Fully Dynamic Sequential and Distributed Algorithms for MAX-CUT

Omer Wasim
Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA
wasim.o@northeastern.edu

Valerie King
Department of Computer Science, University of Victoria, Canada
val@uvic.ca

Abstract

This paper initiates the study of the MAX-CUT problem in fully dynamic graphs. Given a graph $G = (V, E)$, we present deterministic fully dynamic distributed and sequential algorithms to maintain a cut on G which *always* contains at least $\frac{|E|}{2}$ edges in sublinear update time under edge insertions and deletions to G. Our results include the following deterministic algorithms: i) an $O(\Delta)$ *worst-case* update time sequential algorithm, where Δ denotes the maximum degree of G, ii) the first fully dynamic distributed algorithm taking $O(1)$ rounds and $O(\Delta)$ total bits of communication per update in the Massively Parallel Computation (MPC) model with n machines and $O(n)$ words of memory per machine. The aforementioned algorithms require at most *one* adjustment, that is, a move of one vertex from one side of the cut to the other.

We also give the following fully dynamic sequential algorithms: i) a deterministic $O(m^{1/2})$ amortized update time algorithm where m denotes the maximum number of edges in G during any sequence of updates and, ii) a randomized algorithm which takes $\tilde{O}(n^{2/3})$ worst-case update time when edge updates come from an oblivious adversary.

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms

Keywords and phrases data structures, dynamic graph algorithms, approximate maximum cut, distributed computing, parallel computing

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2020.33

Funding This research was completed while the first author was a MSc candidate at the University of Victoria. Both authors were funded by a NSERC Discovery Grant.

Acknowledgements The authors thank Hung Le for useful discussions and an anonymous reviewer for helpful comments.

1 Introduction

A fully dynamic graph algorithm is a data structure to maintain a property of a graph under an arbitrary sequence of edge insertions and deletions. The goal is to update the graph in less time than the best static algorithm which computes the property from scratch. A *fully dynamic* graph algorithm may incur preprocessing time, after which it is able to answer queries regarding the maintained property. Research in this area has focused mostly on dynamic variants of well-known problems such as connectivity [42, 26, 30, 15], minimum spanning trees [24, 26, 47], minimum cut [45], etc., all of which admit polynomial time exact algorithms in the static setting.

Following the seminal work of Onak and Rubinfeld [40] in which fully dynamic algorithms for maintaining constant factor approximations of maximum matching (and vertex cover) were presented, research in dynamic algorithms has broadened to include approximate versions of NP-hard problems. Some natural directions arising in this setting include the design of dynamic algorithms to maintain an approximate solution in sublinear update time and the study of approximability-time trade-off. A list of approximate versions of NP-hard problems investigated in the dynamic setting includes vertex cover [5, 43, 9, 38], set-cover [22], dominating set [25], graph coloring [8], facility location [21] and maximum independent set [23, 3, 4].

In this paper, we initiate the study of the MAX-CUT problem in fully dynamic graphs and pose the question of whether there exist sublinear update time algorithms. Another parameter we look at is the *adjustment cost*, which is defined in a dynamic graph problem as the amount of changes to the maintained solution per update. In the case of MAX-CUT, it is the number of vertices which move from one subset of the cut to the other.

MAX-CUT is one of the fundamental NP-hard problems [32] which continues to be widely studied. Some of its concrete applications arise in the design of integrated circuits [12], communication networks [14] and statistical physics [41]. It also models a standard 2-clustering objective for partitioning a graph such that the number of inter-cluster edges is maximized.

Let $G = (V, E)$ be an undirected, unweighted graph $G = (V, E)$ with $n = |V|, m = |E|$. A cut C is a partition of the vertex set V, and denoted by $C = (S, \bar{S})$, where $S, \bar{S} \subseteq V$ and $\bar{S} = V \backslash S$. The cut-set $E(S, \bar{S})$ of $C = (S, \bar{S})$ is the set of all edges which have exactly one endpoint in S. A cut edge of C is an edge contained in the cut-set $E(C) = E(S, \bar{S})$. A maximum cut of G is a cut whose cut-set is largest among cut-sets for all possible cuts, i.e. $\text{MAX-CUT}(G) = \arg\max_{C=(S,\bar{S}),\, S \subseteq V} |E(S, \bar{S})|$, where $|E(S, \bar{S})|$ denotes the number of cut edges. We say a cut is t-respecting if $|E(C)| \geq t|E|$. Note that the cut-set of a t-respecting cut contains a t fraction of all edges, regardless of the size of the largest cut-set. Let OPT denote the size of the largest cut-set. A cut is t-approximate if $|E(C)| \geq t \cdot OPT$ and a t-approximation algorithm for MAX-CUT yields a t-approximate cut. It follows that a t-respecting cut is *always* a t-approximate cut but not vice-versa. This distinction can be appreciated in the case of K_{2n}, the complete graph on $2n$ vertices where a maximum cut is any cut $C = (S, \bar{S})$ where $|S| = n$. For large n, the size of the cut-set of a $\frac{1}{2}$-respecting cut can be nearly twice the size of a $\frac{1}{2}$-approximate cut. Throughout this paper, we let $[k]$ to denote $\{1, 2, .., k\}$, Δ to be the maximum degree of G and $\tilde{O}$ to hide a $O(\text{polylog}(n))$ factor.

The Massively Parallel Computation (MPC) model was introduced by Karloff et al. [31] and later refined in [20, 6, 1] as a theoretical framework for large scale parallel processing settings such as those in [48, 17]. There are μ machines with S words of memory each, which solve a problem by synchronously communicating over an all-to-all communication network (i.e. a complete network). Initially, input data of size N (which is $O(m + n)$ in the case of a graph problem) are distributed across these machines. It is desirable to have μ and S to be $O(N^{1-\epsilon})$ for some $\epsilon > 0$ and the message size is limited to $O(S)$ bits. In each round, every machine can: i) receive messages of the previous round from other machines ii) do local polynomially bounded computation (i.e. taking poly(S) space and time) without additional communication and iii) send messages to other machines which are received in the next round. The complexity of a MPC algorithm to solve a problem is determined by 3 parameters: i) the number of rounds of communication, ii) the size of the memory per machine and iii) the total amount of communication per round. Typically, MPC algorithms for graph problems use $O(n)$ machines, $\tilde{O}(n)$ memory per machine and take $\tilde{O}(1)$ rounds of communication.

1.1 Previous Work

Static sequential algorithms for $\frac{1}{2}$-respecting cuts. A simple randomized algorithm, hereafter referred to as Randomized Max-Cut obtains a $\frac{1}{2}$-respecting cut $C = (S, \bar{S})$ *in expectation* by placing each vertex independently in S or $\bar{S}$ with probability $\frac{1}{2}$. Any edge $e = \{u, v\}$ is a cut edge of C with probability $\frac{1}{2}$, implying the result. Randomized Max-Cut can be derandomized using the method of conditional expectation or pairwise independence.

Johnson's folklore algorithm [28] hereafter referred to as Greedy Max-Cut, which finds a $\frac{1}{2}$-respecting cut can be viewed as derandomized version of Randomized Max-Cut using the method of conditional expectation. Given $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ it starts with $S = \{v_1\}, \bar{S} = \emptyset$. Each successive vertex v_j, where $j \geq 2$ is added to S or $\bar{S}$ depending on which contains fewer of its neighbors v_i, where $i < j$. Thus, at least half of all edges of the form $\{v_j, v_i\}$ where $i < j$ are contained in the resulting cut. Since each vertex and edge is encountered once, the running time of Greedy Max-Cut is $O(m + n)$.

Randomized Max-Cut can also be derandomized using the idea of pairwise independence [37]. For a set S, let $\mathcal{P}(S)$ denote the power set of S. We first note that one can get a $\frac{1}{2}$-respecting cut (in expectation) which uses only $k = \lceil \log n \rceil$ independent random bits. The idea is to construct a one-to-one function $f : V \rightarrow \mathcal{P}([k])$ and choosing R to be a uniformly random subset of $[k]$. It can be shown that the cut $C = (S, \bar{S})$ where $S = \{v|\,|f(v) \cap R| \text{ is even}\}$ and $\bar{S} = \{v|\,|f(v) \cap R| \text{ is odd}\}$ is $\frac{1}{2}$-respecting in expectation. Enumerating all the $2^k = O(n)$ possibilities for R, and taking the cut which maximizes $|E(S, \bar{S})|$ yields a $\frac{1}{2}$-respecting cut. For a fixed R, the time to compute C is $O(nk)$ while determining the size of $C's$ cut-set takes $O(m)$ time giving a total time of $O(n^2 \log n + mn)$. While this algorithm isn't better in terms of running time as compared to Greedy Max-Cut, it has the advantage of being parallelizable.

Static distributed algorithms for $\frac{1}{2}$-respecting cuts. We observe that the algorithm obtained by derandomizing Randomized Max-Cut via pairwise independence can be used to compute a $\frac{1}{2}$-respecting cut in $O(1)$ rounds in the MPC model of computation with n machines and $\Theta(n)$ memory per machine. We assume there exists a fixed coordinator machine. Given f, each machine corresponds to a vertex v, and stores $f(v)$ along with the list of $v's$ neighbors and $R \subseteq [k]$ which is fixed. In the first round, each machine first computes the count of the number of edges its corresponding vertex is incident to in the cut obtained by considering the i^{th} choice of R where $i \in [n]$. Then each machine sends the i^{th} count to machine i. In the next round all machines send these counts to the coordinator, which chooses a $\frac{1}{2}$-respecting cut and informs all other machines. Thus, at the end of the third round, each machine is able to output the position of its corresponding vertex in the $\frac{1}{2}$-respecting cut. The total amount of communication is bounded by $O(n^2 \log n)$ bits.

A similar adaptation of Greedy Max-Cut in the MPC model with n machines and $\Theta(n)$ memory per machine takes n rounds of communication and $O(n\Delta)$ total communication.

The only *deterministic* distributed algorithm to compute a $\frac{1}{2}$-respecting cut that we are aware of was presented by Censor-Hillel et al. [11] which takes $\tilde{O}(\Delta + \log^* n)$ rounds and $\Omega(\Delta^2)$ messages in the $\mathcal{CONGEST}$ model. Their algorithm can be adapted to the Congested-Clique setting with the same round and message complexity.

Approximation Algorithms for MAX-CUT. We briefly survey the relevant literature on approximation algorithms for MAX-CUT in the static setting. Goemans and Williamson (1994) used a semidefinite programming (SDP) relaxation [19] and randomized rounding to yield a 0.878-approximation to MAX-CUT. This polynomial-time algorithm runs in super-linear time using state-of-the art numerical methods for solving a semidefinite program. Khot et al. showed that MAX-CUT is hard to approximate better than 0.878 [35] under the Unique Games Conjecture [34].

Arora and Kale [2] presented a primal dual (SDP-based) $(0.878 - \epsilon)$-approximation algorithm which runs in $\tilde{O}(m)$ time for d regular graphs with high probability where the running time depends inversely on ϵ. Trevisan later presented a 0.53-approximation algorithm for MAX-CUT utilizing spectral techniques [46] whose analysis was improved to 0.62 by Soto [44]. In the same paper, Trevisan showed that the primal dual SDP-based algorithm of [2] can be made to run in $\tilde{O}(m)$ time for any degree, via a linear time reduction to reduce the maximum degree to $O(\text{polylog}(n))$. By using the algorithm of Arora and Kale [2] together with the rounding scheme of Charikar and Wirth [13], we note that in graphs in which the size of the optimal cut is $(\frac{1}{2} + \epsilon)|E|$, one can get a $(\frac{1}{2} + \Omega(\frac{\epsilon}{\log(1/\epsilon)}))$-respecting cut in $\tilde{O}(m)$ time. However, when $\epsilon = O(\frac{1}{n})$ (as in the case of K_{2n}) and a $\frac{1}{2}$-respecting cut is desired (instead of a $\frac{1}{2}$-approximate cut) this can take $\Omega(mn)$ time.

Kale and Seshadhri [29] presented a combinatorial algorithm based on the spectral method [46] which uses random walks to give a $(0.5+\epsilon)$-approximation with running time depending on ϵ. For $\epsilon = 0.0155$, the running time is $\tilde{O}(n^2)$. As the running time increases, the approximation ratio converges to the spectral algorithm of Trevisan [46].

1.2 The Fully Dynamic Model

In this paper, we seek to maintain a $\frac{1}{2}$-respecting cut in sublinear update time and handle meaningful queries such as determining whether an edge is in the cut-set, the size of vertex partitions and the cut-set in *constant time*. We define the Fully Dynamic Max-Cut problem as follows:

▶ **Problem 1** (Fully Dynamic MAX-CUT)**.** Starting with a graph $G = (V, E)$ on n vertices and an empty edge set E, maintain a $\frac{1}{2}$-respecting cut $C = (S, \bar{S})$ for G under edge insertions and deletions to E such that queries of the following form can be handled in constant time: i) Is the edge $\{v_i, v_j\}$ contained in the cut-set $E(S, \bar{S})$? ii) What is the size of the cut-set, $E(C)$? iii) What are the sizes of S and $\bar{S}$?

Our goal is to update C in $o(m + n)$ time to fare better than running Greedy Max-Cut after every update and we require that answers to all queries between any two updates must be consistent with respect to the maintained cut C.

In the fully dynamic MPC model that we consider in this paper, we start with a graph $G = (V, E)$ on n vertices and m edges. Let $N = O(m + n)$. We use a coordinator machine which can be selected in a single round: machines send their ID's to all other machines and the coordinator is selected to be the machine with ID larger than all ID's it receives. There are a total of n machines each with $\Theta(n)$ memory and the goal is to maintain a $\frac{1}{2}$-respecting cut in $O(1)$ rounds per edge update and $O(n)$ total communication per round. Each machine corresponds to a vertex of G and stores the edges incident to it. After any update $\{u, v\}$ to the graph, the machines corresponding to u and v are informed of the update. In our model, we insist on algorithms which make few adjustments to the maintained cut. We note that there are other fully dynamic MPC models that have been studied very recently such as in [27, 18, 39]. Our dynamic algorithm in the MPC model requires at most one adjustment. Ensuring this is easier in the case of problems such as maximal matching where only the neighborhood of endpoints of the updated edge needs to be examined per update. In our case, this may not always be the case (see Theorem 9).

Attaining a *deterministic worst-case* update time (i.e. without randomization or amortization) is an important objective in the design of dynamic algorithms. For the seminal problem of dynamic connectivity, deterministic algorithms beating $O(\sqrt{n})$ update time [15, 33] were only recently discovered after decades. Another example is the maximal independent set problem for which known deterministic algorithms [3, 23] only achieve a sublinear (in m) amortized update time and polylogarithmic update time algorithms are yet to be discovered.

An event happens with high probability (w.h.p) if its probability is $1 - \frac{1}{n^c}$ for any $c > 0$. For our randomized algorithm, we assume that updates come from an oblivious adversary. This is a standard assumption used in the design of many randomized dynamic algorithms. An oblivious adversary is one which cannot choose updates adaptively in response to the answers returned by queries. Thus, updates to the graph can be assumed to be fixed in advance. We assume the existence of an oracle which randomly labels each vertex uniquely using a number in $\{1, ..., n\}$, and to which the adversary is oblivious. This is only used in the algorithm of Theorem 6. We seek to maintain a $\frac{1}{2}$-respecting cut *exactly* or w.h.p.

1.2.1 Dynamic algorithms from static via lazy recomputation

The following observation allows one to obtain dynamic algorithms by using known static algorithms as subroutines.

▶ **Observation 2.** Given a t-respecting (resp., t-approximation) static algorithm $\mathcal{A}_S$ for MAX-CUT which runs in time $T(m, n)$, there exists a fully dynamic algorithm $\mathcal{A}_D$ which maintains a $(t - \epsilon)$-respecting (resp., $(t - \epsilon)$-approximate) cut for any constant $\epsilon > 0$ in $O(\frac{T(m,n)}{\epsilon m})$ worst-case update time.

The proof of Observation 2 is deferred to the Appendix. Using observation 2 gives the following fully dynamic algorithms. For any constant $\epsilon > 0$, a $(\frac{1}{2} - \epsilon)$-respecting cut can be maintained in $O(1/\epsilon)$ worst-case update time by using Greedy Max-Cut. Similarly, the algorithm of [2] yields a dynamic algorithm to maintain a $(0.878 - \epsilon)$-*approximate* cut (w.h.p) for a fixed constant $\epsilon > 0$ in $O(\text{polylog}(n))$ worst case update time. For instances where the size of the cut-set of the optimal cut contains $(\frac{1}{2} + \epsilon)|E|$ edges, the rounding algorithm of [13] can be used together with the algorithm of [2] to get a $(\frac{1}{2} + \Omega(\frac{\epsilon}{\log 1/\epsilon}))$-respecting cut in $O(\text{polylog}(n))$ worst case update time where $\epsilon > 0$ is a constant. However, to maintain a $\frac{1}{2}$-respecting cut for graphs in which the optimal cut is $(\frac{1}{2} + O(\frac{1}{n}))$ the update time using this technique can be $\tilde{\Omega}(n)$ time which is prohibitive. Thus there remains a need to design dynamic algorithms to maintain a $\frac{1}{2}$-respecting cut *exactly* in *sublinear* update time.

1.3 Our Contribution

We present the first fully dynamic algorithms in the sequential and distributed settings which *exactly* maintain a $\frac{1}{2}$-respecting cut. Our results are summarized in the following theorems.

▶ **Theorem 3.** *There exists a deterministic fully dynamic sequential algorithm which maintains a $\frac{1}{2}$-respecting cut, requires no more than one adjustment per update and takes $O(\Delta)$ worst case update time, where Δ denotes the maximum degree of the graph after the update.*

The algorithm in Theorem 3 is used as a subroutine in all other algorithms in this paper. The next result gives the first fully dynamic *deterministic* algorithm in the MPC setting with n machines and $\Theta(n)$ memory per machine to maintain a $\frac{1}{2}$-respecting cut. Our algorithm takes $O(1)$ rounds, requires no more than one adjustment and uses $O(\Delta)$ total communication per round. This significantly improves on the parallel implementation of the static algorithm to maintain a $\frac{1}{2}$-respecting cut based on the idea of pairwise independence which can take as much as $O(n^2 \log n)$ total communication and $\Omega(n)$ adjustments.

▶ **Theorem 4.** *Given a graph on n vertices and m edges, there exists a deterministic fully dynamic MPC algorithm on n machines having $\Theta(n)$ memory each, which maintains a $\frac{1}{2}$-respecting cut on G and takes $O(1)$ rounds, makes at most one adjustment, and uses $O(\Delta)$ bits of communication per update. If we start with an arbitrary graph, the preprocessing for the algorithm takes $O(1)$ rounds and $O(n^2 \log n)$ bits of communication.*

We note that the worst-case update time of $O(\Delta)$ can be quite large in the case when $\Delta = \Omega(n)$ and thus costly in the dynamic setting. This motivates the design of sublinear update time algorithms for all regimes of Δ. Our next result is a sublinear (in m) amortized update time algorithm which is useful for sufficiently sparse graphs having high maximum degree.

▶ **Theorem 5.** *There exists a deterministic fully dynamic sequential algorithm which maintains a $\frac{1}{2}$-respecting cut, and takes $O(m^{1/2})$ amortized update time where m is the maximum number of edges in the graph during any arbitrary sequence of updates.*

Our final result is a randomized algorithm which always maintains a $\frac{1}{2}$-respecting cut and takes sublinear in n worst-case update time when updates come from an oblivious adversary.

▶ **Theorem 6.** *There exists a randomized fully dynamic sequential algorithm which maintains a $\frac{1}{2}$-respecting cut and takes $\tilde{O}(n^{2/3})$ worst-case update time with high probability.*

We note that for our algorithms in Theorems 5 and 6, the adjustment cost can be $\Omega(n)$.

1.4 Our techniques

Our techniques utilize combinatorial and structural properties of cuts in graphs. The key insight underlying our algorithms is the following: in any cut C which is not $\frac{1}{2}$-respecting, there exists a vertex which can be moved across the cut to increase the size of $C's$ cut-set. We show that this vertex can be efficiently found, yielding a simple deterministic $O(\Delta)$ worst case update time algorithm. This algorithm is not "local" in the sense that endpoints of the updated edge need not qualify as vertices which can be moved to increase the number of cut edges (Theorem 9). Such locality is often exploited to obtain dynamic and distributed algorithms for problems such as vertex cover, independent set and coloring. Despite this, we show that the algorithm can be used to get a deterministic fully dynamic distributed algorithm taking $O(1)$ rounds and no more than one adjustment.

Central to our sublinear time algorithms of Theorem 5 and 6 is a *cut-combining* technique. This allows us to work on induced subgraphs of G and combine their "locally maintained" cuts to yield a $\frac{1}{2}$-respecting cut on G. However, the update time depends the complexity of maintaining $\frac{1}{2}$-respecting cuts on individual subgraphs and the combining step. We work around this non-trivial dependence. For our algorithm of Theorem 5, we partition vertices based on their degree and only *selectively* update data structures. We show that selective updating is sufficient for our purpose and refine the vertex partition after sufficiently many updates. This leads to a simple $O(m^{1/2})$ amortized update time algorithm. To obtain the algorithm of Theorem 6, we extend the cut-combining idea and apply it to a random multi-way k-partition of V and obtain a sublinear in n worst case update time algorithm.

1.5 Organization of the paper

In the next section, we present an $O(\Delta)$ update time algorithm. In Section 3, we present the dynamic distributed algorithm of Theorem 4. In Section 4, we give the $O(m^{1/2})$ amortized update time sequential algorithm of Theorem 5. In section 5, we give the randomized algorithm of Theorem 6.

2 Preliminaries

Starting with an empty graph $G = (V, E)$ where $V = \{v_1, ..., v_n\}$ is fixed, an update to G is either an insertion or a deletion of an edge $\{v_i, v_j\}$ from E. For a cut $C = (S, \bar{S})$ let $\alpha_C(G) = \frac{|E(S,\bar{S})|}{|E|}$ denote the ratio of the sizes of $C's$ cut-set and E. The sizes of sets $S, \bar{S}$ and the cut-set $E(S, \bar{S})$ corresponding to the cut $C = (S, \bar{S})$ are maintained by all algorithms to facilitate queries in constant time. Let $G_k = (V, E_k)$ be the resulting graph after k updates have been made to $G := G_0$ and m denote the number of edges in the graph at any given time. The degree of any vertex v in G_k is denoted by $deg_k(v)$.

The cut on G_0, the empty graph is initialized to $(V, \emptyset)$. Given a $\frac{1}{2}$-respecting cut $C = (S, \bar{S})$, i.e. $\alpha_C(G_{k-1}) \geq \frac{1}{2}$ for some $k \geq 1$, there are a few cases to consider when an edge update $\{v_i, v_j\}$ is made to G_{k-1}. Deletion of a non-cut edge or insertion of a cut edge never decreases the size of $C's$ cut-set. However, C needs to be updated if a cut edge is deleted, or a non-cut edge is inserted since C may cease to be $\frac{1}{2}$-respecting.

2.1 A crucial observation

We say that a vertex u is switched (with respect to a cut $C = (S, \bar{S})$) if u is in S (resp. $\bar{S}$) and moved to $\bar{S}$ (resp. S). We leverage the existence of vertices which can be switched to increase the size of the cut-set $|E(S, \bar{S})|$ of C for *any* cut C which is not $\frac{1}{2}$-respecting. Thus, if C ceases to be $\frac{1}{2}$-respecting following any update there exists a vertex which can be switched to restore the $\frac{1}{2}$-respecting property.

▶ **Definition 7** (Switching vertex). *For a cut $C = (S, \bar{S})$, let $N_S(u) = \{v \in S | (u, v) \in E\}$ be the neighbors of u in S and $N_{\bar{S}}(u) = \{v \in \bar{S} | (u, v) \in E\}$ be the neighbors of u in $\bar{S}$. Then u is a switching vertex if one of the following two conditions holds: i) $u \in S$ and $|N_S(u)| - |N_{\bar{S}}(u)| \geq 1$ and ii) $u \in \bar{S}$ and $|N_{\bar{S}}(u)| - |N_S(u)| \geq 1$.*

▶ **Theorem 8.** *Let C be a $\frac{1}{2}$-respecting cut w.r.t. G_{k-1} i.e., $\alpha_C(G_{k-1}) \geq \frac{1}{2}$ and $\{v_i, v_j\}$ be an update. If $\alpha_C(G_k) < \frac{1}{2}$, then there exists a switching vertex u w.r.t. C such that if u is switched, then $\alpha_C(G_k) \geq \frac{1}{2}$.*

Proof. Suppose there does not exist a switching vertex. Then,

$$\alpha_C(G_k) = \frac{1}{2|E_k|} \sum_{u \in V} \max\{|N_S(v)|, |N_{\bar{S}}(v)|\} \geq \frac{1}{2|E_k|} \sum_{v \in V} \frac{1}{2} deg_k(v) = \frac{1}{4|E_k|} 2|E_k| = \frac{1}{2}.$$

clearly contradicting our assumption that $\alpha_C(G_k) < \frac{1}{2}$. If a switching vertex u is switched, then the size of $C's$ cut set increases by at least 1 so that $\alpha_C(G_k) \geq \frac{1}{2}$. ◀

Given the count of a vertex's neighbors in S and $\bar{S}$, it can be decided whether it is switching or not. Maintaining these neighbor counts is necessary to determine a vertex to switch. However, testing all vertices whether they are switching is costly. In the next section we show how to efficiently maintain a set of switching vertices. The following theorem rules out the possibility of using end points of the updated edge as switching vertices. A proof can be found in the Appendix.

▶ **Theorem 9.** *Given an edge update $\{v_i, v_j\}$ to G_{k-1} for $k \geq 1$, and a $\frac{1}{2}$-respecting cut C_{k-1} maintained on G_{k-1}, a switching vertex with respect to C_{k-1} need not always be one of v_i, v_j.*

2.2 An $O(\Delta)$ worst-case update time algorithm

In this section, we give a simple fully dynamic algorithm with worst case update time $O(\Delta)$.

Data Structures. For each vertex $u \in V$ and a cut $C = (S, \bar{S})$, we maintain the following: i) $N_S(u)$: a list of neighbors of u in S, and its size $|N_S(u)|$, ii) $N_{\bar{S}}(u)$: a list of neighbors of u in $\bar{S}$ and its size $|N_{\bar{S}}(u)|$ and, iii) $flag(u)$: a bit which is 1 if $u \in S$ and -1 if $u \in \bar{S}$.

▶ **Definition 10** (Gain of a vertex). *The gain of a vertex u with respect to a cut $C = (S, \bar{S})$ and denoted by $\mathcal{G}(u)$ is given by $\mathcal{G}(u) = flag(u)(|N_S(u)| - |N_{\bar{S}}(u)|)$.*

The gain of a vertex u measures the change in the number of cut edges of C, if u is switched. Note that a vertex is switching if the gain is positive, and non-switching otherwise. The following (global) data structures are also maintained:

a. A doubly linked list $\mathcal{L}$, which stores nodes corresponding to switching vertices.

b. An array P where $P[i]$ stores the gain of v_i and a pointer. The pointer points to the node in $\mathcal{L}$ corresponding to v_i if $\mathcal{G}(v_i) > 0$ and is $NULL$ otherwise.

The head of $\mathcal{L}$, denoted by $\mathcal{L}$.head is $NULL$ if no switching vertex exists. Each node of $\mathcal{L}$ corresponding to a switching vertex v_i stores i as its value.

Algorithm. The algorithm begins with G_0, the empty graph and $C = (S, \bar{S}) = (V, \emptyset)$ on G_0. It maintains a $\frac{1}{2}$-respecting cut on G_{k-1} for any $k \geq 1$ as follows: when an edge update $\{v_i, v_j\}$ to G_{k-1} arrives, $N_S(v_i), N_{\bar{S}}(v_i), N_S(v_j), N_{\bar{S}}(v_j)$ are updated (including their sizes) along with $P[i]$ and $P[j]$. If either of v_i, v_j become switching or non-switching, $\mathcal{L}$ is appropriately modified. C is checked if it is $\frac{1}{2}$-respecting. If C ceases to be $\frac{1}{2}$-respecting then a switching vertex v_s is found by accessing the node pointed to by $\mathcal{L}.head$ which stores the value s. This node is removed from $\mathcal{L}$, v_s is switched and $P[s]$ is updated. Data structures of v_t and $P[t]$ of all neighbors v_t of v_s are modified to reflect $v_s's$ switch. Thereafter, depending on whether or not $\mathcal{G}(v_t) > 0$ in the updated cut, the node corresponding to v_t in $\mathcal{L}$ is inserted or removed. The pseudo code of the algorithm is as follows.

Algorithm 1 Delta-Dynamic Max-Cut($G_{k-1}, \{v_i, v_j\}, C = (S, \bar{S})$).

```
Update N_S(v_i), N_S(v_j), N_S̄(v_j), N_S̄(v_j), α_C(G_k), P[i], P[j].
for v_t ∈ {v_i, v_j} do
    Add(remove) the node corresponding to v_t in 𝓛 if v_t becomes switching(non-switching).
end for
if α_C(G_k) < 1/2 then
    v_s ← 𝓛.head. Remove v_s from L.
    Switch v_s and update C, flag(v_s), N_S(v_s), N_S̄(v_s), P[s].
    for v_t ∈ N_S(v_s) ∪ N_S̄(v_s) do
        Update N_S(v_t) and N_S̄(v_t) as appropriate.
        Add(remove) the node corresponding to v_t in 𝓛 if v_t becomes switching(non-
    switching) and update P[t].
    end for
end if
return v_s.
```

Running Time. Updates to data structures of v_i, v_j and $P[i], P[j]$ take constant time. Inserting or removing a node from $\mathcal{L}$ also takes constant time. Switching v_s in the case when C is no longer $\frac{1}{2}$-respecting takes time proportional to updating all its neighbors' data structures, their corresponding entries in P and their corresponding nodes in $\mathcal{L}$. This takes $O(\Delta)$ time. Theorem 3 follows.

3 A fully dynamic distributed algorithm

In this section, we present the algorithm of Theorem 4. Dynamic distributed algorithms have been well studied in the past [36, 16], and techniques to design sequential fully dynamic algorithms are often applicable in designing their distributed counterparts. As an example, for the maximal independent set problem the distributed implementation of the dynamic sequential algorithm of Assadi et al.[3] improves on the dynamic distributed algorithm of Censor-Hillel et al. [10]. This is often easier for problems in which only the neighborhood of vertices incident to an update needs to be examined to restore the maintained property. For MAX-CUT, it may not always be the case that endpoints of the update edge can be switched to maintain a $\frac{1}{2}$-approximate cut by Theorem 9. Nevertheless, we show how to use the $O(\Delta)$ update time algorithm to get an efficient fully dynamic distributed algorithm in the MPC model.

In the model we consider, there are n machines $M_1, ..., M_n$ each corresponding to vertices $v_1, ..., v_n$ respectively. Given a graph $G = (V, E)$ where $n = |V|$ and $m = |E|$, each machine M_i initially stores a list of neighbors of v_i in addition to storing $f(v)$ and $R \subseteq [k]$ to run the static distributed algorithm obtained by pairwise independence (see Section 1.1) and obtain an initial $\frac{1}{2}$-respecting cut $C = (S, \bar{S})$ on G. For any $i, j \in [n]$ we say that machine M_i is a neighbor of M_j if $(v_i, v_j) \in E$. We let M_n be the coordinator machine which stores the position of any vertex $v \in V$ in C, i.e. whether $v \in S$ or $\bar{S}$. Given the initial cut C, each machine M_i maintains whether v_i is a switching vertex w.r.t. C or not. This can be done in a single round and $O(m)$ total communication–every machine simply sends the position of its corresponding vertex in C to all its neighbors. We also ensure that the coordinator M_n maintains the list of all switching vertices w.r.t C, the total number of edges in the graph and the size of $C's$ cut-set. After this initial preprocessing which takes $O(1)$ rounds and $O(n^2 \log n)$ bits of communication, the information $f(v)$ and R stored by all machines can be discarded.

We now describe the update algorithm. Whenever an update $\{v_i, v_j\}$ is made to G, machines M_i and M_j are informed and thereafter, they update their list of neighbors. Both M_i and M_j inform the coordinator M_n of the update in addition to informing whether v_i and v_j become switching w.r.t the maintained cut C. This allows M_n to update m, size of the cut-set C and the set of switching vertices. If C ceases to be $\frac{1}{2}$-respecting, M_n selects an arbitrary switching vertex, v_s and informs M_s. Thereafter, M_s updates its local data structures to reflect the switch and informs all its neighbors to reflect the switch. If any neighbor v_k of v_s becomes a switching vertex w.r.t the updated cut C, M_k informs the coordinator M_n, after which M_n updates the list of switching vertices.

This takes $O(1)$ rounds, $O(\Delta)$ total communication per round and at most one adjustment to C after any edge update. Theorem 4 follows.

4 Achieving sublinear (in m) update time

In this section, we present an $O(m^{1/2})$ amortized update algorithm which improves on the $O(\Delta)$ update time algorithm for sufficiently sparse graphs having high maximum degree. The high level ideas involve: i) partitioning the graph G into induced subgraphs G_1 and G_2 on V_{low} and V_{high} respectively where V_{low} and V_{high} are sets of low and high degree vertices respectively, ii) combining $\frac{1}{2}$-respecting cuts C_1 and C_2 on G_1 and G_2 respectively which are maintained using the algorithm of Theorem 3 and, iii) *selectively* updating data structures. The latter idea is crucial to reduce the update time. When a high degree vertex $v \in V_{high}$ switches w.r.t. the cut C_2, data structures of only its neighbors in V_{high} are updated leading to stale information in data structures of its neighbors in V_{low}. A similar idea was used in the fully dynamic algorithm for the maximal independent set problem [3]. We show that lazy updating of low degree vertex data structures is sufficient for our purpose and re-build G_1 and G_2 after sufficiently many updates which leads to $O(m^{1/2})$ amortized update time. Given $\frac{1}{2}$-respecting cuts on any vertex disjoint induced subgraphs of G, we first show that they can be combined to give a $\frac{1}{2}$-respecting on G.

▶ **Theorem 11** (Cut combining). *Let $G = (V, E)$ be any graph and $C_1 = (S, \bar{S})$ and $C_2 = (T, \bar{T})$ be $\frac{1}{2}$-respecting cuts with respect to the vertex disjoint induced subgraphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ of G such that $S \cup \bar{S} = V_1$, $T \cup \bar{T} = V_2$ and $V_1 \cup V_2 = V$. Then one of the following is a $\frac{1}{2}$-respecting cut C of G:*

i) $(S \cup T, \bar{S} \cup \bar{T})$

ii) $(S \cup \bar{T}, \bar{S} \cup T)$.

A formal proof of Theorem 11 is omitted for the sake of brevity but it follows by noting that cut-edges of C_1 and C_2 remain cut edges in both cuts considered in i) and ii), and the cut-set of one of the cuts in i) and ii) must contain half of the remaining edges.

Data Structures. For any $U, W \subseteq V$, let $E(U, W)$ be the set of edges having one endpoint in U and the other in W. To determine C, the following edge counts are maintained: $|E(S,T)|, |E(S,\bar{T})|, |E(\bar{S},T)|, |E(\bar{S},\bar{T})|$. If $|E(S,\bar{T})| + |E(\bar{S},T)| \geq |E(S,T)| + |E(\bar{S},\bar{T})|$, then $C = (S \cup T, \bar{S} \cup \bar{T})$, else we take $C = (S \cup \bar{T}, \bar{S} \cup T)$. Let $N_U(v)$ denote the list of neighbors of v in $U \subseteq V$. In addition to data structures required by the algorithm of Theorem 3, every vertex $v \in V_{low}$ maintains neighbor counts $N_T(v), N_{\bar{T}}(v)$ and every vertex $v \in V_{high}$ maintains neighbor counts $N_S(v), N_{\bar{S}}(v)$. For any subset $U, W \subseteq V$ s.t. $U \in \{S, \bar{S}\}$ and $W \in \{T, \bar{T}\}$, note that the edge count $|E(U,W)| = \sum_{u \in U} |N_W(u)|$.

The main challenge is to correctly maintain these edge counts without updating all the neighbors of a high degree vertex which switches w.r.t C_2. These edge counts change if i) an edge update (v_i, v_j) is encountered and/or ii) a vertex switches w.r.t. either C_1 or C_2. Our update algorithm switches at most a single vertex w.r.t C_1 or C_2 and maintains neighbor counts of high degree vertices accurately at any given time. Combined with recomputing neighbor counts of low degree vertices *only* when they switch, this is sufficient to maintain edge counts correctly at any given time.

Algorithm. The algorithm consists of phases. The k^{th} phase for $k \geq 1$ begins with the graph G containing m_k edges and $\frac{1}{2}$-respecting cuts C_1 and C_2 on the induced subgraphs G_1 and G_2 respectively. Here, G_1 and G_2 are induced subgraphs on $V_{low} = \{v \in V | deg(v) \leq m_k^{1/2}\}$ and $V_{high} = V \backslash V_{low}$ respectively. We assume that the first phase starts with a single edge, i.e. $m_1 = 1$. The k^{th} phase consists of $m_k^{1/2}$ updates after which a new phase corresponding to the new value of m_k begins. Thereafter, all data structures are reinitialized and $\frac{1}{2}$-respecting cuts are computed for G_1 and G_2 (under the new value of m_k). The total time taken to reinitialize a phase is $O(m_k)$, leading to $O(m_k^{1/2})$ amortized update time.

Note that the number of high degree vertices for any phase beginning with m_k edges is bounded by $|V_{high}| = O(m_k)/\Omega(m_k^{1/2}) = O(m_k^{1/2})$. Let $\{v_i, v_j\}$ be an edge update during the k^{th} phase for $k \geq 1$. Then,

1. if $v_i \in V_{low}, v_j \in V_{high}$: One of the lists $N_T(v_i), N_{\bar{T}}(v_i)$ and one of $N_S(v_j), N_{\bar{S}}(v_j)$ is updated. Additionally, one of the edge counts $|E(S,T)|$, $|E(S,\bar{T})|$, $|E(\bar{S},T)|$, $|E(\bar{S},\bar{T})|$ depending on the position of v_i and v_j in C_1 and C_2 respectively, is updated.
2. if $v_i, v_j \in V_{low}$: the algorithm of Theorem 3 is used to restore C_1. Let u be a vertex which is switched w.r.t C_1. All data structures of high degree neighbors of $u \in N_T(u) \cup N_{\bar{T}}(u)$ are updated. Moreover, u recomputes the lists of its high degree neighbors $N_T(u), N_{\bar{T}}(u)$. The edge counts $|E(S,T)|$, $|E(S,\bar{T})|$, $|E(\bar{S},T)|$, $|E(\bar{S},\bar{T})|$ are updated.
3. if $v_i, v_j \in V_{high}$: the algorithm of Theorem 3 is used to restore C_2. The edge counts $|E(S,T)|$, $|E(S,\bar{T})|$, $|E(\bar{S},T)|$, $|E(\bar{S},\bar{T})|$ are updated.

The pseudo code of the update algorithm is as follows.

Algorithm 2 Sublinear Max-Cut $(\{v_i, v_j\}, C_1 = (S, \bar{S}), C_2 = (T, \bar{T}))$.

if $v_i \in V_{low}$ and $v_j \in V_{high}$ **then**
  Update $|E(S,T)|, |E(S,\bar{T})|, |E(\bar{S},T)|, |E(\bar{S},\bar{T})|, N_T(v_i), N_{\bar{T}}(v_i), N_S(v_j), N_{\bar{S}}(v_j)$.
else
  if $v_i, v_j \in V_{low}$ **then**
    $u \leftarrow$ Delta-Dynamic Max-Cut$(G_1, \{v_i, v_j\}, C_1)$.
    for $w \in N_T(u) \cup N_{\bar{T}}(u)$ **do**
      Update $N_S(w)$, $N_{\bar{S}}(w)$ to reflect the new position of u in the cut $(S, \bar{S})$.
    end for
    Update $N_T(u)$, $N_{\bar{T}}(u)$, $|E(S,T)|$, $|E(S,\bar{T})|$, $|E(\bar{S},T)|$, $|E(\bar{S},\bar{T})|$.
  end if
  if $v_i, v_j \in V_{high}$ **then**
    $u \leftarrow$ Delta-Dynamic Max-Cut$(G_2, \{v_i, v_j\}, C_2)$.
    Update $|E(S,T)|$, $|E(S,\bar{T})|$, $|E(\bar{S},T)|$, $|E(\bar{S},\bar{T})|$.
  end if
end if

Running Time. If an update $\{v_i, v_j\}$ is such that $v_i \in V_{low}, v_j \in V_{high}$, the update time is $O(1)$.

If $v_i, v_j \in V_{low}$ the call to the $O(\Delta)$ update time algorithm takes time $O(m_k^{1/2})$ since any vertex in V_{low} has degree at most $2m_k^{1/2} = O(m_k^{1/2})$ throughout the phase, by definition. Updating the list of neighbors of the switched vertex u, and updating the data structures of $u's$ neighbors takes $O(m_k^{1/2})$ time. Updating edge counts takes constant time since they are incremented or decremented by constants which can be determined from the size of neighbor lists of u.

If $v_i, v_j \in V_{high}$: the call to the $O(\Delta)$ update time algorithm takes time $O(m_k^{1/2})$ since $|V_{high}| = O(m_k^{1/2})$. As in the second case, updating edge counts takes constant time.

Thus, the time taken to handle an edge update during a phase beginning with m_k edges is $O(m_k^{1/2})$. Since the amortized cost of re-initialization is $O(m_k^{1/2})$, this gives an $O(m^{1/2})$ amortized update time algorithm where m denotes the maximum number of edges in G during an arbitrary sequence sequence of updates. Theorem 5 follows. A proof of correctness can be found in the Appendix.

5 Achieving sublinear (in n) worst case update time

In this section we give a randomized algorithm which exactly maintains a $\frac{1}{2}$-respecting cut and takes $\tilde{O}(n^{2/3})$ worst case update time w.h.p. We obtain the result by first designing an algorithm with $O(n^{2/3})$ *expected* worst-case update time. Then, we apply the probability amplification result in [7] which gives a $\tilde{O}(n^{2/3})$ worst-case update time algorithm w.h.p.

The high level idea of our algorithm is to use cut-combining idea on k vertex disjoint subgraphs $G_1, G_2, ..., G_k$ induced by a random k-partition of V denoted by $(V_1, V_2, ..., V_k)$. The random partition is constructed using the oracle described in Section 1.2 such that $\bigcup_{i=1}^{k} V_i = V$ and $|V_1| = |V_2| = ... = |V_{k-1}| = \lceil n/k \rceil$, $|V_k| = n - (k-1)\lceil n/k \rceil$. On each subgraph G_i induced by V_i, a $\frac{1}{2}$ respecting cut $C_i = (S_i, \bar{S}_i)$ (where $\bar{S}_i = V_i \setminus S_i$) is dynamically maintained using the algorithm of Theorem 3. We now describe the data structures and the update algorithm.

Data structures. In addition to data structures required by the $O(\Delta)$-update time algorithm, we maintain: i) For each vertex $v \in V$, lists of its neighbors in each S_i, (denoted by $N_{S_i}(v)$) and $\bar{S}_i$ (denoted by $N_{\bar{S}_i}(v)$) for all $1 \le i \le k$ and, ii) For all $1 \le i, j \le k$, the edge counts $|E(S_i, S_j)|$, $|E(S_i, \bar{S}_j)|$ $|E(\bar{S}_i, S_j)|$, $|E(\bar{S}_i, \bar{S}_j)|$ for a total of $\binom{2k}{2} = O(k^2)$ counts. The edge counts can be maintained using the size of neighbor lists maintained for each vertex.

Algorithm.
Cut combining: We first describe how to combine $\frac{1}{2}$-approximate cuts C_i on G_i for $1 \le i \le k$ to get a $\frac{1}{2}$-approximate cut C, on G. Initially, $C = (S_1, \bar{S}_1)$. Whenever considering cut $C_i = (S_i, \bar{S}_i)$ for $2 \le i \le k$ to combine with C, the edge counts $|E(S_i, S_j)|$, $|E(S_i, \bar{S}_j)|$, $|E(\bar{S}_i, S_j)|$, $|E(\bar{S}_i, \bar{S}_j)|$, for $1 \le j \le i-1$ are used to compute the edge counts $|E(S, S_i)|$, $|E(S, \bar{S}_i)|$, $|E(\bar{S}, S_i)|$, $|E(\bar{S}, \bar{S}_i)|$. Depending on the combination which maximizes $|E(S, \bar{S})|$, either S_i (resp. $\bar{S}_i$) is added to S (resp. $\bar{S}$) or S_i (resp. $\bar{S}_i$) is added to $\bar{S}$ (resp. S). Computing the edge counts takes $O(k)$ time, yielding $O(k^2)$ time to compute C.
Update algorithm: Let $\{v_i, v_j\}$ be an edge update. Then,

1. if $v_i \in V_p$ and $v_j \in V_q$ s.t. $p \neq q$: Only the lists $N_{S_q}(v_i), N_{\bar{S}_q}(v_i)$, $N_{S_p}(v_j), N_{\bar{S}_p}(v_j)$ and edge counts $|E(S_p, S_q)|, |E(S_p, \bar{S}_q)|, |E(\bar{S}_p, S_q)|, |E(\bar{S}_p, \bar{S}_q)|$ are updated which takes $O(1)$ time.
2. if $v_i, v_j \in V_p$ for some p: the cut C_p is updated using the $O(\Delta)$ update time algorithm. Let u be the switched vertex w.r.t C_p. The lists $N_{S_p}(w), N_{\bar{S}_p}(w)$ of all neighbors w of u are updated to reflect $u's$ switch. For all $1 \le q \le k$ such that $N_{S_q}(u) \cup N_{\bar{S}_q}(u) \neq \emptyset$, edge counts of the form $|E(S_p, S_q)|$, $|E(S_p, \bar{S}_q)|$, $|E(\bar{S}_p, S_q)|$, $|E(\bar{S}_p, \bar{S}_q)|$ are also updated. This can be done by using the values of $|N_{S_q}(u)|$ and $|N_{\bar{S}_q}(u)|$.

Following this, the cuts $C_1, ..., C_k$ are combined to yield C. The pseudo code of the update algorithm is as follows.
Note that the only information required to determine how to combine the cut $(S_t, \bar{S}_t)$ with $(S, \bar{S})$ in each iteration of the for loop is the position of all $S_i, \bar{S}_i$ for all $i \le t-1$ in $(S, \bar{S})$. Thus, computing the edge counts $|E(S \cup S_t, \bar{S} \cup \bar{S}_t)|$, $|E(S \cup \bar{S}_t, \bar{S} \cup S_t)|$ can be done in $O(k)$ time, and lines 14 and 16 of Algorithm 3 do not need to be explicitly implemented.

Running Time. For the case when $v_i \in V_p$ and $v_j \in V_q$ s.t. $p \neq q$ updating the edge counts takes constant time. However, the combining cost is incurred. This is because a single update can possibly cause the cuts to combine differently in order to maintain a $\frac{1}{2}$-respecting cut on G.

Algorithm 3 Randomized Sublinear MAX-CUT ($\{v_i, v_j\}, G_1, ..., G_k, C_1, ..., C_k$).

if $v_i \in V_p, v_j \in V_q$ s.t. $p \neq q$ **then**
  Update $N_{S_q}(v_i), N_{\bar{S_q}}(v_i), N_{S_p}(v_j), N_{\bar{S_p}}(v_j)$.
  Update $|E(S_p, S_q)|, |E(\bar{S}_p, S_q)|, |E(S_p, \bar{S}_q)|, |E(\bar{S}_p, \bar{S}_q)|$ appropriately.
else
  $u \leftarrow$ Delta-Dynamic Max-Cut$(G_p, \{v_i, v_j\}, C_p)$.
  for all neighbors v of u where $v \in V_r$ for any $1 \leq r \leq k$ **do**
    Update $N_{S_r}(u), N_{\bar{S_r}}(u), N_{S_p}(v), N_{\bar{S_p}}(v)$.
    Update $|E(S_p, S_r)|, |E(\bar{S}_p, S_r)|, |E(S_p, \bar{S}_r)|, |E(\bar{S}_p, \bar{S}_r)|$ appropriately.
  end for
end if
$S = S_1, \bar{S} = \bar{S}_1$.
for $t = 2, ..., k$ **do**
  if $|E(S \cup S_t, \bar{S} \cup \bar{S}_t)| \geq |E(S \cup \bar{S}_t, \bar{S} \cup S_t)|$ **then**
    $S = S \cup S_t$, $\bar{S} = \bar{S} \cup \bar{S}_t$.
  else
    $S = S \cup \bar{S}_t$, $\bar{S} = \bar{S} \cup S_t$.
  end if
end for

For the case when $v_i, v_j \in V_p$ for some p, the algorithm of Theorem 3 takes $O(n/k)$ time. Let u be the switched vertex w.r.t. C_p. Updating the neighbor lists of all neighbors of u takes $O(\Delta)$ time. Thus, the update time in this case is $O(\Delta + \frac{n}{k} + k^2) = O(\Delta + k^2)$.

▶ **Lemma 12.** *The running time of the update algorithm is $O(\frac{\Delta}{k} + k^2)$. With $k = \Theta(n^{1/3})$, this yields $O(n^{2/3})$ expected worst-case update time.*

Proof. Let $\{v_i, v_j\}$ be an edge update. The probability that this update is of the second type, i.e. $v_i, v_j \in V_p$ for some $p \in [k]$ is at most $1/k$. The expected update time, denoted by $E[T(n, k)]$ can be written as,

$$\begin{aligned}
E[T(n,k)] &= \Pr[v_i, v_j \in V_p]O(\Delta + k^2) + \Pr[v_i \in V_p, v_j \in V_q, p \neq q]O(k^2) \\
&= \Pr[v_i, v_j \in V_p]O(\Delta + k^2) + (1 - \Pr[v_i, v_j \in V_p,])O(k^2) \\
&= \frac{1}{k}O(\Delta) + O(k^2) \\
&= O(\frac{\Delta}{k} + k^2) \\
&= O(\frac{n}{k} + k^2).
\end{aligned}$$

The value of k which minimizes $E[T(n, k)]$ is $\Theta(n^{1/3})$ yielding $O(n^{2/3})$ expected worst case update time. ◀

Bernstein et al. [7] give a general technique to convert a fully dynamic data structure with expected worst-case update time to one with a worst-case update time with high probability. See [7] for technical details. By using their technique as a black-box, we convert our randomized algorithm described in this section taking $O(n^{2/3})$ expected worst-case update time to one taking $O(n^{2/3} \log^2(n)) = \tilde{O}(n^{2/3})$ update time with high probability. Theorem 6 follows.

6 Conclusion

The following open problems arise from our work. First, it would be interesting to improve on the algorithm in Theorem 5 to get a better update time in the *worst-case*. Second, the Algorithm in Theorem 6 works only for an oblivious adversary, and it would be interesting to design a randomized worst-case algorithm with better update time which works against an adaptive adversary.

We believe that ideas from our fully dynamic distributed MPC algorithm may be useful in other models such as the ones considered in [27, 39]. We observe that our dynamic algorithm for MPC can be implemented in the Congested-Clique model. Moreover, we believe that a dynamic MPC algorithm to maintain a $\frac{1}{2}$-respecting cut using only sublinear (in n) memory per machine (in contrast to $\Omega(n)$ memory as in the algorithm of Theorem 4) may be possible without a blow up in the round, adjustment or message complexity. A natural open question is whether there exists a deterministic fully dynamic algorithm with $o(\Delta)$ round complexity and $O(1)$ adjustment and message complexity to preserve a $\frac{1}{2}$-respecting cut in the $\mathcal{CONGEST}$ model. This may necessitate new techniques and lead to interesting connections to other fundamental problems studied in the distributed computing and dynamic algorithms literature.

References

1 Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 574–583, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591805.

2 Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *J. ACM*, 63(2):12:1–12:35, May 2016. doi:10.1145/2837020.

3 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 815–826, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188922.

4 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 1919–1936, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3310435.3310551.

5 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in o (log n) update time. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 383–392, October 2011. doi:10.1109/FOCS.2011.89.

6 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '13, page 273–284, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2463664.2465224.

7 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918. SIAM, 2019. doi:10.1137/1.9781611975482.115.

8 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In Artur Czumaj, editor, *Proceedings of the Twenty-*

Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, pages 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.

9 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Dynamic algorithms via the primal-dual method. *Inf. Comput.*, 261(Part):219–239, 2018. doi:10.1016/j.ic.2018.02.005.

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*, PODC '16, page 217–226, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2933057.2933083.

11 Keren Censor-Hillel, Rina Levy, and Hadas Shachnai. Fast distributed approximation for max-cut. In Antonio Fernández Anta, Tomasz Jurdzinski, Miguel A. Mosteiro, and Yanyong Zhang, editors, *Algorithms for Sensor Systems - 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, volume 10718 of *Lecture Notes in Computer Science*, pages 41–56. Springer, 2017. doi:10.1007/978-3-319-72751-6_4.

12 K. C. Chang and D. H. . Du. Efficient algorithms for layer assignment problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(1):67–78, 1987.

13 Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending grothendieck's inequality. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, page 54–60, USA, 2004. IEEE Computer Society. doi:10.1109/FOCS.2004.39.

14 Kwan-Wu Chin, Sieteng Soh, and Chen Meng. Novel scheduling algorithms for concurrent transmit/receive wireless mesh networks. *Computer Networks*, 56:1200–1214, March 2012. doi:10.1016/j.comnet.2011.12.001.

15 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond, 2019. arXiv:1910.08025.

16 Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. Distributed computation in dynamic networks via random walks. *Theor. Comput. Sci.*, 581(C):45–66, May 2015. doi:10.1016/j.tcs.2015.02.044.

17 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. doi:10.1145/1327452.1327492.

18 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 Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 1300–1319, USA, 2020. Society for Industrial and Applied Mathematics.

19 Michel X. Goemans and David P. Williamson. .879-approximation algorithms for MAX CUT and MAX 2sat. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 422–431, 1994. doi:10.1145/195058.195216.

20 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Proceedings of the 22nd International Conference on Algorithms and Computation*, ISAAC'11, page 374–383, Berlin, Heidelberg, 2011. Springer-Verlag. doi:10.1007/978-3-642-25591-5_39.

21 Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski. A tree structure for dynamic facility location. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 39:1–39:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.39.

22 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of*

Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017, pages 537–550. ACM, 2017. doi:10.1145/3055399.3055493.

23 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018. arXiv:1804.01823.

24 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. doi:10.1145/320211.320215.

25 Niklas Hjuler, Giuseppe F. Italiano, Nikos Parotsidis, and David Saulpic. Dominating sets and connected dominating sets in dynamic graphs. In *STACS*, 2019.

26 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48:723–760, July 2001. doi:10.1145/276698.276715.

27 Giuseppe F. Italiano, Silvio Lattanzi, Vahab S. Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model. In *Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '19, page 49–58, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3323165.3323202.

28 David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, December 1974. doi:10.1016/S0022-0000(74)80044-9.

29 Satyen Kale and C. Seshadhri. Combinatorial approximation algorithms for maxcut using random walks. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 367–388. Tsinghua University Press, 2011. URL: http://conference.iiis.tsinghua.edu.cn/ICS2011/content/papers/20.html.

30 Bruce Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1131–1142, January 2013. doi:10.1137/1.9781611973105.81.

31 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*, SODA '10, page 938–948, USA, 2010. Society for Industrial and Applied Mathematics.

32 Richard M. Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming*, 1972.

33 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Deterministic worst case dynamic connectivity: Simpler and faster. *CoRR*, abs/1507.05944, 2015. arXiv:1507.05944.

34 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 146–154, 2004. doi:10.1109/FOCS.2004.49.

35 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, April 2007. doi:10.1137/S0097539705447372.

36 Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, page 513–522, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806760.

37 M. Luby. Removing randomness in parallel computation without a processor penalty. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 162–173, 1988.

38 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016. doi:10.1145/2700206.

39 Krzysztof Nowicki and Krzysztof Onak. Dynamic graph algorithms with batch updates in the massively parallel computation model, 2020. arXiv:2002.07800.

40 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464. ACM, 2010. doi:10.1145/1806689.1806753.

41 M. Preissmann and Andras Sebo. Optimal cuts in graphs and statistical mechanics. *Mathematical and Computer Modelling - MATH COMPUT MODELLING*, 26:1–11, October 1997. doi:10.1016/S0895-7177(97)00195-7.

42 Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, January 1981. doi:10.1145/322234.322235.

43 Shay Solomon. Fully dynamic maximal matching in constant update time. *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325–334, 2016.

44 José A. Soto. Improved analysis of a max-cut algorithm based on spectral partitioning. *SIAM J. Discret. Math.*, 29(1):259–268, 2015. doi:10.1137/14099098X.

45 Mikkel Thorup. Fully-dynamic min-cut. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 224–230, 2001. doi:10.1145/380752.380804.

46 L. Trevisan. Max cut and the smallest eigenvalue. *SIAM Journal on Computing*, 41(6):1769–1786, 2012. doi:10.1137/090773714.

47 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time, 2016. arXiv:1611.02864.

48 Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, USA, 2010. USENIX Association.

7 Appendix

7.1 Proof of Observation 2

Proof. The high level idea is to partition the update sequence into phases consisting of $O(\epsilon m)$ updates and spreading the time to recompute a t-respecting (resp., t-approximate) cut using $\mathcal{A}_S$ over any phase. Let P_i denote phase i, G_{P_i} the graph at the beginning of phase i and m_i the number of edges in G_{P_i}. We let $m_i = m$ so that phases P_{i+1} and P_{i+2} begin after $\frac{\epsilon m}{2}$ and ϵm updates have been made to G_{P_i}, respectively. Algorithm $\mathcal{A}_S$ is used to compute a t-respecting (resp., t-approximate) cut C_{P_i} on G_{P_i} by spending $T(m,n)$ time spread over updates between phase P_i and P_{i+1}, and C_{P_i} is used to answer all queries between phase P_{i+1} and P_{i+2}. This takes $\frac{2T(m,n)}{\epsilon m} = O(\frac{T(m,n)}{\epsilon m})$ worst-case update time where C_{P_i} is a $(t-\epsilon)$-respecting (resp., t-approximate) cut until phase P_{i+2} begins. Moreover, after P_{i+1} begins, $\mathcal{A}_S$ is used to compute a t-respecting (resp., t-approximate) cut $C_{P_{i+1}}$ on $G_{P_{i+1}}$ by spending $T(m_{i+1},n)$ time spread over updates between phase P_{i+1} and P_{i+2}, yielding a worst-case update time of $\frac{2T(m_{i+1},n)}{\epsilon m} \leq \frac{2T(m(1+\epsilon/2),n)}{\epsilon m} = O(\frac{T(m,n)}{\epsilon m})$. Thus, the total worst-case update time is bounded by $O(\frac{T(m,n)}{\epsilon m})$. ◀

7.2 Endpoints of an updated edge may not be switching

▶ **Theorem 9.** *Given an edge update $\{v_i, v_j\}$ to G_{k-1} for $k \geq 1$, and a $\frac{1}{2}$-respecting cut C_{k-1} maintained on G_{k-1}, a switching vertex with respect to C_{k-1} need not always be one of v_i, v_j.*

Proof. We refer to Figures 7.1 and 7.2 for the sake of illustration. Let $V = \{v_1, ..., v_9\}$ be the set of vertices such that $S = V, \bar{S} = \emptyset$. Consider the following sequence of edge insertions $\{v_1, v_6\}, \{v_1, v_7\}, \{v_2, v_7\}, \{v_3, v_7\}, \{v_3, v_8\}, \{v_3, v_9\}, \{v_4, v_9\}, \{v_5, v_8\}$ which leads to

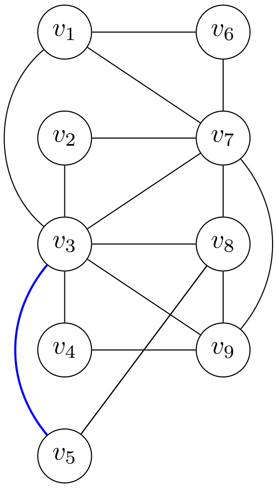

Figure 7.1 $S = \{v_1, .., v_5\}, \bar{S} = \{v_6, .., v_9\}$. After $\{v_3, v_5\}$ is added, v_3 switches.

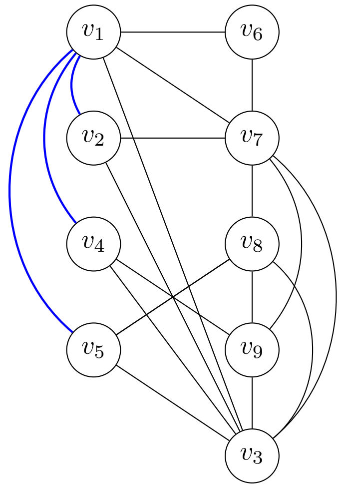

Figure 7.2 After v_3 switches and edges $\{v_1, v_2\}, \{v_1, v_4\}, \{v_1, v_5\}$ are added, none of v_1, v_2, v_4, v_5 are switching, yet the cut ceases to be $\frac{1}{2}$ respecting.

v_6, v_7, v_8, v_9 moving to $\bar{S}$ in that order, as a result. Next, consider the following non-cut edge insertions in no particular order: $\{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_6, v_7\}, \{v_7, v_8\}, \{v_8, v_9\}, \{v_7, v_9\}$. The latter set of edge insertions does not make any vertex switching, After the edge $\{v_3, v_5\}$ is added v_3 switches to $\bar{S}$. Now consider the insertion of non-cut edges $\{v_1, v_2\}, \{v_1, v_4\}, \{v_1, v_5\}$ so that none of their endpoints namely v_1, v_2, v_4, v_5 become switching. But, $(S, \bar{S})$ is no longer $\frac{1}{2}$-respecting. ◀

7.3 On the sublinear (in m) update time algorithm

7.3.1 Proof of correctness

▶ **Lemma 13.** *Algorithm 2 correctly maintains the edge counts $|E(S,T)|, |E(S,\bar{T})|, |E(\bar{S},T)|, |E(\bar{S},\bar{T})|$ where $C_1 = (S, \bar{S}), C_2 = (T, \bar{T})$.*

Proof. Assume that the edge counts $(|E(S,T)|, |E(S,\bar{T})|, |E(\bar{S},T)|, |E(\bar{S},\bar{T})|)$ are accurate before Algorithm 2 is executed to handle the edge update $\{v_i, v_j\}$. For $v_i \in V_{low}$ and $v_j \in V_{high}$ let $X \in \{S, \bar{S}\}$, $Y \in \{T, \bar{T}\}$ be such that $v_i \in X, v_j \in Y$. If $\{v_i, v_j\}$ is an edge insertion, then v_i is added to $N_X(v_j)$, v_j to $N_Y(v_i)$ and $|E(X,Y)|$ is increased by 1. On the other hand, if $\{v_i, v_j\}$ is an edge deletion, v_i is removed from $N_X(v_j)$, v_j from $N_Y(v_i)$ and $|E(X,Y)|$ is decremented by 1. Thus, the edge counts are correctly updated in this case.

In the case when $v_i, v_j \in V_{low}$, Algorithm 1 is called in order to handle the edge update with respect to the induced subgraph G_1. Let $u \in V_{low}$ be a switched vertex and let $X, \bar{X} \in \{S, \bar{S}\}$ be such that $u \in X$ moves to $\bar{X}$ after the switch. Now, u may no longer have an accurate count of its neighbors in T and $\bar{T}$ since when high degree neighbors of u possibly switch in previous updates, the data structures of u namely $N_T(u), N_{\bar{T}}(u)$ are not modified. Thus, lists $N_T(u), N_{\bar{T}}(u)$ are updated and for all high degree neighbors w of u, $N_X(w), N_{\bar{X}}(w)$ are also updated to reflect u's switch. Since u switched from X to $\bar{X}$, the sizes of lists $N_X(w), N_{\bar{X}}(w)$ are modified appropriately. For all neighbors $w \in V_{low}$ of u, their data structures due to $u's$ switch to $\bar{X}$ are already updated in the call to Algorithm 1. Since $u's$ neighbor lists are up-to-date, the counts $|E(X,T)|, |E(X,\bar{T})|, |E(\bar{X},T)|, |E(\bar{X},\bar{T})|$ are correctly updated.

For the case when $v_i, v_j \in V_{high}$, Algorithm 1 is called in order to handle the edge update with respect to the induced subgraph G_2. Let $u \in V_{high}$ be a vertex which switches and let $Y, \bar{Y} \in \{T, \bar{T}\}$ be such that $u \in Y$ before the update and switches to $\bar{Y}$. Vertices in V_{high} are updated to reflect the switch of u with respect to the cut $(T, \bar{T})$ during the call to Algorithm 1. Since u is a high degree vertex, the neighbor lists $N_S(u), N_{\bar{S}}(u)$ are always up-to-date. Thus, the edge counts $|E(X,T)|, |E(X,\bar{T})|, |E(\bar{X},T)|, |E(\bar{X},\bar{T})|$ are correctly updated. ◀