type of edge-update in the following two subsections.

3.2 Edge Insertions

▶ **Lemma 11.** *If the (oblivious or adaptive) adversary inserts a batch of ℓ edges, then the update time is $O(\ell/k)$ rounds and Invariant 1 holds.*



(a) A maximal matching $\{e_1, e_2, e_3\}$ in graph G.



(b) The adversary has deleted edges e_2 and e_3 from G. The light orange-shaded nodes form V_f, whereas V' consists of the dark green-shaded nodes.

Figure 3 The residual graph after edge deletions.

Proof. If an edge is inserted between vertices u and v, and one of them is already matched, there is nothing to do. Thus, assume that all edges are inserted between unmatched nodes. Every player that hosts a vertex incident to some of the inserted edges sends the IDs of the endpoints of these edges to player P_1. Since this amounts to $O(\ell \log n)$ bits in total P_1 can receive all of these messages in $O(\ell/k)$ rounds. Then, P_1 locally computes a maximal matching M' on the subgraph induced by the edges that were inserted between unmatched nodes and sends M' to every other player. Since $|M'| = O(\ell)$, this again takes $O(\ell/k)$ rounds. From this information, each player learns about all newly matched edges that are incident to its own vertices, which ensures that Invariant 1 is satisfied. ◀

3.3 Edge Deletions

We now focus on the significantly more difficult case of handling deletions of ℓ edges. Since our algorithm can simply ignore deletions of non-matched edges, we assume that all removed edges were part of the current matching.

Conceptually, we split these updates into $\lceil \ell/\beta \rceil$ smaller *mini-batches* and process one mini-batch at a time. All mini-batches have a size of exactly β, with the possible exception of the last one, which contains at most β updates. For the oblivious adversary, we choose a mini-batch size of $\beta = k$, whereas, for the adaptive adversary, we set $\beta = \lfloor \sqrt{k} \rfloor$.

For a given subset of vertices $S \subseteq V(G)$, let $G[S]$ denote the subgraph induced by S. Suppose that we are given some (not necessarily maximal) matching M in G. We say that a vertex is *free* or *unmatched*, if it does not have an incident edge in M. We define the *free degree* $f_{G,M}(u)$ of a node u to be the number of neighbors incident to u in G that are free according to the matching M. For a given matching M' in $G[S]$, we sometimes abuse notation and simply write $f_{S,M'}(u)$ instead of $f_{G[S],M'}(u)$; we omit the subscripts when they are clear from the context.

The next lemma captures important properties of the resulting graph obtained right after the adversary has deleted a set of edges, denoted by D. We define V_f to be the set of unmatched nodes incident to edges in D, and let $V' \subseteq V(G) \setminus V_f$ be the set of unmatched vertices that have neighbors in V_f. Figure 3 gives an example of these sets.

▶ **Lemma 12.** *Suppose that Invariant 1 holds in the current graph G with maximal matching M. Assume that the adversary deletes a set D of at most β edges, and let G' denote the resulting graph. Then:*

(a) $|V_f| \leqslant 2\beta$*;*

(b) *for all $u \in V'$ we have $f_{G',M}(u) \leqslant 2\beta$;*

(c) *V' is an independent set in G (i.e., $G[V']$ contains no edges).*

Proof. Property (a) is immediate from the fact that at most β edges are deleted. Next, consider Property (c): Recall that every $v \in V'$ was itself unmatched in G. Since Invariant 1 ensures that the algorithm has computed a maximal matching on G, there cannot be any edge between nodes in V'.

Finally, for Property (b), observe that (c) implies $f_{G,M}(v) = 0$ for all $u \in V'$, and the only way that the free degree of v can increase in G' is if u is connected to some of the nodes in V_f. ◀

For simplicity, we assume that the given mini-batch of updates results in the deletion of a set D of exactly β edges from G that were part of the matching M, yielding the graph G'.

3.3.1 Special Case: Updating Graphs with High Free Degree in $O(1)$ Rounds

Before we explain the details of our general update algorithm in Section 3.3.2, we first give a simple and time-optimal way of handling graphs where every node has a high free degree.

As a first step, every player sends the IDs of all its nodes that are in V_f to every other player via Spreading, and hence every player learns the IDs of all nodes of V_f. Let $L = (v_1, v_2, \ldots, v_{2\beta})$ be an ordering of the vertices in V_f in increasing order of their IDs. By locally processing the vertices in V_f sequentially according to L, each player P tries to pair up every node $v_i \in V_f$ with some edge e_i that connects v_i to one of its unmatched neighbors $w \in V' \cap V(P)$, if available. If P pairs up a free vertex w with v_i, then it marks w as taken, and does not consider it when pairing up the remaining nodes $v_{i+1}, \ldots, v_{2\beta}$. Once it has processed the entire list L, player P sends the paired-up edges $\{v_i, w\}$ and $\{v_{i+\beta}, w'\}$ directly to P_i, for each $i \in [1, \beta]$; if P was unable to find a suitable edge for a vertex, it simply omits the corresponding message.

Consequently, every player P_i receives at most 2 edges from each of the $k-1$ other players. Subsequently, P_i performs a local "filtering" step by discarding from the received edges all but one incident edge e_i for v_i and an incident edge $e_{i+\beta}$ for $v_{i+\beta}$. If e_i exists, then it becomes part of the matching, and we handle $e_{i+\beta}$ analogously. Then, P_i broadcasts the (at most two) matched edges to all other players, and each player updates the incident matched edges of its vertices accordingly.

▶ **Lemma 13.** *Let G be a dynamic graph where, for every vertex u, it holds that $f_{G,M}(u) \geqslant 2(k\beta + \beta) + 1$. There exists an deterministic algorithm that can update the maximal matching in constant rounds.*

Proof. We first show that every node u with $f_{G,M}(u) \geqslant 2(k\beta + \beta) + 1$ is matched: Each time that a player P pairs-up a node $v_i \in V_f$, which occurs at index i of the ordered list L, with one of its neighbors $w \in V(P) \cap V'$, this may reduce the available edges for pairing up a node v_j ($j \in [i+1, 2\beta]$) by at most one. Note that w itself does not occur in L. According to the algorithm, there are k players that are executing this pair-up process in parallel for v_i, and thus there are at most $2k\beta$ nodes in V' that are used for pairing up. Since v_j has at most 2β neighbors in V_f, it follows that the number of unpaired free neighbors of v_j in V' is at least $f_{G',M}(v_j) - 2k\beta - 2\beta$. If v_j has high free degree, then $f_{G',M}(v_j) - 2k\beta - 2\beta \geqslant 1$, and hence there is at least one edge e available for being paired up with v_j at any point in this process, which will be sent to the corresponding player P_j. It follows that any high free degree node is guaranteed to be matched and hence has an incident edge in A_1.

Next, we argue that M_1 forms a valid matching on G': Observe that all edges used for pairing-up have one endpoint in V_f and the other in V'. Each vertex $v \in V'$ is used at most once during this process by the player who hosts v. Together with the fact that each player P_i discards all but one edge for v_i and $v_{i+\beta}$, this ensures that no edges in A_1 share an endpoint.

According to the algorithm, every player broadcasts its matched edges to everyone else and thus all players learn M_1. Finally, we show the claimed bound on the running time: The initial spreading of the 2β IDs in V_f requires $O(\beta/k) = O(1)$ rounds, according to Lemma 8. For the process of pairing up vertices with edges, each player P_i receives at most two messages from every other player, which means that P_i can learn about all paired-up edges for vertices v_i and $v_{i+\beta}$ within $O(1)$ rounds. Subsequently, P_i broadcasts at most one message for v_i and $v_{i+\beta}$ to every other player, which again takes $O(1)$ rounds. ◀

3.3.2 High-level Overview

In our analysis, we will show that the algorithm maintains Invariant 1 at the end of its update step. We conceptually split the update algorithm into two phases. In Phase 1, we focus on the unmatched nodes in V_f that have at least $2\beta + 1$ free neighbors in V'. We sample a free neighbor for each of these nodes and show that this process matches a constant fraction of these nodes with high probability in each iteration, by making use of the fact that the corresponding random variables are negatively correlated. Once the remaining number of these unmatched nodes is sufficiently small (i.e., $O(\log n)$), we sample a small subgraph of their free edges that we quickly disseminate to every player, enabling us to locally compute a matching for these nodes. In Phase 2, our approach depends on the type of adversary: For the oblivious adversary, we leverage the fact that the vertices are randomly partitioned among the players, and argue that we can directly use the same matching algorithm as in the initialization step. On the other hand, for the adaptive adversary, we use the fact that the size of the mini-batch is only $O(\sqrt{k})$, which, together with the degree reduction achieved in Phase 1, allows us to aggregate the entire subgraph induced by the remaining unmatched nodes in V_f at a single player. In the remainder of this section, we give a detailed description of these two phases.

3.3.3 Phase 1

Let $T \subseteq V_f$ be the subset of nodes that each have at least $2\beta + 1$ unmatched neighbors in V' with respect to M_1, where $M_1 = M \setminus D$. In this phase, our goal is to match all vertices in T. Recall that, at the start of Phase 1, each player P learned the IDs and name of the respective player $P(u)$ of every vertex $u \in V_f$, and hence it also knows this information for every vertex in T.

Let T_P be the subset of T hosted by player P. Observe that P knows T_P since V_f and M_1 are known to everyone. We use the shorthand $f_{V'}(u) = f_{G'[\{u\} \cup V'], M_1}(u)$ for the number of unmatched neighbors of u that are in V', and we define $f^{(i)}_{V'}(u)$ to be the number of such neighbors hosted at player P_i, i.e., $f^{(i)}_{V'}(u) = f_{G'[\{u\} \cup (V' \cap V(P_i))], M_1}(u)$. Note that P_i can locally compute $f_{V'}(u)$ and $f^{(j)}_{V'}(u)$ for every $j \in [k]$ and every vertex $u \in V(P_i)$ from its knowledge of M_1 and V_f.

We perform a sequence of $\Theta(\log k)$ iterations: Our goal, in a given iteration, is to ensure that a node in T has a constant probability of being matched to a node that is not in T. Note that $\Theta(\log k)$ iterations suffice, because Lemma 12 tells us that the size of the subgraph induced by T and its (unmatched) neighbors in V' is at most $O(\beta^2) = O(k^2)$.

All players execute the following process in parallel: For each $u \in T_P$, player P first samples an index i from $[k]$ according to the probability distribution $\left(\frac{f^{(1)}_{V'}(u)}{f_{V'}(u)}, \ldots, \frac{f^{(k)}_{V'}(u)}{f_{V'}(u)}\right)$. Then, P sends a $\langle$match-up!$\rangle$ request for u to player P_i, who in turn samples an unmatched neighbor v uniformly at random from the unmatched neighbors of u in $V(P_i) \cap V'$, and considers the $\langle$match-up!$\rangle$ message as being "received by v". If P_i does not receive any other $\langle$match-up!$\rangle$ message for v in this iteration, then it adds the edge $\{u, v\}$ to the matching and also informs P about this. Note that all of these messages are sent via Spreading (see Lemma 8).

After processing all $\langle$match-up!$\rangle$ requests, each player P updates its set T_P by removing all nodes for which it found a matching in this iteration, as well as those nodes whose number of free neighbors in V' has dropped below $2\beta + 1$. To fulfill these requirements, we need to ensure that a player knows when the free degree of one of its vertices changes, and thus we instruct each player to broadcast its newly matched nodes to everyone via Spreading (see Lemma 8). Then, every player P sends the new value of $|T_P|$ to P_1, who locally computes the sum, i.e., the updated size $|T|$, and broadcasts this count to everyone. As long as $|T| \geqslant c_1 \log n$, for a suitable constant $c_1 > 0$, the players move on to the next iteration by repeating the above process.

▶ **Lemma 14.** *If $|T| \geqslant c_1 \log n$ at the start of an iteration, then a constant fraction of the nodes in T are matched with high probability.*

Proof. We start our analysis by considering the probability of finding a matching for a node $u \in T_P$ in a given iteration. According to the algorithm, a player P randomly samples some player P_i with probability $\frac{f^{(i)}_{V'}(u)}{f_{V'}(u)}$, who in turn uniformly samples a hosted free neighbor of u with probability $\frac{1}{f^{(i)}_{V'}(u)}$. Since

$$\frac{f^{(i)}_{V'}(u)}{f_{V'}(u)} \cdot \frac{1}{f^{(i)}_{V'}(u)} = \frac{1}{f_{V'}(u)},$$

it follows that the $\langle$match-up!$\rangle$ request for u is indeed sent to a uniformly at random sampled node among its free neighbors outside T. For each $v_j \in T$, let X_j be the indicator random variable that is 1 if and only if v_j's $\langle$match-up!$\rangle$ request reaches the same node as the request of another node in this iteration, i.e., if $X_j = 0$, then v_j is matched. It follows that

$$\begin{aligned}
\mathbf{Pr}[X_j = 0] &\geqslant \left(1 - \frac{1}{f_{V'}(u)}\right)^{|T|} \\
(\text{since } T \leqslant 2\beta) \quad &\geqslant \left(1 - \frac{1}{f_{V'}(u)}\right)^{2\beta} \\
(\text{since } \forall u \in T\colon f_{V'}(u) \geqslant 2\beta + 1) \quad &\geqslant \left(1 - \frac{1}{2\beta + 1}\right)^{2\beta} \\
(\text{since } 1 - x \geqslant e^{-2x} \text{ for } x < \tfrac{1}{2}) \quad &\geqslant e^{-4\beta/(2\beta+1)}.
\end{aligned}$$

Conversely, this tells us that the probability that u is not matched in this iteration is at most some constant

$$\delta \leqslant 1 - e^{-3} \leqslant 1 - e^{-4\beta/(2\beta+1)}. \tag{9}$$

A technical complication stems from the fact that X_i and X_j are not necessarily independent, for nodes v_i and v_j that share neighbors. In particular, if a node v_i is not matched (i.e., $X_i = 1$), then its request reached the same free node $w \notin T$ that was also reached by the request of some other node. Conditioning on this event may skew the probability distribution of X_j for some distinct node v_j. However, since this event fixes the destination of two request messages to the same vertex, this cannot *increase* the probability of $X_j = 1$. In other words, conditioning on this event, it may be less likely that another $\langle$match-up!$\rangle$ message collides with v_j's own $\langle$match-up!$\rangle$ request, possibly lowering the probability of event $X_j = 1$, and hence it holds that $\mathbf{Pr}[X_j = 1 \mid X_i = 1] \leqslant \mathbf{Pr}[X_j = 1]$. A similar argument applies when considering a subset of vertices $\{v_1, \ldots, v_s\}$, that is,

$$\begin{aligned}\mathbf{Pr}\left[\bigwedge_{j=1}^{s}(X_j = 1)\right] &= \prod_{j=1}^{s}\mathbf{Pr}\left[X_j = 1 \;\middle|\; \bigwedge_{r=1}^{j-1}(X_r = 1)\right] \\ &\leqslant \prod_{j=1}^{s}\mathbf{Pr}[X_j = 1\,] \\ &\leqslant \delta^s \end{aligned} \tag{10}$$

The bound (10) enables us to instantiate the generalization of the Chernoff Bound stated in Lemma 15 with parameters $N = |T| \geqslant c_1 \log n$, $\eta = 1 - e^{-4}$. Moreover, we recall that $\delta \leqslant 1 - e^{-3}$ due to (9):

▶ **Lemma 15** (see Theorem 1.1 in [18]). *Let $X_1, \ldots, X_N$ be (not necessarily independent) Boolean random variables and suppose that, for some $\delta \in [0, 1]$, it holds that, for every index set $S \subseteq [N]$, $\mathbf{Pr}\left[\bigwedge_{i \in S} X_i\right] \leqslant \delta^{|S|}$. Then, for any $\eta \in [\delta, 1]$, we have*

$$\mathbf{Pr}\left[\sum_{i=1}^{N} X_i \geqslant \eta N\right] \leqslant e^{-2N(\eta-\delta)^2}.$$

It follows that $\mathbf{Pr}\left[\sum_{j=1}^{|T|} X_j \geqslant (1 - e^{-e})|T|\right] \leqslant e^{-\Omega(\log n)} \leqslant n^{-\Omega(1)}$, which completes the proof of Lemma 14. ◀

Once we have reached an iteration after which it holds that $|T| < c_1 \log n$, we sample a subgraph S as follows: Each player samples uniformly at random, for each of its vertices in T, a subset of $c_2 \log^2 n$ incident edges connecting to free neighbors, where $c_2 > 0$ is a suitable constant. The players perform an all-to-all exchange of the sampled edges using Spreading, after which everyone knows the entire subgraph S. In the proof of the following Lemma 16, we show that this enables the players to locally compute a matching that is maximal for all remaining nodes in T by using some fixed deterministic function f that maps each possible subgraph S to a set of matched edges. Note that f is hard-coded into the algorithm and hence known to all players.

▶ **Lemma 16.** *If $|T| < c_1 \log n$ at the end of an iteration, then all remaining nodes in T are matched in $O(1)$ additional rounds.*

Proof. We first argue the bound on the time complexity. By assumption, any given player holds at most $O(\log n)$ nodes from T. According to the algorithm, we select $c_2 \log^2 n$ edges to unmatched neighbors for each of these nodes. Note that this is possible because every node in T has a free degree of at least $2\beta + 1$ and due to the assumptions that $k = \Omega(\log^4 n)$, which means that, for both, oblivious and adaptive adversary, we have

$$2\beta + 1 = \Omega(\sqrt{k}) = \Omega\big(\log^2 n\big).$$

Moreover, since $|T| = O(\log n)$, the total number of messages that need to be sent is $O(\log^3 n) = O(k)$. Thus, Lemma 8 tells us that we can apply Spreading to disseminate these edges to all other players in $O(1)$ rounds.

After Spreading has completed, all players know the entire sampled subgraph S. Observe that each node in T has a degree of $\Omega(\log^2 n)$ in S, whereas $|T| = O(\log n)$. Thus, it follows that there always exists a maximal matching in S that matches every node in T. ◀

The next lemma combines Lemmas 14 and 16 to show a significant reduction of the free degree of the remaining unmatched nodes in V_f.

▶ **Lemma 17.** *At the end of Phase 1, every player knows the matching $M_2 = M_1 \cup N_2$, where N_2 denotes the set of newly matched edges, and $M_1 = M \setminus D$. Phase 1 takes $O(\log k)$ rounds with high probability, and matches every vertex in $v \in V_f$ that had a free degree of at least $4\beta + 1$, i.e., if $f_{G',M_1}(v) \geqslant 4\beta + 1$.*

Proof. Recall that T denotes the subset of nodes in V_f that have at least $2\beta + 1$ free neighbors in V'. It follows that any node in V_f with a free degree of at least $4\beta + 1$ must have $4\beta + 1 - 2\beta$ free neighbors in V', and hence will be in set T. Lemma 14 tells us that $|T| \leqslant c_1 \log n$ (w.h.p.) after $O(\log k)$ iterations, which, together with Lemma 16, implies the sought time complexity bound.

By the description of the algorithm, all players exchange the newly matched edges after each iteration. Moreover, once $|T| < c_1 \log n$, every player uses the same function f to compute the matching on the sampled subgraph S, which ensures that all players know M_2. ◀

3.3.4 Phase 2

Let $L \subseteq V_f$ be the remaining unmatched vertices in V_f after Phase 1. In Phase 2, we take care of the vertices in L, which we call the *low free degree vertices*, as Lemma 17 ensures that each of them has a free degree of at most 4β. We provide two approaches for this phase, depending on the strength of the adversary.

Oblivious Adversary. We first consider the setting where the adversary must choose all updates before the vertices are randomly partitioned among the players. Let $G_L \subseteq G[V_f \cup V']$ be the subgraph induced by the free edges incident to nodes in L. Observe that G_L contains all remaining unmatched nodes that have free neighbors. Similarly to the initialization phase, we use the algorithm of [19], for which we have described an implementation in Section 3.1. According to Lemma 17, every $u \in L$ has at most $4\beta + 1$ unmatched neighbors when considering the matching M_2, and we know from Lemma 12(a) that $|L| \leqslant |V_f| \leqslant 2\beta$. It follows that G_L has at most $\Gamma = O(\beta)$ vertices. Since the adversary is oblivious to the random vertex partitioning, we can instantiate Lemma 9 with $n' = \Gamma = O(k)$, since $\beta = k$. This immediately implies the following lemma:

▶ **Lemma 18.** *Phase 2 ensures Invariant 1 and takes $O(\log k)$ rounds with high probability (in k) against an oblivious adversary, assuming a mini-batch size of $\beta = k$ edge deletions.*

Adaptive Adversary. When the adversary has full knowledge of the vertex partitioning among the players, it may happen that all nodes in L are distributed among a small number of players, thus rendering the vast majority of available resources (i.e., players and bandwidth between them) useless. To avoid this pitfall, we leverage the assumption that the mini-batch

size is only $\lfloor\sqrt{k}\rfloor$. In the proof of Lemma 19, we show that we can aggregate the entire subgraph G_L at a single player P_1, who locally computes a matching, and subsequently informs all other players of the result.

▶ **Lemma 19.** *Phase 2 ensures Invariant 1 and takes $O(1)$ rounds against an adaptive adversary, assuming a mini-batch size of $\beta = \lfloor\sqrt{k}\rfloor$.*

Proof. The correctness of the computed matching is immediate from the fact that P_1 locally computes the solution. Since each node in L has a free degree of at most $4\beta \leqslant 4\sqrt{k}$ and $|L| \leqslant 2\beta \leqslant 2\sqrt{k}$, it follows that G_L consists of $O(k)$ edges. Thus, according to Lemma 8, player P_1 can receive the entire graph in just $O(1)$ rounds by executing Spreading. ◀

3.4 Completing the Proof of Theorem 2

The bound on the initialization time follows from Lemma 10, which also tells us that Invariant 1 holds after the initialization step.

For edge insertions, Lemma 11 tells us that the update time for handling ℓ edge insertions is only $O(\frac{\ell}{k})$ (for both oblivious and adaptive adversaries). Next, we derive the bound on the update time for edge deletions: Recall that we have ℓ updates that we split into mini-batches of size β. Lemma 17 shows that Phases 1 requires $O(\log k)$ rounds with high probability. Finally, Lemma 18 shows that under an oblivious adversary, Phase 2 takes $O(\log k)$ rounds with high probability (in k), for mini-batch updates of size k, which yields a total update time of $O\left(\frac{\ell}{\beta}\log k\right)$. Lemma 19 shows that, under an adaptive adversary, Phase 2 can be done in $O(\log k)$ rounds for mini-batch updates of size $\Theta(\sqrt{k})$, and hence the two phases take at most $O\left(\frac{\ell}{\beta}\log k\right)$ rounds in total with high probability. The claimed bounds of Theorem 2 follow by plugging in $\beta = k$ for the oblivious adversary and $\beta = \lfloor\sqrt{k}\rfloor$ for the adaptive adversary, respectively.

To see that each player P_i uses at most $O(\max\{n, |G[V(P_i)]|\} \cdot \log n)$ bits of local memory, where $|G[V(P_i)]|$ is the number of edges incident to vertices hosted by P_i, observe that each player sends and receives at most 1 message per vertex in every step of the algorithm, which requires storing at most $O(n \log n)$ bits. Moreover, each player needs to locally maintain at most $O(\log n)$ bits for each edge incident to its hosted vertices, using at most $O(|G[V(P_i)]| \log n)$ bits.

Finally, note that Lemma 18 and Lemma 19 guarantee that Invariant 1 holds at the end of each update step, which ensures the correctness of the computed maximal matching.

4 Batch-Dynamic Maximal Matching in the Congested Clique Model

It is not surprising that our algorithm also work in the congested clique due to the equivalence between the two models when $k = n$ and if each player gets exactly one vertex (see Section 1). However, we now show that we can even get dynamic algorithms in the Congested Clique model that handle updates *communication-efficiently*, which means that the number of messages sent grow proportionally to the number of edge-changes. To this end, we define the *update message complexity* as the worst case number of messages sent following a batch of ℓ edge updates, assuming that the state of the nodes reflects a maximal matching prior to these updates. In the full paper, we prove the following result:

▶ **Theorem 20.** *Consider the congested clique and suppose the adversary may add or delete up to ℓ edges per batch. There exists a randomized dynamic algorithm for maximal matching that, with high probability, has an initialization time of $O(\log\log\Delta)$ rounds. With high probability, the update message complexity is $O(\ell n)$, and the following hold:*

1. *The update time complexity is $O\left(\lceil\frac{\ell}{n}\rceil\log n\right)$ against an oblivious adversary.*
2. *The update time complexity is $O\left(\lceil\frac{\ell}{\sqrt{n}}\rceil\log n\right)$ against an adaptive adversary.*

We emphasize that the message complexity of the initialization algorithm in Theorem 20 can be as large as $O(n^2)$. In fact, obtaining a message complexity of $o(n^2)$ for maximal matching in the congested clique is itself an interesting open problem to the best of our knowledge.

References

1 Umut A Acar, Daniel Anderson, Guy E Blelloch, and Laxman Dhulipala. Parallel batch-dynamic graph connectivity. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 381–392, 2019.

2 Umut A Acar, Andrew Cotter, Benoît Hudson, and Duru Türkoglu. Parallelism in dynamic well-spaced point sets. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 33–42, 2011.

3 Daniel Anderson. *Parallel Batch-Dynamic Algorithms Dynamic Trees, Graphs, and Self-Adjusting Computation*. PhD thesis, Carnegie Mellon University, 2023.

4 Shiri Antaki, Quanquan C Liu, and Shay Solomon. Near-optimal distributed implementations of dynamic algorithms for symmetry breaking problems. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

5 John Augustine, Kishore Kothapalli, and Gopal Pandurangan. Efficient distributed algorithms in the k-machine model via pram simulations. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 223–232. IEEE, 2021.

6 Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local distributed algorithms in highly dynamic networks. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 33–42. IEEE, 2019.

7 Sayan Bandyapadhyay, Tanmay Inamdar, Shreyas Pai, and Sriram V Pemmaraju. Near-optimal clustering in the k-machine model. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, pages 1–10, 2018.

8 Soheil Behnezhad, Mohammad Taghi Hajiaghayi, and David G Harris. Exponentially faster massively parallel maximal matching. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1637–1649. IEEE, 2019.

9 Keren Censor-Hillel, Neta Dafni, Victor I Kolobov, Ami Paz, and Gregory Schwartzman. Fast deterministic algorithms for highly-dynamic networks. In *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

10 Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 217–226, 2016.

11 Keren Censor-Hillel, Victor I Kolobov, and Gregory Schwartzman. Finding subgraphs in highly dynamic networks. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 140–150, 2021.

12 Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *PVLDB*, 8(12):1804–1815, 2015. URL: `http://www.vldb.org/pvldb/vol8/p1804-ching.pdf`, `doi:10.14778/2824032.2824077`.

13 Laxman Dhulipala, David Durfee, Janardhan Kulkarni, Richard Peng, Saurabh Sawlani, and Xiaorui Sun. Parallel batch-dynamic graphs: Algorithms and lower bounds. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1300–1319. SIAM, 2020.

14 Klaus-Tycho Foerster, Janne H Korhonen, Ami Paz, Joel Rybicki, and Stefan Schmid. Input-dynamic distributed algorithms for communication networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(1):1–33, 2021.

15 Seth Gilbert and Lawrence Er Lu Li. How fast can you update your MST? In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 531–533. ACM, 2020. `doi:10.1145/3350755.3400240`.

16 Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *USENIX OSDI 2014*, pages 599–613, 2014. URL: `https://www.usenix.org/conference/osdi14/technical-sessions/presentation/gonzalez`.

17 Khalid Hourani, Hartmut Klauck, William K Moses Jr, Danupon Nanongkai, Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. Distributed algorithms for large-scale graphs. *arXiv e-prints*, pages arXiv–1311, 2023.

18 Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 617–631. Springer, 2010.

19 Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986.

20 Giuseppe F Italiano, Silvio Lattanzi, Vahab S Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, pages 49–58, 2019.

21 Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM, 2010.

22 Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. Distributed computation of large-scale graph problems. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 391–410. SIAM, 2014.

23 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005. `doi:10.1137/S0097539704441848`.

24 Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD Conference*, pages 135–146, 2010. `doi:10.1145/1807167.1807184`.

25 Krzysztof Nowicki and Krzysztof Onak. Dynamic graph algorithms with batch updates in the massively parallel computation model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2939–2958. SIAM, 2021.

26 Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. *ACM Transactions on Parallel Computing (TOPC)*, 8(2):1–28, 2021.

27 Kenneth H Rosen. *Discrete mathematics and its applications*. The McGraw Hill Companies, 2007.