Simple dynamic algorithms for Maximal Independent Set, Maximum Flow and Maximum Matching

Gupta, Manoj

Society for Industrial and Applied Mathematics
2021-01-07

Gupta , M & Khan , S 2021 , Simple dynamic algorithms for Maximal Independent Set, Maximum Flow and Maximum Matching . in V King & H Viet Le (eds) , Proceedings Symposium on Simplicity in Algorithms (SOSA) . Society for Industrial and Applied Mathematics , pp. 86-91 , Symposium on Simplicity in Algorithms , 11/01/2021 . https://doi.org/10.1137/1.978161197

http://hdl.handle.net/10138/340008
https://doi.org/10.1137/1.9781611976496.10

unspecified
acceptedVersion

Simple dynamic algorithms for Maximal Independent Set and other problems

Manoj Gupta
IIT Gandhinagar,
Gandhinagar, India
gmanoj@iitgn.ac.in

Shahbaz Khan*
Faculty of Computer Science,
University of Vienna, Austria
shahbaz.khan@univie.ac.at

Abstract

Most graphs in real life keep changing with time. These changes can be in the form of insertion or deletion of edges or vertices. Such rapidly changing graphs motivate us to study dynamic graph algorithms. However, three important graph problems that are perhaps not sufficiently addressed in the literature include independent sets, maximum matching (exact) and maximum flows.

Maximal Independent Set (MIS) is one of the most prominently studied problems in the distributed setting. Recently, the first dynamic MIS algorithm for distributed networks was given by Censor-Hillel et al. [PODC16], requiring expected $O(1)$ amortized rounds with $O(\Delta)$ messages per update, where Δ is the maximum degree of a vertex in the graph. They suggested an open problem to maintain MIS in fully dynamic centralized setting more efficiently. Assadi et al. [STOC18] presented a deterministic centralized fully dynamic MIS algorithm requiring $O(\min\{\Delta, m^{3/4}\})$ amortized time per update. This result is quite complex involving an exhaustive case analysis. We report a surprisingly simple deterministic *centralized* algorithm which improves the amortized update time to $O(\min\{\Delta, m^{2/3}\})$.

Additionally, we present some other minor results related to dynamic MIS, Maximum Flow, and Maximum Matching. A common trait of all our results is that despite improving state of the art upper bounds or matching state of the art lower bounds, they are surprisingly simple and are analysed using simple amortization arguments. Further, they use no complicated data structures or black box algorithms for their implementation.

*This research work was supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

1 Introduction

In the last two decades, there has been a flurry of results in the area of dynamic graph algorithms. The motivation behind studying such problems is that many graphs encountered in the real world keep changing with time. These changes can be in the form of addition and/or deletion of edges and/or vertices. The aim of a dynamic graph algorithm is to report the solution of the concerned graph problem after every such update in a graph. One can trivially use the best static algorithm to compute the solution from scratch after each such update. Hence, the aim is to update the solution more efficiently as compared to the best static algorithm. Various graph problems studied in the dynamic setting include connectivity [27, 28, 31], minimum spanning tree [14, 17, 36], reachability [12, 41, 39], shortest path [43, 5, 11], matching [8, 4, 26], etc. However, three important graph problems that are perhaps not sufficiently addressed in the literature include independent sets, maximum matching (exact) and maximum flows.

For a given graph $G = (V, E)$ with n vertices and m edges where maximum degree of a vertex is Δ, a set of vertices $\mathcal{M} \subseteq V$ is called an *independent set* if no two vertices in $\mathcal{M}$ share an edge in E, i.e. $\forall x, y \in \mathcal{M}$, $(x, y) \notin E$. Computing the maximum cardinality *independent set* is known to be NP-Hard [21]. However, a simple greedy algorithm is known to compute the Maximal Independent Set (MIS) in $O(m)$ time, where MIS is any independent set $\mathcal{M}$ such that no proper superset $\mathcal{M}'$ of $\mathcal{M}$, i.e. $\mathcal{M} \subset \mathcal{M}'$, is an independent set of the graph. Note that MIS is not a good approximation of maximum cardinality independent set[1] unlike other known problems such as matching (which is a 2-approximation of maximum matching).

Fully Dynamic Maximal Independent Set

In the dynamic setting, one can trivially maintain MIS in $O(m)$ update time, by computing MIS from scratch after every update. Moreover, the *adjustment* complexity[2] of this algorithm can be $O(n)$. Until recently no non-trivial algorithm was known to maintain MIS in $o(m)$ time.

The problem of MIS has been extensively studied in the distributed setting [3, 22, 30]. The first dynamic MIS algorithm was given by Censor-Hillel et al. [9] showing the maintenance of dynamic MIS in expected $O(1)$ amortized rounds with $O(\Delta)$ messages per update, where Δ is the maximum degree of a vertex in G. This also translates to a *centralized* algorithm with update time of amortized $\Omega(\Delta)$ [2]. Recently, Assadi et al. [2] presented a deterministic *centralized* fully dynamic MIS algorithm requiring $O(\min\{\Delta, m^{3/4}\})$ amortized time and $O(1)$ amortized adjustments per edge update. Further, the update time for centralized algorithm was recently improved [37] for low arboricity graphs. In this paper, we present surprisingly simple *deterministic* algorithms to improve the results for general arboricity graphs.

Theorem 1.1 (Fully dynamic MIS (centralized))**.** *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be maintained in $O(\min\{\Delta, m^{2/3}\})$ amortized time and $O(1)$ amortized adjustments per insertion or deletion of an edge or a vertex, where Δ is the maximum degree of a vertex in G.*

Remark: A limitation of our result over Assadi et al. [2] is that it cannot be trivially extended to the distributed setting with efficient round and adjustment complexity. Trivial adaptation of our algorithm in the distributed $\mathcal{CONGEST}$ model requires $O(m^{1/3})$ amortized rounds and adjustments per update, whereas that of [2] requires $O(1)$ rounds and adjustments per update. The message complexity in both cases matches the update time of the centralized setting.

[1] Consider a star graph where a vertex s has an edge to all other vertices in V. The maximum independent set is $\mathcal{M}^* = V \setminus \{s\}$, whereas $\mathcal{M}' = \{s\}$ is a valid MIS. Here approximation factor $|\mathcal{M}^*|/|\mathcal{M}'|$ is $n - 1$.

[2] The *adjustment* complexity refers to the number of vertices that enter or leave $\mathcal{M}$ after an update.

Additionally, we present some other minor results related to dynamic MIS, Maximum Flow, and Maximum Matching as follows.

1. **Hardness of dynamic MIS and Incremental MIS**
   We first discuss the hardness of different dynamic updates for maintaining MIS and report that the hardest update for maintaining MIS is that of edge insertions. For the remaining dynamic settings (fully dynamic vertex updates and decremental edge updates), a simple algorithm [2] maintains MIS optimally, i.e. total update time is of the order of input size. Hence, in addition to fully dynamic edge updates, the only interesting dynamic setting is handling edge insertions, where we again present a very simple algorithm to prove the following.

   Theorem 1.2 (Incremental MIS)**.** *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be maintained in $O(\min\{\Delta, \sqrt{m}\})$ amortized time and $O(1)$ amortized adjustments per edge insertion, where Δ is the maximum degree of a vertex in G.*

   Remark: Both these algorithms (for incremental MIS and remaining simple updates) trivially extend to the distributed CONGEST model with $O(1)$ amortized rounds per update.

2. **Worst case bounds for dynamic MIS**
   All the previous results for dynamic MIS primarily focus on amortized guarantees. This was explained by Assadi et al. [2] by proving that the *adjustment* complexity (and hence update time) of any dynamic MIS algorithm must be $\Omega(n)$ in the worst case. Hence, in order to achieve better worst case bounds, we are required to consider a *relaxed* model. Thus, we relax the requirement to *explicitly* maintain the MIS after each update. Rather, we allow queries of the form In-Mis(v), which reports whether a vertex $v \in V$ is present in some MIS of the updated graph, ensuring the following properties. Firstly, the responses to all the queries after an update are consistent to some MIS of the updated graph. Secondly, the adjustment complexity of the algorithm is amortized $O(1)$ per update (considering only updates), or worst case $O(1)$ per update and query. Such a model have been previously studied for several problems including MIS [40, 1, 35]. In this *relaxed model* we show that

   Theorem 1.3 (Fully dynamic MIS (worst case))**.** *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be implicitly maintained under fully dynamic edge updates requiring $O(1)$ adjustments per update and query, which allows queries of the form* In-Mis(v)*, where both update and query require worst case $O(\min\{\Delta, \sqrt{m}\})$ time.*

   Remark: It trivially extends to CONGEST model with $O(1)$ rounds per update and query.

3. **Dynamic Maximum Flow and Maximum Matching**
   The maximum flow and maximum matching problems are some of the most studied combinatorial optimization problem having a lot of practical applications. Recently, Dahlgaard [10] proved conditional lower bounds for partially dynamic problems including maximum flow and maximum matching. Assuming the correctness of OMv conjecture, he proved that maintaining incremental Maximum Flow or Maximum Matching for unweighted graphs requires $\Omega(n)$ update time (even amortized). We report trivial extensions of two classical algorithms based on *augmenting paths*, namely incremental reachability algorithm [29] and blossoms algorithm [13, 19], which match these lower bounds.

   For the sake of completeness, we also report the folklore algorithms to update the maximum flow of an unweighted (unit-capacity) graph and maximum cardinality matching, in $O(m)$ time using simple reachability queries. To the best of our knowledge, it is widely known but so far it has not been a part of any literature.

2 Overview

Let the given graph be $G = (V, E)$ with n vertices and m edges, where the maximum degree of a vertex is Δ. The degree of a vertex $v \in V$ shall be denoted by $deg(v)$. We shall represent the currently maintained MIS by $\mathcal{M}$. We shall now give a brief overview of our results and the difference of our approach from the current state of the art.

A trivial static algorithm can compute an MIS of a graph in $O(m)$ time. It visits each vertex v and checks whether any of its neighbours is in $\mathcal{M}$. If no such neighbour exists, it adds v to $\mathcal{M}$, clearly taking $O(1)$ time for each edge while visiting its endpoints.

Assadi et al. [2] described a *simple* dynamic algorithm (henceforth referred as $\mathcal{A}_s$) requiring $O(\Delta)$ amortized time per update, where essentially each vertex maintains a *count* of the number of its neighbours in $\mathcal{M}$. They also described an improved algorithm requiring $O(\min\{\Delta, m^{3/4}\})$ time, which is based on a complicated case analysis. The algorithm essentially divides the vertices into four sets based on their degrees, where the *count* of all the vertices except the low degree vertices V_{low} is maintained exactly. For vertices in V_{low}, instead some *partial estimate* of *count* is maintained ignoring the high degree vertices in $\mathcal{M}$. The key difference of this improved algorithm from $\mathcal{A}_s$ is the following: Instead of adding a vertex to $\mathcal{M}$ only when none of its neighbours is in $\mathcal{M}$ (i.e. when the true value of *count* is zero), in some cases the algorithm adds a low degree vertex to $\mathcal{M}$ even when merely the *partial estimate* of *count* is zero. As a result, the algorithm may need to remove some vertices from $\mathcal{M}$ to ensure correctness of the algorithm.

Now, the simple $\mathcal{A}_s$ algorithm can be used to maintain MIS under all possible graph updates. We firstly provide a *mildly* tighter analysis of $\mathcal{A}_s$ to demonstrate what kind of dynamic settings are harder for the MIS problem. We show that $\mathcal{A}_s$ solves the problem optimally for all kinds of updates except *edge insertions*. Hence, the two dynamic settings in which the MIS problem is interesting are the fully dynamic and the incremental settings under *edge updates*. We improve the state-of-the-art for dynamic MIS in both these settings using very simple algorithms, which are essentially based on the simple $\mathcal{A}_s$ algorithm as follows.

Our *fully dynamic algorithm* processes the low degree vertices V_{low} (say having degree $\leq \Delta_c$) totally independent of the high degree vertices. Note that this simplifies the approach of [2] since the key difference of their algorithm from $\mathcal{A}_s$ is the partial independence of processing V_{low} with respect to higher degree vertices. Hence, we maintain the MIS $\mathcal{M}$ of the subgraph induced by the vertices in V_{low} irrespective of its high degree neighbours in $\mathcal{M}$. Using $\mathcal{A}_s$ the MIS of this subgraph can be maintained in $O(\Delta_c)$ amortized update time. After each update, the MIS of high degree vertices not adjacent to any low vertex in $\mathcal{M}$ (i.e. $\mathcal{M} \cap V_{low}$) can be recomputed from scratch using the trivial static algorithm. This gives a trade off since the size of the subgraph induced by the high degree vertices (and hence the time taken by the static algorithm) decreases as Δ_c is increased. Hence, choosing an appropriate Δ_c results in amortized $O(\min\{\Delta, m^{2/3}\})$ update time. Note that recomputing the MIS for high degree vertices from scratch may lead to a larger *adjustment* and *round* complexity in the distributed setting.

In the *incremental setting*, we show that $\mathcal{A}_s$ indeed takes $\Theta(\Delta)$ amortized time per update (see Appendix A). Moreover, we improve $\mathcal{A}_s$ using a simple modification which prioritizes the removal of low degree vertices from $\mathcal{M}$ instead of an arbitrary choice by $\mathcal{A}_s$. This simple modification improves the amortized update time to $O(\min\{\Delta, \sqrt{m}\})$, which is also shown to be tight (see Appendix A).

For *worst case complexity* of a fully dynamic MIS algorithm Assadi et al. [2] showed strong lower bounds, where a single update may lead to $\Theta(n)$ vertices to enter or leave $\mathcal{M}$. Hence, for achieving better *worst case* bounds our relaxed model essentially maintains merely an *independent set* explicitly rather than MIS. Thus, the MIS is completely built over several queries allowing better worst case complexity. We present a simple algorithm for maintaining fully dynamic MIS in this model requiring $O(\min\{\Delta, \sqrt{m}\})$ worst case time for update and query.

3 Dynamic MIS

Assadi et al. [2] demonstrated a simple algorithm $\mathcal{A}_s$ for maintaining fully dynamic MIS using $O(\Delta)$ amortized time per update. We first briefly describe the algorithm and its analysis, which shall be followed by a tighter analysis that can be used to argue the kind of dynamic updates for which it is harder to maintain MIS.

3.1 Simple dynamic algorithm $\mathcal{A}_s$ [2]

The following algorithm is the most natural approach to study the dynamic MIS problem. It essentially maintains the *count* of the number of neighbours of a vertex in the MIS. It is easy to see that the *count* for each vertex can be initialized in $O(m)$ time using the simple greedy algorithm for computing the MIS of the initial graph.

Now, under dynamic updates the *count* of each vertex needs to be maintained explicitly. Hence, whenever a vertex v enters or leaves $\mathcal{M}$, the *count* of its neighbours is updated in $O(deg(v))$ time. On insertion of a vertex v, the *count* of v is computed in $O(deg(v))$ time. In case this *count* is *zero*, v is added to $\mathcal{M}$. In case of deletion of a vertex v, an update is only required when $v \in \mathcal{M}$, where v is simply removed from $\mathcal{M}$. In case *count* of any neighbour of v reduces to *zero*, it is added to $\mathcal{M}$. In case of deletion of an edge (u, v), update is required only when one of them (say u) is in $\mathcal{M}$. If *count* of v reduces to *zero*, it is added to $\mathcal{M}$. Finally, on insertion of an edge (u, v), update is required only in case both are in $\mathcal{M}$, where either one of them (say v) is removed from $\mathcal{M}$. Again, if *count* of any neighbour of v reduces to *zero*, it is added to $\mathcal{M}$.

Notice that in case of each update at most *one* vertex may be removed from $\mathcal{M}$ and several vertices may be added to $\mathcal{M}$, where both addition or deletion of a vertex v takes $O(deg(v))$ time. Hence, whenever a vertex is removed from $\mathcal{M}$, $O(1)$ adjustments and $O(\Delta)$ work is charged for both this removal as well as the next insertion (if any) to $\mathcal{M}$. The initial charge required is the sum of degrees of all the vertices in $\mathcal{M}$, which is $O(m)$. As a result, fully dynamic MIS can be maintained in $O(1)$ amortized adjustments and $O(\Delta)$ amortized time per update.

Theorem 3.1 (Fully dynamic MIS [2])**.** *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be maintained in $O(1)$ amortized adjustments and $O(\Delta)$ amortized time per insertion or deletion of edge or vertex, where Δ is the maximum degree of a vertex in G.*

3.1.1 Tighter analysis

We shall now show a mildly tighter analysis of the algorithm which is again very simple and apparent from the algorithm itself. Instead of generalizing the update time to $O(\Delta)$, we show that the update time of the algorithm is $O(deg(v))$, where v is the vertex that is removed from $\mathcal{M}$ or inserted in the graph (for vertex insertions), otherwise the charged update time is $O(1)$.

This analysis is again fairly straight forward using the similar arguments as in the previous section, which stated that a vertex can be charged twice $O(\Delta)$ when it is removed from $\mathcal{M}$ to pay for the future cost of its insertion to $\mathcal{M}$. The only difference in our analysis is as follows: Since we are not charging the update with the maximum degree Δ, rather the exact degree $deg(v)$ of a vertex v, this $deg(v)$ may change between the time it was charged and when it is used. Particularly, consider a vertex v which was charged $deg(v)$ when removed from $\mathcal{M}$ or inserted in the graph. Now, on being inserted back in $\mathcal{M}$, its new $deg(v)$ can be much higher than its old $deg(v)$. Hence, we need to explicitly associate this increase of degree to the update which led to this increase. To this end, we analyze the algorithm using the following potential function: $\Phi = \sum_{v \notin \mathcal{M}} deg(v)$. Thus, the amortized cost (time) of an update is the sum of the actual work done and the change in potential.

The insertion of a vertex v requires amortized $O(deg(v))$ time for the actual $deg(v)$ work to add the edges, and the increase in potential Φ. The potential Φ can be increased by $O(deg(v))$ because of v (if $v \notin \mathcal{M}$), and its neighbours not in $\mathcal{M}$ (change in their degrees). For the remaining updates, note that the work done for the addition of a vertex v in $\mathcal{M}$ is always balanced by the corresponding decrease in potential by $deg(v)$. *We thus only focus on the work done for removing a vertex from $\mathcal{M}$ and the change in potential due to change in the degree of vertices.* The deletion of a vertex v requires amortized $O(1)$ time, as the time required to remove edges of v is $deg(v)$, and the potential Φ decreases by at least $deg(v)$ (if $v \notin \mathcal{M}$ then Φ reduces due to v, else it reduces due to neighbours of v). The insertion of an edge (u, v) requires amortized $O(1)$ time if both u and v are not simultaneously in $\mathcal{M}$, for the $O(1)$ work done to add the edge in the graph and $O(1)$ increase in Φ because of degrees of its endpoints. In case both $u, v \in \mathcal{M}$, the algorithm removes one vertex (say v) from $\mathcal{M}$, which requires amortized $O(deg(v))$ time for actual $deg(v)$ time taken to update all the neighbours of v and increase in potential Φ by $O(deg(v))$, because of v and its neighbours. The deletion of an edge (u, v) again takes amortized $O(1)$ time for $O(1)$ work done to remove the edge and decrease in Φ. Finally, since at most one vertex (if any) is removed from $\mathcal{M}$ during an update, the adjustment complexity is amortized $O(1)$ as the removal also accounts for the future insertion of the vertex in $\mathcal{M}$. Thus, we have the following theorem.

Theorem 3.2. *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be maintained using $O(1)$ amortized adjustments per update, where the amortized update time is $O(deg(v))$ if v is the only vertex (if any) that is removed from $\mathcal{M}$ or the vertex inserted into G (vertex insertion), otherwise the amortized update time is $O(1)$.*

Remark: This does not imply tighter amortized bound to the fully dynamic algorithm, but is merely described to aid in the analysis of future algorithms. Further, since adjustment complexity is amortized $O(1)$, the algorithm can be trivially adapted to the distributed $\mathcal{CONGEST}$ model [2] requiring amortized $O(1)$ rounds per update.

3.1.2 Hardness of Dynamic MIS

Using Theorem 3.2 we can understand the hardest form of update in case of dynamic MIS. Additionally, we use the fact that a vertex v is removed from $\mathcal{M}$ only in case of edge insertion or when v is deleted. Consider the case of fully dynamic MIS under only vertex updates. Here each vertex v can enter into $\mathcal{M}$ only once, either on being inserted or when all its neighbours in $\mathcal{M}$ are deleted. Thus, it requires total $O(deg(v))$ time throughout its lifetime, requiring overall $O(m)$ time. Similarly, consider the case of decremental MIS under edge updates. Here again, each vertex v can enter $\mathcal{M}$ exactly once, and never leave $\mathcal{M}$. Hence, using Theorem 3.2, it incurs amortized cost of $O(1)$, requiring total $O(m)$ time. Thus, the algorithm $\mathcal{A}_s$ works optimally under fully dynamic vertex updates or decremental MIS under edge updates.

As a result, the only update for which the simple algorithm does not solve the problem optimally is that of edge insertions, leaving the two problems of incremental MIS and fully dynamic MIS under edge updates. Under the current analysis both these algorithm takes total $\Omega(m)$ time which may not be optimal. Thus, we shall now focus on solving the two problems better than the improved fully dynamic MIS algorithm by Assadi et al. [2] requiring amortized $O(\min\{\Delta, m^{3/4}\})$ time.

4 Improved algorithm for Fully Dynamic MIS

In addition to the simple $O(\Delta)$ amortized time algorithm $\mathcal{A}_s$, Assadi et al. [2] presented a substantially complex fully dynamic algorithm requiring amortized $O(\min\{\Delta, m^{3/4}\})$ time per update. We show a very simple extension to $\mathcal{A}_s$ that achieves amortized $O(\min\{\Delta, m^{2/3}\})$ time per update.

The core idea used by the improved algorithm of Assadi et al. [2] gives partial preference to low degree vertices when being inserted to $\mathcal{M}$. More precisely, in *some cases* they allow a low degree vertex v to be inserted to $\mathcal{M}$ despite having high degree neighbours in $\mathcal{M}$. This is followed by removal of these high degree neighbours from $\mathcal{M}$ to maintain MIS property. We essentially use the same idea by using a stronger preference order. The low degree vertices are *always* inserted to $\mathcal{M}$ despite having high degree neighbours in $\mathcal{M}$.

The main idea is as follows. We divide the vertices of the graph into *heavy* vertices V_H having degree $\geq \Delta_c$, and *light* vertices V_L having degree $< \Delta_c$, for some fixed constant Δ_c. Let the subgraph induced by the vertices in V_H and V_L be $G_H = (V_H, E_H)$ and $G_L = (V_L, E_L)$ respectively. We use simple algorithm $\mathcal{A}_s$ to maintain the MIS of G_L in amortized $O(\Delta_c)$ time. After every such update, we can afford to rebuild the MIS for G_H from scratch, using the trivial static algorithm. Since the total number of heavy vertices can be $O(m/\Delta_c)$, the time taken to build the MIS for the heavy vertices is $O(|E_H|) = O((m/\Delta_c)^2)$ time. Choosing $\Delta_c = m^{2/3}$ we get an amortized $O(m^{2/3})$ update time algorithm. However, if $\Delta \leq m^{2/3}$ the entire graph is present in G_L and we have an empty G_H. Hence, our algorithm merely performs $\mathcal{A}_s$ on the whole graph, resulting in amortized $O(\min\{\Delta, m^{2/3}\})$ update time of our algorithm. Note that the main reason for faster update of this algorithm (compared to $\mathcal{A}_s$) is as follows: When a heavy vertex enters or leaves $\mathcal{M}$, it does not inform its light neighbours.

Implementation Details

We shall now describe a few low level details regarding the implementation of the algorithm. Each vertex of both V_H and V_L will store a *count* of their light neighbours in $\mathcal{M}$, i.e. neighbours in $V_L \cap \mathcal{M}$. This *count* can be maintained easily by each vertex as whenever a light vertex enters of leaves $\mathcal{M}$, it informs all its Δ_c neighbours (both heavy and light).

As we have mentioned before, we use algorithm $\mathcal{A}_s$ to maintain a maximal independent set for all the light vertices. Thus, at each step we may have to rebuild the MIS for the heavy vertices from scratch. While rebuilding MIS for G_H we have to only consider those heavy vertices whose *count* $= 0$, i.e., which do not have any light neighbour in $\mathcal{M}$. We can rebuild the MIS of G_H from scratch in $O(|E_H|)$ time using the trivial static algorithm. We have already argued that $|E_H| = O((m/\Delta_c)^2)$. Thus, the update time of our algorithm for light vertices is amortized $O(\Delta_c)$ (by algorithm $\mathcal{A}_s$) and the update time of our algorithm for heavy vertices is $O((m/\Delta_c)^2)$. Choosing $\Delta_c = m^{2/3}$, the update time of our algorithm becomes $O(m^{2/3})$.

Finally, note that the value of Δ_c have to be changed during the fully dynamic procedure, because of the change is the value of m. This can be achieved using the standard technique of *periodic rebuilding*, which can be described using *phases* as follows. We choose $\Delta_c = m_c^{2/3}$, where m_c denotes the number of edges in the graph at the start of a phase. Hence at the start of the algorithm, $m_c = m$. Whenever m decreases to $m_c/2$ or m increases to $2m_c$, we end our current phase and start a new phase. Also, we re-initialize *count* and rebuild $\mathcal{M}$ but using the new value of m_c, and hence new V_L. Note that the cost $O(m)$ for this rebuilding at the start of each phase is accounted to the $\geq m/2$ edge updates during the phase. Thus, we have the following theorem.

Theorem 1.1 (Fully dynamic MIS)**.** *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be maintained in $O(\min\{\Delta, m^{2/3}\})$ amortized time per insertion or deletion, where Δ is the maximum degree of a vertex in G.*

5 Improved Algorithm for Incremental MIS

We shall now consider the incremental MIS problem and show that if we restrict the updates to edge insertion only, we can achieve a faster amortized update time of $O(\min\{\Delta, \sqrt{m}\})$. In this algorithm, we again give preference for a vertex being in MIS, based on its degrees. However, instead of dividing the vertices into sets of V_L and V_H explicitly, we simply consider the exact degree of the vertex.

Recall that in case of insertion of an edge, the simple algorithm $\mathcal{A}_s$ updates $\mathcal{M}$, only when both the end vertices belong to $\mathcal{M}$. This leads to one of the vertices (say v) being removed from $\mathcal{M}$, and the *count* of its neighbours being updated. Several of these neighbours can now have $count = 0$ and hence enter $\mathcal{M}$. However, using Theorem 3.2 we know that the amortized time required by the update is only $O(deg(v))$, the degree of the vertex removed from $\mathcal{M}$.

Our main idea is to modify $\mathcal{A}_s$ to ensure that when an edge is inserted between two vertices in $\mathcal{M}$, the end vertex with lower degree is removed from $\mathcal{M}$. Note that in the original $\mathcal{A}_s$, the end vertex to be removed from MIS is chosen arbitrarily. We show that this key difference of choosing the lower degree (instead of arbitrary) end vertex to be removed from $\mathcal{M}$, proves crucial in improving the amortized update time. We further prove our argument by showing worst case examples demonstrating the tightness of the upper bounds of the two algorithms (see Appendix A).

5.0.1 Analysis

We shall now analyze our incremental MIS algorithm. Consider the insertion of i^{th} edge (u_i, v_i), where without loss of generality u_i is the lower degree vertex amongst u_i and v_i. Using Theorem 3.2, we know that the amortized update time of the algorithm is $O(deg(u_i))$ in case both $u_i, v_i \in \mathcal{M}$, else $O(1)$. Hence, the total update time taken by the algorithm is $\sum_{i=1}^{m} O(deg(u_i))$.

We shall analyze the time taken by a vertex, during two *phases*, when it is *light* ($deg(u_i) \leq \sqrt{m}$), and when it becomes *heavy* ($deg(u_i) > \sqrt{m}$). If u_i is light, then we simply remove u_i from $\mathcal{M}$ in $O(\sqrt{m})$ amortized time (as $deg(u_i) \leq \sqrt{m}$). If u_i is heavy, then v_i must also be heavy. However, there are only $O(\sqrt{m})$ heavy vertices in the graph. Hence, a heavy vertex u_i can be removed $O(\sqrt{m})$ times from $\mathcal{M}$ due to its heavy neighbours. Further, the time taken for such a vertex is $O(deg_f(u_i))$, where $deg_f(u_i) (\geq deg(u_i))$ is the final degree of u_i after the end of updates. Thus, the total update time is calculated as follows.

$$\begin{aligned}
\sum_{i=1}^{m} O(deg(u_i)) &= \sum_{u_i \text{ is light}} O(deg(u_i)) + \sum_{u_i \text{is heavy}} O(deg(u_i)) \\
&\leq \sum_{i=1}^{m} O(\sqrt{m}) + \textstyle\sum_{u \in V} |\{(u,v) \in E | v \text{ is heavy}\}| \times O(deg(u)) \\
&\leq \sum_{i=1}^{m} O(\sqrt{m}) + \textstyle\sum_{u \in V} \sqrt{m} \times O(deg_f(u)) \\
&\leq O(m\sqrt{m}) + O(m\sqrt{m})
\end{aligned}$$

Hence, the total update time of our incremental algorithm is $O(m\sqrt{m})$. Also, since it is simply a special case of the simple MIS algorithm [2], we also have adjustment complexity of amortized $O(1)$, and an upper bound of $O(m\Delta)$ total time, making the amortized update time $O(\min\{\Delta, \sqrt{m}\})$.

Theorem 1.2 (Incremental MIS)**.** *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be maintained in $O(\min\{\Delta, \sqrt{m}\})$ amortized time and $O(1)$ amortized adjustments per edge insertion, where Δ is the maximum degree of a vertex in G.*

Note: This technique cannot be trivially extended to the fully dynamic setting. This is because here the crucial fact exploited is that the high degree vertices would be removed less number of times from MIS, which cannot be ensured in a fully dynamic environment. Also, as $\mathcal{A}_s$ it can be trivially adapted to $\mathcal{CONGEST}$ model with amortized $O(1)$ rounds and adjustments per update.

We have seen that the key difference of choosing the lower degree (instead of arbitrary) end vertex to be removed from $\mathcal{M}$ proves crucial in reducing the amortized update time. This argument can be proved by the following worst case examples demonstrating the tightness of the upper bounds of the two algorithms (see Appendix A). Note that this example for $\mathcal{A}_s$ in incremental setting also shows tightness of the fully dynamic case, as the incremental setting is merely a special case of the fully dynamic setting.

6 MIS with worst case guarantees

Assadi et al. [2] demonstrated using a worst case example that the adjustment complexity (number of vertices which enter or leave $\mathcal{M}$ during an update) can be $\Omega(n)$ in the worst case. This justifies why the entire work on dynamic MIS is focused primarily on amortized bounds. However, in case we still want to achieve better worst case bounds, we are required to consider a *relaxed* model, where we settle at not maintaining the MIS *explicitly*. Rather we allow queries to answer whether a vertex is present in MIS after an update, such that the results of all the queries are consistent to some MIS of the graph.

Implicit Maintenance of MIS

We now formally define this model for the dynamic maintenance of MIS. The model supports *updates* in the graph, such that the MIS is not *explicitly* maintained after each update. Additionally, the model allows queries of the form In-Mis(v), which reports whether a vertex $v \in V$ is present in the MIS of the updated graph. Such a model essentially allows us to compute $\mathcal{M}$ partially, along with maintaining some information which makes sure that the results of the queries are consistent with each other. Thus, this model allows us to achieve $o(n)$ time worst case bounds for both queries and updates, for the dynamic maintenance of MIS.

The underlying idea is as follows. Recall that in every update of $\mathcal{A}_s$ exactly *one* vertex is removed from $\mathcal{M}$ and several vertices may be added to $\mathcal{M}$ (see Theorem 3.2). Thus, it is easier to maintain an *independent set* rather than a *maximal independent set*, by always processing the removal of a vertex from $\mathcal{M}$ but not the insertion of vertices in $\mathcal{M}$. Hence, we only make sure that after each update, no two vertices in $\mathcal{M}$ share an edge which leads to removing exactly one vertex from $\mathcal{M}$. Now, whenever a vertex is queried we verify whether it is already in $\mathcal{M}$ or can be moved to $\mathcal{M}$, and respond accordingly. In order to resolve these queries efficiently, we again maintain the *partial count* of neighbours of a vertex in $\mathcal{M}$, which is described as follows.

We consider the vertices with high degree ($> \sqrt{m}$) as *heavy* and the rest as *light*, resulting in total $O(\sqrt{m})$ heavy vertices. Our algorithm maintains the *count* for only heavy vertices and not for light vertices. Thus, whenever a vertex enters or leaves $\mathcal{M}$ it only informs its heavy neighbours. We shall now describe the *update* and *query* algorithms.

The *update* algorithm merely updates the *count* of heavy end vertex (if any) in case of edge deletion, or edge insertion when both end vertices are not in $\mathcal{M}$. For edge insertion having both end vertices in $\mathcal{M}$, it removes one of the vertices from $\mathcal{M}$ and updates the *count* of its $O(\sqrt{m})$ heavy neighbours. Note that the update algorithm does not ensure that $\mathcal{M}$ is an MIS, but necessarily ensures that $\mathcal{M}$ is an *independent set* since it always removes a vertex from $\mathcal{M}$ if any of its neighbours is in $\mathcal{M}$.

A vertex v may be added to $\mathcal{M}$ only if it is queried in In-Mis(v) as follows. If $v \in \mathcal{M}$, then we simply report it. Else, there are two cases depending on whether v is light or heavy. If v is heavy and its *count* is zero, it implies that v has no neighbour in $\mathcal{M}$. Hence, we simply add v into $\mathcal{M}$ and update the *count* of its heavy neighbours. However, if v is light we do not have the *count* of its neighbours in $\mathcal{M}$. Hence, we check if there is any neighbour of v in $\mathcal{M}$ in $O(\sqrt{m})$ time. If none of the neighbours of v are in $\mathcal{M}$, then we add v to $\mathcal{M}$ and update the *count* of its heavy neighbours. This completes the query algorithm. It is easy to see that both *update* and *query* algorithms require $O(\sqrt{m})$ worst case time to add or remove at most one vertex from $\mathcal{M}$ and visit its $O(\sqrt{m})$ neighbours. Further, in case $\Delta < \sqrt{m}$, this update and query time reduces to $O(\Delta)$, as each vertex can have $O(\Delta)$ neighbours.

Now, in the fully dynamic setting the value of m may change significantly after sufficient updates. So we instead define the *heavy* status of a vertex using a constant m_c which is initialized as $m_c = m$, and we ensure that $m_c/2 < m < 2m_c$. In case the value of m increases beyond $2m_c$, we simply need to make some *heavy* vertices light by removing their corresponding value of *count*. This can be performed in a single step in $O(\sqrt{m})$ time as there are only $O(\sqrt{m})$ heavy vertices, and we update $m_c = 2m_c$. However, in case the value of m decreases below m_c, we need to compute the value of *count* for the *light* vertices in each update, which will become *heavy* if m_c is reduced to half. Since total degree of all vertices is $O(m)$, there can be only $O(\sqrt{m})$ such vertices. Hence, in the next $O(\sqrt{m})$ updates we can compute the *count* for one such vertex in each update requiring $O(\sqrt{m})$ time. Since we have $O(m)$ updates before the value of $m = m_c/2$ and we update $m_c = m_c/2$, all the new *heavy* vertices will have computed its value of *count*. Also, if an edge update changes the *heavy* status of its end points, their corresponding *count* can be updated in $O(\sqrt{m})$ time.

In order to prove the correctness of our algorithm, we are required to prove that (1) $\mathcal{M}$ remains an independent set throughout the algorithm, and (2) All the neighbours of a vertex v are verified before answering a query In-Mis(v). The former clearly follows from the update algorithm, and to prove the latter we look at the heavy and light vertices separately. Since every vertex entering or leaving $\mathcal{M}$ necessarily informs its heavy neighbours, the second condition is clearly true for heavy vertices. For light vertices, the correctness of the second condition follows from the query algorithm. Finally, since at most one vertex (if any) leaves MIS in an update or enters MIS in a query, the adjustment complexity is $O(1)$. However, if we consider adjustment complexity considering updates only, it is still amortized $O(1)$ (similar to $\mathcal{A}_s$). Thus we have the following theorem.

Theorem 6.1 (Fully dynamic MIS (worst case))**.** *Given any graph $G = (V, E)$ having n vertices and m edges, MIS can be implicitly maintained under fully dynamic edge updates requiring $O(1)$ adjustments per update and query, allowing queries of the form* In-Mis(v)*, where both update and query requires worst case $O(\min\{\Delta, \sqrt{m}\})$ time.*

Note: The above described algorithm can be trivially adapted to distributed $\mathcal{CONGEST}$ model requiring $O(\min\{\Delta, \sqrt{m}\})$ messages, and $O(1)$ rounds and adjustments per update or query in the worst case.

Remark: The algorithm can also support vertex deletion similar to edge insertion in the same time. However, vertex insertion with an arbitrary number of incident edges would not be allowed as processing input itself may require $O(\Delta)$ time. Thus, allowing fully dynamic vertex updates requires $O(\Delta)$ worst case update time. However, in case the vertices are inserted without any incident edges the same complexity of $O(\min\{\Delta, \sqrt{m}\})$ is also applicable for fully dynamic vertex updates.

7 Maximum Flow and Maximum Matching

A standard approach to both these problems uses the concept of *augmenting paths*, where finding an augmenting path allows us to increase the value of the solution by a single unit. Computing an augmenting path takes $O(m)$ time for both the problems and in the unweighted case the maximum value of the solution can be $O(n)$. This simply gives an $O(mn)$ time algorithm to compute the solution for both the problems in the static setting.

However, maintaining them in the dynamic setting have not been explicitly examined. We report that both these bounds can be exactly matched by trivial extensions of the *augmenting path* algorithms in the incremental setting. Concretely, the incremental reachability algorithm [29] can be used to incrementally compute an augmenting path for maximum flow in total $O(m)$ time. Similarly, the blossoms algorithm [19] (which is inherently incremental) can be used to incrementally compute an augmenting path for maximum matching in $O(m)$ time. Once the augmenting path is found, the solution is updated and its value is increased by 1 unit. Then we restart the computation of augmenting path in the updated graph. This process can be repeated $O(n)$ times, which is the maximum value of the solution. Hence, both the problems can be solved in total $O(mn)$ time, matching the lower bound by Dahlgaard [10]. Furthermore, the fully dynamic algorithms for both these problems requiring $O(m)$ time per update are known as *folklore*, though not explicitly stated in the literature. Both these algorithms are also based on the *augmenting path* approach. We also state those algorithms for the sake of completeness. Refer to Appendix B and Appendix C for details.

8 Conclusion

We have presented several surprisingly simple algorithms for fundamental graph problems in the dynamic setting. These algorithms either improve the known upper bounds of the problem or match the known lower bounds. Additionally, we considered some relaxed settings under which such problems can be solved better. A common trait among all these algorithms is that they are extremely simple and use no complicated data structures, making it suitable for even classroom teaching of fundamental concepts as *amortization* and introducing *dynamic graph algorithms*.

In the dynamic MIS problem, we also discuss the hardness of the problem in the dynamic setting. Most graph problems (as connectivity, reachability, maximum flow, maximum matching, MST, DFS, BFS etc.) are found to be harder to handle vertex updates instead of edge updates and handle deletions instead of insertions. This is surprisingly the opposite case with dynamic MIS, where except for edge insertions (which is the easiest update for most other problems), a trivial algorithm solves the problem optimally. Notably, this is also the case a few other fundamental problems as topological ordering, cycle detection, planarity testing, etc. We conjecture the reason for such a behaviour to be the following fundamental property: *If the solution of the problem is still valid (though sub-optimal) after an update, it shall be easier to handle the update.* This supports the behaviour of the problems mentioned above, both for which edge insertions are easiest, and for which they are hardest.

Finally, in the light of the above discussion, we propose some future directions of research in these problems. It seems that the fully dynamic MIS under edge updates, shouldn't be much harder than the incremental setting. Hence, it would be interesting to see an algorithm to maintain fully dynamic MIS in $O(\min\{\Delta, \sqrt{m}\})$ amortized update time, which can preferably also be extended to the distributed $\mathcal{CONGEST}$ model with amortized $O(1)$ round and adjustment complexity. On the other hand, we believe decremental unit capacity maximum flow and maximum cardinality matching would be harder than the incremental setting. Hence, stronger lower bounds for these problems in the decremental setting would be interesting.

References

[1] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139, 2012.

[2] 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, Los Angeles, CA, USA, June 25-29, 2018*, pages 815–826, 2018.

[3] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 321–330, 2012.

[4] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in o(log n) update time. *SIAM J. Comput.*, 44(1):88–113, 2015.

[5] Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *J. Algorithms*, 62(2):74–92, 2007.

[6] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711. SIAM, 2016.

[7] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 398–411. ACM, 2016.

[8] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 470–489. SIAM, 2017.

[9] 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, pages 217–226, 2016.

[10] Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 48:1–48:14, 2016.

[11] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.

[12] Camil Demetrescu and Giuseppe F. Italiano. Mantaining dynamic matrices for fully dynamic transitive closure. *Algorithmica*, 51(4):387–427, 2008.

[13] Jack Edmonds. *Paths, Trees, and Flowers*, pages 361–379. Birkhäuser Boston, Boston, MA, 1987.

[14] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

[15] Shimon Even and Robert Endre Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975.

[16] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 1962.

[17] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.

[18] Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 246–251, 1983.

[19] Harold Neil Gabow. *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs.* PhD thesis, Stanford University, Stanford, CA, USA, 1974.

[20] Zvi Galil, Silvio Micali, and Harold N. Gabow. An O(EV log V) algorithm for finding a maximal weighted matching in general graphs. *SIAM J. Comput.*, 15(1):120–130, 1986.

[21] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[22] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 270–277, 2016.

[23] Andrew V. Goldberg, Sagi Hed, Haim Kaplan, Pushmeet Kohli, Robert Endre Tarjan, and Renato F. Werneck. Faster and more dynamic maximum flow by incremental breadth-first search. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 619–630, 2015.

[24] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5), September 1998.

[25] Andrew V. Goldberg and Robert Endre Tarjan. Efficient maximum flow algorithms. *Commun. ACM*, 57(8):82–89, 2014.

[26] Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science*, 2013.

[27] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[28] 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(4):723–760, 2001.

[29] Giuseppe F. Italiano. Amortized efficiency of a path retrieval data structure. *Theor. Comput. Sci.*, 48(3):273–281, 1986.

[30] Peter Jeavons, Alex Scott, and Lei Xu. Feedback from nature: simple randomised distributed algorithms for maximal independent set selection and greedy colouring. *Distributed Computing*, 29(5):377–393, 2016.

[31] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *SODA*, pages 1131–1141, 2013.

[32] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[33] Pushmeet Kohli and Philip H. S. Torr. Dynamic graph cuts and their applications in computer vision. In *Computer Vision: Detection, Recognition and Reconstruction*, pages 51–108. Springer Berlin Heidelberg, 2010.

[34] S. Kumar and P. Gupta. An incremental algorithm for the maximum flow problem. *J. Math. Model. Algorithms*, 2(1):1–16, 2003.

[35] Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 653–664, 2012.

[36] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961, 2017.

[37] Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic MIS in uniformly sparse graphs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 92:1–92:14, 2018.

[38] James B. Orlin. Max flows in o(nm) time, or better. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774, 2013.

[39] Liam Roditty and Uri Zwick. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. *SIAM J. Comput.*, 45(3):712–733, 2016.

[40] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238, 2011.

[41] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 509–517, 2004.

[42] Shay Solomon. Fully dynamic maximal matching in constant update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334, 2016.

[43] Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 112–119, 2005.

A Tightness of Incremental Algorithms

We now present worst case examples demonstrating the tightness of the analysis of $\mathcal{A}_s$ and our incremental MIS algorithm. These essentially highlight the difference between arbitrary removal and degree biased removal of an end vertex on insertion of an edge between two vertices of the MIS.

A.1 Arbitrary removal

We start with an empty graph where all the vertices are in MIS. Let the vertices be divided into two sets $\mathcal{A} = \{a_1, ..., a_k\}$ and $\mathcal{B} = b_1, ..., b_t$ (refer to Figure 1), where $\mathcal{A}$ has $k = m/\Delta$ vertices and $\mathcal{B}$ has the remaining $t = n - m/\Delta = O(n)$ vertices.

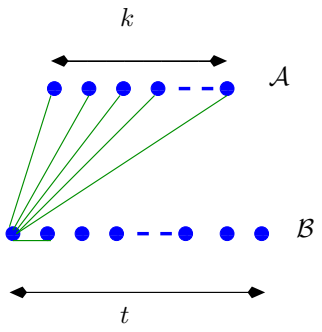

Figure 1: Worst case example for arbitrary removal

We shall divide the insertion of edges into t phases, where in the j^{th} phase we add an edge from each vertex of $\mathcal{A}$ to b_j. And we always chose the vertex of $\mathcal{A}$ to be removed from MIS $\mathcal{M}$. At the end of the phase, all vertices of $\mathcal{A}$ are out of $\mathcal{M}$ and are connected to $b_j \in \mathcal{M}$. The phase ends with the addition of an edge between b_j and b_{j+1}, which removes b_j from $\mathcal{M}$ and hence all vertices of $\mathcal{A}$ are moved back to $\mathcal{M}$. Since each vertex is allowed only Δ neighbours, and each phase adds a neighbour to each vertex in $\mathcal{A}$, we stop after $t^* = \Delta$ phases.

Hence, after t^* phases we have added all the edges between $\mathcal{A}$ and the first t^* vertices in $\mathcal{B}$, and among the first t^* adjacent vertices of $\mathcal{B}$. Overall we add $O(|\mathcal{A}| \times t^* + t^*)$ edges, which equals to $O(kt^*) = O(\frac{m}{\Delta} * \Delta) = O(m)$ edges.

Using Theorem 3.2, the total edges processed during the j^{th} phase, is the sum of the degrees of vertices that were removed from $\mathcal{M}$, i.e., all the vertices in $\mathcal{A}$ and b_j. Now, in the j^{th} phase the degree of each vertex in $\mathcal{A}$ is $j-1$, being connected to $b_1, ..., b_{j-1}$ and the degree of b_j is k. Hence, the total work in j^{th} phase is $|\mathcal{A}| \times (j-1) + k = k \times j$. Thus, the total work done over all phases is

$$\sum_{j=1}^{t^*} \Omega(k \times j) = \Omega(k \times t^{*2}) = \Omega(\frac{m}{\Delta} \times \Delta^2) = \Omega(m\Delta)$$

Thus, we have the following bound for $\mathcal{A}_s$ in incremental (and hence fully dynamic) setting.

Theorem A.1. *For each value of $n \leq m \leq \binom{n}{2}$ and $1 \leq \Delta \leq n$, there exists a sequence of m edge insertions where the degree of each vertex is bounded by Δ for which $\mathcal{A}_s$ requires total $\Theta(m\Delta)$ time to maintain the MIS.*

A.2 Degree biased removal

In this example, we consider our incremental MIS algorithm, which chooses the end vertex with the lower degree to be removed from $\mathcal{M}$ when an edge is inserted between two vertices in $\mathcal{M}$.

We essentially modify the previous example to make sure the vertices of $\mathcal{A}$ necessarily fall when connected to the vertices in $\mathcal{B}$.

Let the vertices to be divided into two sets $\mathcal{A} = \{a_1, ..., a_k\}$ and $\mathcal{B} = \{b_1, ..., b_t\}$ as before, and an additional set $\mathcal{C}$ of residual vertices to ensure that degrees of vertices in $\mathcal{B}$ are sufficiently high (refer to Figure 2), where $k = t = \sqrt{m}/4$. We connect each vertex b_i with some $\sqrt{m}+1$ vertices in $\mathcal{C}$. Additionally, we have a vertex b_0, connected to all the $O(n)$ vertices in $\mathcal{C}$. We initialize the MIS with all vertices of $\mathcal{A}$, $\mathcal{B}$ and b_0 in $\mathcal{M}$.

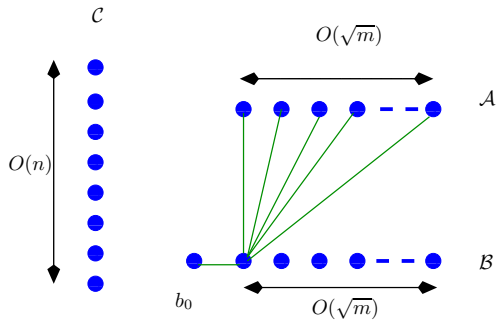

Figure 2: Tightness Example for degree biased removal

Again, we divide the insertion of edges into t phases, where in the j^{th} phase we add an edge from each vertex of $\mathcal{A}$ to b_j. Since the maximum degree of a vertex in $\mathcal{A}$ is $|\mathcal{B}|$, and degree of each b_j is $\sqrt{m}+1$, we always have the vertex of $\mathcal{A}$ to be removed from MIS $\mathcal{M}$. At the end of the phase, all vertices of $\mathcal{A}$ are out of $\mathcal{M}$ and are connected to $b_j \in \mathcal{M}$. The phase ends with the addition of (b_j, b_0), which removes b_j from $\mathcal{M}$ and hence all vertices of $\mathcal{A}$ are moved back to $\mathcal{M}$.

Hence, after t phases we have added all the edges between $\mathcal{A}$ and $\mathcal{B}$, between each vertex of $\mathcal{B}$ and b_0. The initial graph already had each vertex of $\mathcal{B}$ connected to some $\sqrt{m}+1$ neighbours in $\mathcal{C}$ and b_0 connected to all neighbours of $\mathcal{C}$. Overall we add $O(|\mathcal{A}| \times |\mathcal{B}| + |\mathcal{B}| \times (\sqrt{m}+1) + |\mathcal{B}| + n)$ edges, which equals to $O(kt + t\sqrt{m} + n) = O(m + n)$ edges.

Again, using Theorem 3.2, the total edges processed during the j^{th} phase, is the sum of the degrees of vertices that were removed from $\mathcal{M}$, i.e., all the vertices in $\mathcal{A}$ and b_j. Now, in the j^{th} phase the degree of each vertex in $\mathcal{A}$ is $j-1$, being connected to $b_1, ..., b_{j-1}$ and the degree of b_j is $\Omega(\sqrt{m})$. Hence, the total work done in j^{th} phase is $\Omega(|\mathcal{A}| \times j + \sqrt{m}) = \Omega(k \times j + \sqrt{m})$. Thus, the total work done over all phases is

$$\sum_{j=1}^{t} \Omega(k \times (j-1) + \sqrt{m}) = \Omega(k \times t^2 + k\sqrt{m}) = \Omega(\sqrt{m} \times \sqrt{m}^2 + m) = \Omega(m\sqrt{m})$$

Thus, we have the following bound for our incremental MIS algorithm.

Theorem A.2. *For each value of $n \leq m \leq \binom{n}{2}$, there exists a sequence of m edge insertions for which our incremental algorithm requires total $\Theta(m\sqrt{m})$ time to maintain the MIS.*

Remark: This example is similar to the previous example with an exception that in this case, the degree of vertices in $\mathcal{B}$ should remain higher than the degree of a vertex in $\mathcal{A}$, i.e., $|\mathcal{B}|$ at the end of all the phases. Hence, $|\mathcal{B}| \leq \sqrt{m}$ else the total edges in G would be $\Omega(m)$. Thus, the number of neighbours of $\mathcal{A}$ in $\mathcal{B}$ cannot be increased despite choosing any large value of Δ. As evident from the previous example in absence of such a restriction, the amortized time can be raised to Δ implying the significance of the degree biased removal in the incremental algorithm.

B Unit capacity Maximum Flow

The maximum flow problem is one of the most studied combinatorial optimization problem having a lot of practical applications (see [25] for a survey). Given a graph $G = (V, E)$ having n vertices and m edges where each edge has a capacity (positive real weight) $c : E \to \mathbb{R}^+$ associated to it. A flow from a *source* s to a *sink* t is an assignment of flow $f : E \to \mathbb{R}^+$ to each edge such that it satisfies the following two constraints. Firstly, the *capacity* constraints imply that the flow on each edge is limited by its capacity, i.e. $\forall e \in E, f(e) \leq c(e)$. Secondly, the *conservation* constraints imply that for each non-terminal vertex ($v \in V \setminus \{s, t\}$), the flow passing through v is conserved, i.e. $\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w)$. The amount of flow leaving s or entering t is referred as the flow F of the network, i.e., $F = \sum_{(s,u)\in E} f(s, u) = \sum_{(v,t)\in E} f(v, t)$. Clearly, in a unit capacity simple graph $F = O(n)$, corresponding to each edge leaving s (or entering t). The maximum flow problem evaluates the flow f from s to t which maximizes the value of F.

Orlin [38] presented an algorithm to find the maximum flow in $O(mn)$ time. For integral arc capacities (bounded by U), it can be evaluated $O(m \cdot \min(n^{2/3}, m^{1/2}) \log(n^2/m) \log U))$ time [24], which further reduces to $O(m \cdot \min(n^{2/3}, m^{1/2}))$ time for unweighted (unit capacity) graphs [15]. In the dynamic setting, a few algorithms are known for maintaining maximum flow. Kumar and Gupta [34] showed that partially dynamic maximum flow can be maintained in $O(\Delta n^2 m)$ update time, where Δn is the number of vertices whose flow is affected. Other algorithms are by Goldberg et al. [23] and Kohli and Torr [33], which work very well in practice on computer vision problems but do not have strong asymptotic guarantees.

Recently, Dahlgaard [10] proved conditional lower bounds for partially dynamic problems including maximum flow under OMv conjecture. For directed and weighted sparse graphs, no algorithm can maintain partially dynamic max flow in $O(m^{1-\epsilon})$ amortized update time. For directed unweighted (unit capacity) graphs and undirected weighted graphs, no algorithm can maintain partially dynamic maximum flow in $O(n^{1-\epsilon})$ amortized update time. We report a trivial extension of the incremental reachability alogrithm [29] to solve the incremental unit capacity (unweighted) maximum flow problem matching its corresponding lower bound.

We shall now describe some simple dynamic algorithms for maintaining maximum flow in unit capacity graphs. The main idea behind such algorithms is based on the classical *augmenting path* approach of the standard Ford Fulkerson [16] algorithm. In the interest of completeness, before describing our incremental algorithm for maintaining maximum flow in $O(F)$ (where $F = O(n)$) amortized time per update, we present the *folklore* algorithm supporting fully dynamic edges updates in $O(m)$ time. To describe these algorithms succinctly, we briefly describe the *augmenting path* approach and the concept of *residual graphs*.

Residual Graphs and Augmenting paths

Given an unweighted (unit capacity) directed graph $G = (V, E)$, having flow function defined on each edge $f : E \to \{0, 1\}$. The *residual graph* is computed by changing the direction of every edge e, if $f(e) = 1$. Note that this may create two edges in a graph between the same endpoints. This residual graph allows a simple characterization of whether the current flow F is indeed a maximum flow because of the following property: *If there exists an $s - t$ path in the residual graph, referred as* augmenting path*, the value of the flow can be increased by pushing flow along the* augmenting *path in the residual graph* [16]. This results in reversing the direction of each edge on this path in the updated residual path. The flow reaches the maximum value when there does not exist any augmenting path, i.e. $s - t$ path, in the residual graph.

B.1 Fully Dynamic Maximum Flow

We now describe the *folklore* algorithm for maintaining Maximum Flow of a unit capacity graph under fully dynamic edge updates. The flow is updated by the computation of a single augmenting path in the residual graph after the corresponding edge update as follows.

- **Insertion of an edge:** An edge insertion can either increase the maximum flow by one unit or leave it unchanged. Recall that when f is a maximum flow, the sink t is not reachable from the source s in the residual graph. Hence if the inserted edge creates an $s-t$ path in the residual graph, the maximum flow increases by exactly one unit. Thus, the algorithm finds an $s-t$ path in the residual graph and pushes a flow of one unit along the path. If no such path is found, the maximum flow has not increased and the solution remains unchanged.
- **Deletion of an edge:** If the deleted edge doesn't carry any flow, the flow remains unchanged. Otherwise, the edge deletion can either decrease the maximum flow or simply make the current flow invalid requiring us to reroute the flow without changing its value. Hence, on deletion of an edge (x, y) the algorithm first attempts to restore the flow by finding an alternate path from x to y in the residual graph. If such a path is found the algorithm pushes a flow of unit capacity along the path, restoring the maximum flow in the graph. Otherwise, we have to send back one unit of flow each, from x to s and from t to y, to reduce the maximum flow by one unit. For this an edge (s, t) is added and a path from x to y is found, which necessarily exists containing the edge (s, t). Again, the algorithm pushes a flow of unit capacity along the path, restoring the maximum flow in the graph. After updating the residual graph, we remove the extra added edge (s, t) from the graph.

Thus, each update can be performed in $O(m)$ time, as it performs $O(1)$ reachability queries (using BFS/DFS traversals) and updates the residual graph. Hence, we have the following theorem.

Theorem B.1 (Fully dynamic Unit Capacity Maximum Flow (folklore))**.** *Given an unweighted (unit-capacity) graph $G = (V, E)$ having n vertices and m edges, fully dynamic maximum flow under edge updates can be maintained in $O(m)$ worst case time per update.*

B.2 Incremental Maximum Flow

In the incremental setting, the unit-capacity maximum flow can be maintained using amortized $O(F)$ update time. Recall that in the fully dynamic algorithm, on insertion of an edge the flow is increased only if the t becomes reachable from s in the residual graph as a result of the update. Trivially, verifying whether t is reachable after each update requires $O(m)$ time per update, even when the flow is not increased. However, this can be computed more efficiently using the single source incremental reachability algorithm [29] requiring total $O(m)$ time for every increase in the value of flow. In the interest of completeness, let us briefly describe the incremental reachability algorithm [29] as follows.

Single source incremental reachability [29]

This algorithm essentially maintains a reachability tree T from the source vertex s. This tree T is initialized with a single node s and grown to include all the vertices reachable from s. On insertion of an edge (u, v), an update is required only when $u \in T$ and $v \notin T$. In such a case, the edge (u, v) is added making v a child of u. Further, the update algorithm processes every outgoing edge (v, w) of v, i.e., to find vertices $w \notin T$, which are added to T recursively using the same procedure. Thus, this process continues until all the vertices reachable from s are added to T. Clearly, the process

takes total $O(m)$ time as each edge (u, v) is processed $O(1)$ times, when it is inserted in the graph and when u is added to T.

Algorithm

The algorithm prominently uses the fact that the value of maximum flow F in a unit capacity simple graph is $O(n)$ since the number of outgoing edges from s (or the incoming edges to t) is $O(n)$. Our algorithm is divided into F stages where at the end of each stage the value of maximum flow increases by a single unit. Each stage starts by building an incremental single source reachability structure [29], i.e. the reachability tree T, from s on the residual graph. On insertion of an edge, the reachability tree T is updated using the incremental reachability algorithm. The stage continues until t becomes reachable from s and added to T. This gives the $s-t$ augmenting path and we push a unit of flow along the path and update the residual graph. Thus, each stage requires total $O(m)$ time for maintaining incremental reachability structure [29], taking overall $O(mF)$ time (where $F = O(n)$), giving us the following result.

Theorem B.2 (Incremental Unit Capacity Maximum Flow)**.** *Given an unweighted (unit-capacity) graph $G = (V, E)$ having n vertices and m edges, incremental maximum flow can be maintained in amortized $O(F)$ update time, where $F = O(n)$ is the value of the maximum flow of the final graph.*

Remark: At the end of a stage, it is necessary to rebuild the incremental reachability structure [29] from scratch as it does not support edge deletions or reversals. This is required because when the flow is pushed along the $s-t$ path the residual graph is updated by reversing the direction of each edge on the $s-t$ path.

C Maximum Cardinality Matching

Maximum Matching is one of the most prominently studied combinatorial graph problems having a lot of practical applications. For a given graph $G = (V, E)$ with n vertices and m edges, a set of edges $\mathcal{E}_M \subseteq E$ is called a *matching*, if no two edges in $\mathcal{E}_M$ share an end vertex in V. In the Maximum Matching problem, the aim is to compute the matching $\mathcal{E}_M$ of the maximum weight. In case the graph is unweighted, the problem is called Maximum Cardinality Matching. Micali and Vazirani presented an algorithm to compute maximum cardinality matching in $O(m\sqrt{n})$ time. For the weighted case, the fastest algorithm is by Galil et. al [20] requiring $O(mn \log n)$ time which improves the $O(n^3)$ time algorithm [19] for sparse graphs. In the dynamic setting, only the problem of computing α-approximate matching (having cardinality $\geq \mathcal{E}_M/\alpha$) has been extensively studied [42, 7, 6].

Recently, Dahlgaard [10] proved conditional lower bounds for partially dynamic problems including maximum matching under OMv conjecture. For bipartite graphs (and hence general graphs), no algorithm can maintain partially dynamic maximum cardinality matching in $O(n^{1-\epsilon})$ amortized update time. We report a trivial extension of the classical blossom's algorithm [13, 19], to solve this problem for general unweighted graphs matching its corresponding lower bound.

We first like to point out that maximum bipartite matching can be easily solved by a maximum flow algorithm using the standard reduction [32]. Hence an incremental algorithm for maintaining unit capacity maximum flow, also solves the problem of maintaining incremental maximum matching in bipartite graphs in the same bounds, i.e., amortized $O(n)$ update time. This matches the lower bound of $\Omega(n)$ given by Dahlgard [10] for bipartite matching. In this section, we show that the same upper bound can also be maintained for general graphs by using a *trivial* extension of

the Blossoms algorithm [13, 19] for finding an augmenting path, which is defined differently for maximum matching as follows.

Augmenting paths

Given any graph $G = (V, E)$, a matching $\mathcal{E}_M \subseteq E$ is a set of edges such that no two edges in $\mathcal{E}_M$ share an endpoint. The vertices on which some edge from $\mathcal{E}_M$ (also called *matched* edge) is incident is called a *matched* vertex. The remaining unmatched vertices are called *free* vertices. A given matching $\mathcal{E}_M$ is called the maximum matching if there does not exist an *augmenting path*, which is defined as follows: A *simple* path which starts and ends at a free vertex such that every odd edge on the path is a *matched* edge, and hence all the intermediate vertices are *matched* vertices. In case such a path p exists, it can be used to increase the cardinality of the matching $\mathcal{E}_M$, by removing all the *matched* edges on p from $\mathcal{E}_M$ and adding all the *unmatched* edges on p to $\mathcal{E}_M$.

C.1 Fully Dynamic Maximum Cardinality Matching

Computing the augmenting path starting from a free vertex v requires a single BFS traversal from v requiring $O(m)$ time, where on even layers the algorithm only explores *matched* edges. This results in trivial *folklore* fully dynamic maximum matching algorithm requiring $O(m)$ worst case update time as follows.

The algorithm essentially selects the end vertex v based on the update. In case of vertex insertion, the inserted vertex is chosen as v. In case of deletion of a vertex x, $\mathcal{E}_M$ is not updated in case x was a *free* vertex. Else if x was matched to y, we start the computation of augmenting path from y (as v). In case of insertion of an edge (x, y), $\mathcal{E}_M$ is not updated if both x and y are *matched*. Else, if both are *free* we simply add (x, y) to $\mathcal{E}_M$. Else the augmenting path is computed starting from the *free* vertex amongst x and y (as v). Finally, in case of deletion of an edge (x, y), $\mathcal{E}_M$ is not updated if (x, y) was not a matched edge. Else, the augmenting path is computed twice starting from x and y (as v) in the two computations. Thus, each update can be performed in $O(m)$ time resulting in the following theorem.

Theorem C.1 (Fully dynamic Maximum Cardinality Matching (folklore))**.** *Given a graph $G = (V, E)$ having n vertices and m edges, fully dynamic maximum cardinality matching under edge or vertex updates can be maintained in $O(m)$ worst case time per update.*

C.2 Incremental Maximum Cardinality Matching

In the incremental setting Maximum Cardinality Matching can be maintained in $O(|\mathcal{E}_M|)$ amortized time per update, where $|\mathcal{E}_M| = O(n)$. Note that it does not directly follow from the fully dynamic case, as we don't know the corresponding starting vertex of the augmenting path. However, for computing *augmenting paths* from any starting vertex in $O(m)$ time, we can use the standard Blossom's algorithm [13, 19]. It turns out that the Blossom's algorithm trivially extends to the incremental setting, such that it requires $O(m)$ time per increase in the cardinality of maximum matching. In the interest of completeness, we briefly describe the Blossom's algorithm as follows.

Blossom's Augmenting Path algorithm [13, 19]

The algorithm essentially maintains a tree from each free vertex, where each node is either a single vertex or a set of vertices which occur in form of a *blossom*. Further, it ensures that each vertex of the graph occurs in only one such tree. The vertices are called as *odd* or *even* if they occur respectively on the odd level or the even level (including free vertices in level *zero*) of any tree,

and *unvisited* otherwise. The children of *even* vertices are connected to them through *unmatched* edges, whereas those of *odd* vertices are connected through *matched* edges. Thus, all the paths from roots to leaves in such trees have alternating *unmatched* and *matched* edges, as required by an augmenting path.

The algorithm starts with all free vertices as *even* vertices on singleton trees, and the remaining vertices as *unvisited*. Then each edge (u, v) is considered one by one (and hence naturally extends to incremental setting) updating the trees accordingly as follows: If no end point is *even*, ignore the edge. Else, without loss of generality let u be an *even* vertex. If v is an *unvisited* vertex, simply add v as a child of u. Else, if v is an *odd* vertex, simply ignore it. However, if v is an *even* vertex, the action depends on the trees to which u and v belong. In case they belong to the same tree we get a *blossom*, which is processed as follows. Find the lowest common ancestor w of u and v, and shrink all the vertices on tree paths connecting w with u and v to a single *blossom* vertex w', to be placed in place of w. The children of the *even* vertices in this blossom, which are not a part of this blossom, become the children of w'. Also, now each *odd* vertex in this blossom, have become *even* so all its edges that were previously ignored are explored recursively using the same procedure. Note that the compressed blossoms need to be maintained using modified Disjoint Set Union algorithm [18] (taking overall $O(m)$ time), allowing a vertex v to quickly identify the blossom (and hence node w' in the tree) they belong to. Thus, an insertion of (u, v) is first evaluated to find nodes (vertices or blossoms) of the final tree u' and v' representing u and v, and the edge (u', v') is processed accordingly.

Finally, in case both u and v are *even* and they belong to different trees, and we have found an augmenting path from the root of the first tree to u, through (u, v), followed by the tree path from v to its root. However, some vertices on this path may be the *shrinked blossoms* and hence to report the augmenting path they are needed to be *un-shrinked*. An alternate simpler way is to simply start the augmenting path computation using the simple algorithm from the root of the tree containing v, as it is necessarily one end vertex of the augmenting path. Thus, both the *blossom's* algorithm and augmenting path reporting can be performed in total $O(m)$ time.

Algorithm

The algorithm prominently uses the fact that the maximum cardinality of a matching $|\mathcal{E}_M|$ is $O(n)$, since the each vertex can have at most one *matched* edge incident to it. The algorithm is again divided into $|\mathcal{E}_M|$ stages, where at the end of each stage the Blossom's algorithm detects an *augmenting path* and hence increases the cardinality of $\mathcal{E}_M$ by a single unit. Each stage starts by initiating the Blossom's algorithm [13, 19] from scratch in the updated matching, and continues inserting edges until the algorithm detects the augmenting path. Thus, each stage requires total $O(m)$ time for computing the augmenting path and updating $\mathcal{E}_M$, taking overall $O(m|\mathcal{E}_M|)$ time where $|\mathcal{E}_M| = O(n)$ is the value of the maximum cardinality matching of the graph, giving us the following result.

Theorem C.2 (Incremental Maximum Cardinality Matching)**.** *Given a graph $G = (V, E)$ having n vertices and m edges, incremental maximum cardinality matching can be maintained in amortized $O(|\mathcal{E}_M|)$ update time, where $|\mathcal{E}_M| = O(n)$ is the size of the maximum matching of the final graph.*

Remark: The algorithm also works for the case of incremental vertex updates. Also, Blossom's algorithm can also be used for fully dynamic updates having $O(m)$ update time. However, the algorithm described in Section C.1 is simpler and more intuitive.