# Dynamic MIS Revisited: Incremental, Fault Tolerant and Fully Dynamic

**Manoj Gupta**
Department of Computer Science and Engineering, Indian Institute of Technology Gandhinagar, India

**Shahbaz Khan**[1]
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

**Madhu Surendra**
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

## Abstract

Given a dynamic graph $G$, we aim to maintain a maximal independent set (MIS). This problem admits a deterministic solution in $O(m^{2/3})$ time [SOSA21], while randomized algorithms for oblivious adversary take $O(\text{poly}\log n)$ [FOCS19] time. Recently, Bernstein et al. [SODA26] proved a conditional lower bound of $\Omega(n^{1-o(1)})$ amortized update time even for incremental MIS with adaptive adversary. In this paper, we establish similar deterministic bounds through the lens of incremental and fault tolerant algorithms for MIS, and show the following:

1. **Incremental:** MIS can be maintained under edge insertions in $O(\sqrt{m})$ amortized update time.
2. **Fault Tolerant:** Using $O(m)$ preprocessing time, the MIS can be reported after $k$ edge deletions in $O(k+\overline{n}^2)$ time, where $\overline{n}$ is the number of vertices on which deleted edges are incident.
3. **Fully Dynamic:** MIS can be maintained under fully dynamic edge updates (insertions and deletions) in $O(m^{2/3})$ amortized update time.

Our incremental result is thus polynomially optimal even on allowing randomization with an adaptive adversary. Our fully dynamic result matches the state-of-the-art [SOSA21], and uses our incremental and fault tolerant algorithms as a black box. Although our fully dynamic algorithm does not improve the existing bounds, it provides additional insights into the harder edge-insertion case. We believe these insights may potentially be useful for improving the bounds for fully dynamic MIS in the future to match the incremental MIS bounds.

**2012 ACM Subject Classification** Theory of computation → Dynamic graph algorithms

**Keywords and phrases** Maximal Independent Set, MIS, Incremental, Fault Tolerant, Fully dynamic

**Digital Object Identifier** 10.4230/LIPIcs.SWAT.2026.21

**Related Version** The incremental MIS algorithm was previously reported [9] but not published [10].
*Previous Version*: https://arxiv.org/abs/1804.01823 [9]

**Acknowledgements** We would like to thank the anonymous reviewers for pointing out the matching lower bound [SODA26] for incremental MIS with adaptive adversaries, strengthening our result.

## 1 Introduction

Given a graph $G = (V, E)$ with $n$ vertices and $m$ edges, a set $\mathcal{M} \subseteq V$ forms an *independent set* if no two vertices in $\mathcal{M}$ share an edge, i.e., $\forall x, y \in \mathcal{M}, (x, y) \notin E$. A *maximal* independent set is one that cannot be extended by including any other vertex while preserving independence. Although computing a maximum cardinality independent set is NP-hard [8], a simple greedy algorithm computes a maximal independent set (MIS) in $O(m)$ time.

[1] Corresponding author

© Manoj Gupta, Shahbaz Khan, and Madhu Surendra;
licensed under Creative Commons License CC-BY 4.0

A graph is *incremental* if it gains edges over time, *decremental* if it loses edges, and *fully dynamic* if both insertions and deletions occur. In dynamic settings, one can trivially maintain a maximal independent set (MIS) in $O(m)$ update time by recomputing it from scratch after each update. Recent work has focused on improving this update time. Censor-Hillel et al. [5] presented the first non-trivial algorithm requiring $O(\Delta) = O(n)$ amortized update time, where $\Delta$ is the maximum degree of a vertex in the graph. Assadi, Onak, Schieber, and Solomon [1] improved it to an $O(m^{3/4})$ time algorithm to maintain an MIS. Du and Zhang [7] improved this update time to $\tilde{O}(m^{2/3})$[2] amortized, and independently Gupta and Khan [10] acheived $O(m^{2/3})$, all using deterministic algorithms. Du and Zhang [7] also proposed a randomized algorithm with $\tilde{O}(\sqrt{m})$ update time, which was later improved to $\tilde{O}(\sqrt{n})$ and $\tilde{O}(m^{1/3})$ by Assadi et al. [2]. Finally, Chechik and Zhang [6] and Behnezhad et al. [3] independently achieved an $\tilde{O}(1)$ update time using randomized methods. All known algorithms with $o(m^c)$ update time are randomized assuming an oblivious adversary. Recently, Bernstein et al. [4] showed that even for randomized algorithms with an adaptive adversary, assuming the combinatorial BMM conjecture, there exists a lower bound of $O(n^{1-o(1)})$ amortized update time for incremental MIS, which naturally extend to deterministic and fully dynamic variants.

In this paper, we focus on deterministic algorithms for dynamically maintaining MIS. Despite not improving the state-of-the-art fully dynamic algorithm, our aim is to present further insights into the problem by using an alternate approach, potentially paving a way for improving the theoretical bounds in the future. Our results are as follows:

As a first step, we present a deterministic algorithm for maintaining an MIS in an *incremental* graph, where only edge insertions occur. Our algorithm achieves an amortized $O(\sqrt{m})$ update time, improving over the current state-of-the-art [10].

▶ **Theorem 1** (Incremental MIS). *Given a graph $G$ with $n$ vertices, its MIS can be maintained under the insertion of $m$ edges in amortized $O(\sqrt{m})$ update time.*

▶ Remark 2. We also show that the upper bound of the incremental algorithm is tight in Appendix A. In the light of recent hardness result [4], our algorithm is polynomially optimal.

Next, we design a fault-tolerant algorithm for maintaining an MIS. In this setting, the graph $G(V, E)$ remains static, and we preprocess it to construct a data structure that can efficiently answer the following query:

FaultTolerant($F$): Compute an MIS of $G(V, E \setminus F)$, where $F$ is a set of edge failures.

We present a deterministic fault-tolerant algorithm with the following guarantee:

▶ **Theorem 3** (Fault Tolerant MIS). *Given a graph $G(V, E)$ with $n$ vertices, it can be preprocessed in $O(m)$ time such that for any set $F \subseteq E$ of $k$ edges, the MIS of $G(V, E \setminus F)$ can be reported in $O(k + \overline{n}^2)$ time, where $\overline{n}$ is the number of vertices incident to edges in $F$.*

▶ Remark 4. Since $\overline{n} = O(k)$, a simpler bound for the algorithm may be $O(k^2)$. However, we report the bound using $\overline{n}$ because $k$ can be $O(m) = O(n^2)$, but the time is bounded by $O(n^2)$ not $O(k^2) = O(n^4)$. We further restrict it to $O(\overline{n}^2)$ so that it can be directly used in our result as a black box, simplifying its analysis.

Finally, we combine the incremental and fault-tolerant algorithms to get the following:

▶ **Theorem 5** (Fully Dynamic MIS). *Given a graph $G(V, E)$ with $n$ vertices, its MIS can be maintained under fully dynamic edge updates in amortized $O(m^{2/3})$ update time, where $m$ is the current number of edges in $G$.*

[2] $\tilde{O}$ hides polylogarithmic factors in $n$

Although our algorithm does not improve the state-of-the-art theoretical bound, the new approach may present new insights to potentially improve the state-of-the-art in the future.

## 2 Preliminaries

Given a graph $G(V, E)$ having $n$ vertices and $m$ edges, which is connected ($m \geqslant n-1$). We represent the graph $G$ formed by deleting a set of edges $F \subseteq E$ from $G$ as $G(V, E \setminus F)$. Also, we represent a subgraph of $G$ induced by the vertices in $V'$ as $G(V')$ which refers to $G(V', E')$ where $E' = E \cap \{V' \times V'\}$. We represent the degree of a vertex $v$ by $\deg(v)$, i.e., the number of neighbours of a vertex $v$ in $G$. We also represent the maximum degree of a vertex in $G$ as $\Delta$. A set $M \subseteq V$ is *independent* if no two vertices in $M$ share an edge, i.e., $\forall x, y \in M, (x, y) \notin E$. A *maximal* independent $\mathcal{M}$ set is an independent set that cannot be extended by including any other vertex while preserving independence.

### Fully dynamic algorithm with amortized $O(\Delta)$ update time

We now review the basic $O(\Delta)$ [5] algorithm for maintaining a maximal independent set $\mathcal{M}$. Some of its procedures and results would be used as a black box in our algorithm.

The algorithm maintains a $\texttt{count}[u]$ for each vertex $u$, which contains the number of neighbours of $u$ in $\mathcal{M}$. Assuming we start with the empty graph, we set $\texttt{count}[u] = 0$ for every $u \in G$ and all the vertices are in $\mathcal{M}$. The algorithm maintains the following invariant

$$v \in \text{MIS}, \text{ iff } \texttt{count}[v] = 0 \tag{1}$$

On insertion of an edge $(u, v)$, InsertΔ (see Algorithm 1) checks if at least one of $u$ or $v$ is not in $\mathcal{M}$, in which case it simply updates its $\texttt{count}$ if required. Else, if both are in $\mathcal{M}$, it removes one (say $u$) from $\mathcal{M}$, reducing the $\texttt{count}$ of the neighbours of $u$. In case the count of any neighbour $w$ becomes $\texttt{zero}$, it is added to $\mathcal{M}$, and the $\texttt{count}$ of the neighbours of $w$ is increased. On deletion of an edge $(u, v)$, DeleteΔ simply updates the $\texttt{count}$ of $u$ or $v$ if required. In case the $\texttt{count}$ becomes $\texttt{zero}$, the corresponding vertex is added to $\mathcal{M}$, increasing the $\texttt{count}$ of its neighbours.

By construction, whenever an edge is inserted or deleted, or a vertex enters or leaves $\mathcal{M}$ its count is updated, and it is added to the $\mathcal{M}$ according to Invariant 1. Thus, the correctness of the algorithm follows from the invariant.

We now analyze the running time of the algorithm. The analysis hinges on the fact that both in InsertΔ$(u, v)$ and DeleteΔ$(u, v)$, at most one vertex is removed from $\mathcal{M}$ while multiple vertices can be potentially added to $\mathcal{M}$.

Let $\phi_i(u) = \deg(u)$ if $u \notin \mathcal{M}$, else 0 after the $i$-th update step of the algorithm. We define $\phi_i = \sum_u \phi_i(u)$. After initialization, $\phi_o(u) = 0$ for each $u \in V$ as in an empty graph, all the vertices are in $\mathcal{M}$. Thus, $\phi_o = 0$. Now, if DeleteΔ$(u, v)$ is called at the $i$-th update step. In the first step of this procedure, $(u, v)$ is removed from the graph. If $u \notin \mathcal{M}$ (or $v \notin \mathcal{M}$), this decreases the potential of $u$ (or $v$) by 1. In any case, the amortized cost of the first step is $O(1)$. Let us look at the main part of the algorithm when $w$ is added to $\mathcal{M}$. In this case, $w$ informs all its neighbors about its inclusion in $\mathcal{M}$, incurring a cost of $\deg(w)$. But its potential also decreases from $\deg(w)$ to 0. Thus, the amortized cost of processing $w$ is 0. This implies that the amortized running time of DeleteΔ$(u, v)$ is $O(1)$.

Now, if InsertΔ$(u, v)$ is called at the $i$-th update step when an edge $(u, v)$ is added to $G$. Addition of $(u, v)$ in the graph, may increase the potential of $u$ (or $v$) if $u \notin \mathcal{M}$ (or if $v \notin \mathcal{M}$) by 1. Thus, the amortized running time of the last step of InsertΔ$(u, v)$ is $O(1)$. The main

**Algorithm 1** Fully dynamic algorithm for MIS.

```
1  InsertΔ(u, v):
2  if u ∈ M or v ∈ M then
3      if u ∉ M then Exchange u and v;
4      if v ∉ M then
5          count[v] ← count[v] + 1
6      else
7          Remove u from M;
8          foreach neighbor w of u do
9              count[w] ← count[w] − 1;
10             if count[w] = 0 then
11                 Add w to M
12                 foreach neighbor x of w do
13                     count[x] ← count[x] + 1
14 Add (u, v) to the graph G;
15 DeleteΔ(u, v):
16 Remove (u, v) to the graph G;
17 if u ∈ M then
18     count[v] ← count[v] − 1
19 if v ∈ M then
20     count[u] ← count[u] − 1
21 foreach w ∈ {u, v} do
22     if count[w] = 0 then
23         Add w to M
24         foreach neighbor x of w do
25             count[x] ← count[x] + 1
```

part of the algorithm is when $u$ is removed from $M$ and many of its neighbours enter $M$. Let $w$ be a neighbor of $u$ that enters $M$. As discussed in the procedure DeleteΔ, the amortized cost of this operation is 0. Now, when $u$ is removed from $M$, the potential of $u$ increases to $\deg(u)$. Also, $u$ incurs $\deg(u)$ work to update the count of its neighbours. Thus, the amortized time to process $u$ is $O(\deg(u))$. Thus, the amortized time of InsertΔ$(u, v)$ is $O(\deg(u))$ if $u$ is removed from $M$ and is $O(1)$ otherwise, proving the following result.

▶ **Lemma 6.** *For a given graph $G(V, E)$, having $m$ edges and $n$ vertices, the amortized update time of an algorithm is $O(\deg(u))$ if vertex $u$ leaves the MIS $M$, and $O(1)$ otherwise.*

Since in both InsertΔ and DeleteΔ, at most one vertex (say $u$) leaves the MIS $M$ in each update, the amortized update time is bounded by $O(\deg(u)) \leqslant O(\Delta)$. This proves the amortized $O(\Delta)$ update time of the algorithm.

## 3 Incremental MIS

We now consider the problem of maintaining an MIS in an incremental graph. We refer to a vertex $v$ as *light* if $\deg(v) \leqslant \Delta_l$ and *heavy* otherwise, where $\Delta_l$ shall be computed later. Since the graph is incremental, all vertices start as light, and once a vertex becomes heavy, it remains heavy. Note that this distinction is required only for the analysis and not the algorithm. Similar to the $O(\Delta)$ algorithm, each vertex $v$ maintains the count of its neighbours in the $M$ as $\mathsf{count}[v]$.

Now, our algorithm matches the insert procedure InsertΔ exactly except for one crucial modification (see Algorithm 2). On insertion of an edge $(u, v)$, if both the vertices are in $M$, instead of *arbitrarily* removing a vertex $u$ from $M$, we perform a *degree biased removal*, i.e., we remove the vertex $u$ having lower degree $\deg(u)$. Since the remaining procedure exactly matches InsertΔ, it correctness also follows from the correctness of $O(\Delta)$ algorithm.

**Algorithm 2** INCREMENTAL$(u, v)$.

```
1  if u ∈ M or v ∈ M then
2      if u ∉ M then Exchange u and v;
3      if v ∉ M then count[v] ← count[v] + 1 ;
4      else
5          if deg(u) > deg(v) then Exchange u and v;
6          Remove u from M;
7          foreach neighbor w of u do
8              count[w] ← count[w] − 1;
9              if count[w] = 0 then
10                 Add w to M
11                 foreach neighbor x of w do
12                     count[x] ← count[x] + 1
13 Add (u, v) to the graph G;
```

We now analyze the running time of our algorithm. Lemma 6 shows that the amortized update time of an update is $O(\deg(u))$ if a vertex $u$ leaves $M$. Moreover, in each update, at most one vertex can leave $M$. Let the $i^{th}$ edge update inserts the edge $(u_i, v_i)$, such that $u_i$ leave $M$ (if any). We analyze the total time taken by a vertex over two *phases*: when it is *light* ($\deg(u_i) \leqslant \Delta_l$) and when it becomes *heavy* ($\deg(u_i) > \Delta_l$). If $u_i$ is light, then we remove it from $M$ in $O(\Delta_l)$ amortized time, since $\deg(u_i) \leqslant \Delta_l$. If $u_i$ is heavy, then $v_i$ must also be heavy since $u_i$ was removed from $M$ only because $\deg(v_i) \geqslant \deg(u_i) \geqslant \Delta_l$. Let the number of times a heavy vertex leaves $M$ be $n_h$. Since a heavy $u_i$ can leave $M$ only when $v_i$ is heavy, $n_h \leqslant O(\frac{m}{\Delta_l})$ as there are $O(\frac{m}{\Delta_l})$ heavy vertices in the final graph. Therefore, $u_i$ can be removed from $M$ at most $O(n_h) = O(\frac{m}{\Delta_l})$ times due to its heavy neighbors.

Combining both phases, we now compute the total amortized update time for $m$ insertions.

$$\begin{aligned}
\sum_{i=1}^{m} O(\deg(u_i)) &= \sum_{i:u_i \text{ is light}} O(\deg(u_i)) + \sum_{i:u_i \text{is heavy}} O(\deg(u_i)) \\
&\leqslant \sum_{i=1}^{m} O(\Delta_l) + \textstyle\sum_{u \in V} n_h \times O(\deg(u)) \\
&\leqslant O(m\Delta_l) + O(mn_h)
\end{aligned}$$

Hence, the total amortized update time of our incremental algorithm is $O(m(\Delta_l + n_h))$, giving amortized $O(\Delta_l + n_h)$ update time, resulting in the following lemma:

▶ **Lemma 7.** *Maintaining the MIS in the incremental setting requires amortized* $O(\Delta_l + n_h)$ *update time, where* $\Delta_l$ *is the maximum degree of a light vertex, and* $n_h$ *is the number of times a heavy vertex leaves* $M$*.*

Since $n_h = O(\frac{m}{\Delta_l})$, substituting $\Delta_l = \sqrt{m}$ gives us the following result.

▶ **Theorem 1** (Incremental MIS)**.** *Given a graph* $G$ *with* $n$ *vertices, its MIS can be maintained under the insertion of* $m$ *edges in amortized* $O(\sqrt{m})$ *update time.*

▶ Remark 8. We show that the upper bounds for the $O(\Delta)$ algorithm and our incremental algorithm are tight in Appendix A.

## 4 Fault Tolerant MIS

We now describe our fault tolerant algorithm. It follows from the simple delete procedure *Delete*$\Delta$ of the $O(\Delta)$ algorithm with a careful analysis. Given that static graph $G(V, E)$, we preprocess it to build a data structure to answer the following query.

FaultTolerant($F$): Compute an MIS of $G(V, E \setminus F)$, where $F$ is a set of edge failures.

Our fault tolerant algorithm preprocesses the graph to compute the maximal independent set $\mathcal{M}$ of $G$ and $\texttt{count}[v]$ for every $v \in V$. These can be computed in $O(m)$ time using the static algorithm. The pseudocode for the query algorithm is shown in Algorithm 3.

Let the end vertices of the edges in $F$ be $V_F$. The deletion of edges in $F$ can potentially reduce the $\texttt{count}[v]$ for a vertex $v \in V_F$ if the other end vertex is in $\mathcal{M}$. In case this count becomes zero, the vertex may be potentially added to $\mathcal{M}$. Let the set of such vertices be $V_d$. Since the vertices in $V_d$ may not be independent, we cannot simply add all the vertices in $V_d$ to $\mathcal{M}$. Instead, we add the MIS $\mathcal{M}$' of the subgraph $G(V_d)$ to $\mathcal{M}$. However, since we are dealing with the fault tolerant model, we do not modify the $\texttt{count}[\cdot]$ and $\mathcal{M}$ directly. We instead make a copy $c[v]$ of $\texttt{count}[v]$, which is modified, so that the changes need not be reversed after the query. Thus, our data structure can be directly used for the next query $F'$.

**Algorithm 3** FaultTolerant($F$): Return vertices added to $\mathcal{M}$ after deleting the edges $F$.

```
1 V_F ← Vertices incident on F;
2 foreach v ∈ V_F do c[v] ← count[v];
3 foreach (u, v) ∈ F do
4     if u ∈ M then c[v] ← c[v] − 1;
5     if v ∈M then c[u] ← c[u] − 1;
6 foreach v ∈ V_F do
7     if c[v] = 0 then Add v to V_d;
8 M'← MIS of G(V_d);
9 Return M ∪M'
```

Let $\overline{n}$ be the number of vertices on which the edges in $F$ are incident, hence $|V_F| = \overline{n}$. Now, $V_F$ can be computed in $O(k)$ time, $c[v]$ can be initialized in $O(\overline{n})$ time and updated in $O(k)$ time while processing each edge in $F$. Thereafter, $V_d$ can be computed in $O(\overline{n})$ time where $V_d \subseteq V_F$ hence $|V_d| = O(\overline{n})$. The MIS on $G(V_d)$ can computed in $O(|V_d|^2) = O(\overline{n}^2)$ time. We thus get the following result for fault tolerant MIS.

▶ **Theorem 3** (Fault Tolerant MIS). *Given a graph $G(V, E)$ with $n$ vertices, it can be preprocessed in $O(m)$ time such that for any set $F \subseteq E$ of $k$ edges, the MIS of $G(V, E \setminus F)$ can be reported in $O(k + \overline{n}^2)$ time, where $\overline{n}$ is the number of vertices incident to edges in $F$.*

## 5 Fully dynamic MIS without rebuilding

We now describe our fully dynamic algorithm for MIS requiring $O(\min\{\Delta, m^{2/3}\})$ time which matches the state-of-the-art algorithm [10]. While the previous algorithm was based on $O(\Delta)$ algorithm [1] and the static algorithm, our algorithm is based on the incremental and fault tolerant algorithms described earlier.

At its core, our algorithm classifies a vertex $v$ as *light* if $\deg(v) \leqslant m^{2/3}$, and *heavy* otherwise (similar to [10]). All insertions are performed using the incremental algorithm, while the deletions are performed based on the incident vertices. In case at least one of the incident vertices is *light*, we use the delete procedure Delete$\Delta$ from the basic $O(\Delta)$ algorithm. However, if both incident vertices are heavy, we collect the deleted edges in $F$ and use our fault tolerant algorithm to report the solution after every update. This essentially

handles the harder case of edge insertion between two heavy vertices, as handling their deletion limits such insertions similar to the incremental algorithm. If such an edge (deleted and stored in $F$) is inserted back into the graph, it is simply removed from $F$, and the fault tolerant algorithm updates $\mathcal{M}$ accordingly (see pseudocode in Algorithm 4).

**Algorithm 4** Fully dynamic MIS using incremental and fault tolerant MIS algorithms.

```
1  Insert(u, v):
2  if (u, v) ∈ F then
3  |  Delete (u, v) from F;
4  else Incremental(u, v) ;
5  M" ← FaultTolerant(F);
6  Report M ∪M";
7  Delete(u, v):
8  if both u and v are heavy then
9  |  Add (u, v) in F;
10 else DeleteΔ(u, v) ;
11 M" ← FaultTolerant(F);
12 Report M ∪M";
```

The main intuition behind the time complexity is to limit the number of times a heavy vertex $v$ leaves the MIS in the incremental algorithm. This is bounded by the number of edges inserted between $v$ and a heavy vertex. In the fully dynamic setting, we avoid repeated insertions of such edges using the fault tolerant algorithm to process their deletions temporarily without affecting the main algorithm. Thus, given $\Delta_l = m^{2/3}$ and $n_h = m^{1/3}$ (as an edge to each of $m^{1/3}$ heavy vertex can be inserted exactly once), the edge insertions can be processed in amortized $O(m^{2/3})$ time (Lemma 7). Now, the edges collected in $F$ are incident on both *heavy* vertices, resulting in $k = m^{2/3}$ and $\overline{n} = m^{1/3}$, allowing the fault tolerant algorithm to be processed in $O(m^{2/3})$ time (Theorem 3). Thus, the edge deletions can be processed in $O(m^{2/3})$ amortized time requiring $O(1)$ amortized if at least one incident vertex is light (Lemma 6, as no vertex leaves $\mathcal{M}$) and $O(m^{2/3})$ for the fault tolerant algorithm. This results in the following fully dynamic algorithm.

▶ **Theorem 5** (Fully Dynamic MIS). *Given a graph $G(V, E)$ with $n$ vertices, its MIS can be maintained under fully dynamic edge updates in amortized $O(m^{2/3})$ update time, where $m$ is the current number of edges in $G$.*

▶ Remark 9. The algorithm distinguishes the vertices as heavy or light based on a degree threshold $c = m^{2/3}$. However, the value of $m$ keeps changing in a fully dynamic graph. We thus initialize the threshold $c_0 = m_0^{2/3}$, where $m_0$ is the initial number of edges. We keep the same threshold while $\frac{m_0}{2} \leqslant m \leqslant 2m_0$. In case $m$ passes this range, we rebuild the MIS with $F = \emptyset$ and $m_0 = m$. The total cost of the algorithm $O(m \times m^{2/3})$ can be amortized among the $O(m_0)$ updates after which $m$ exceeded the bounds, requiring amortized $O(m^{2/3})$ time.

▶ Remark 10. The fully dynamic setting can also potentially change the status of a vertex between light and heavy. In case a vertex $v$ converts from light to heavy, no update is required as newly deleted edges on $v$ will be added to $F$. However, in case a vertex converts from heavy to light, all its edges in $F$ need to be deleted from the graph $G$, requiring potentially $O(m^{2/3})$ amortized time per deletion. We thus relax the criteria, converting a light vertex $v$ to heavy if $\deg(v) > m^{2/3}$ but converting it to light when $\deg(v) < \frac{m^{2/3}}{2}$. This allows to charge $O(m^{2/3})$ for the deletion of its $O(m^{1/3})$ edges to heavy vertices during the $O(m^{2/3})$ updates when $\frac{m^{2/3}}{2} < \deg(v) < m^{2/3}$ resulting in amortized $O(m^{1/3})$ time per update.

▶ Remark 11. Though our proposed algorithm divides the vertices into light and heavy, similar to the state-of-the-art algorithm [10], its handling of heavy updates avoids rebuilding the MIS for heavy vertices using a simpler fault tolerant algorithm. Thus, our algorithm is expected to perform better in practice for edge updates between heavy vertices.

## 6 Conclusion

The dynamic MIS problem is one of the unique problems in which handling edge insertions is apparently harder than edge deletions. For most classical graph problems, such as connectivity, reachability, shortest paths, etc., handling edge deletions is harder than edge insertions. Hence, given a faster algorithm for incremental MIS requiring $O(\sqrt{m})$ amortized update time, we are hopeful to achieve the same bound for the fully dynamic case. Further, given the lower bound of $O(n^{1-o(1)})$ for incremental MIS, a polynomial improvement is not possible even for a randomized algorithm with an adaptive adversary.

In this paper, we highlighted an alternate way to handle insertions faster. We also presented a fully dynamic algorithm utilizing insights from the incremental and fault tolerant algorithm, although without improving the state-of-the-art. We believe that new insights may be drawn from our work, potentially resulting in a faster fully dynamic algorithm.

## References


1. 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. doi:10.1145/3188745.3188922.
2. Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. 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 1919–1936. SIAM, 2019. doi:10.1137/1.9781611975482.116.
3. Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00032.
4. Aaron Bernstein, Sayan Bhattacharya, Nick Fischer, Peter Kiss, and Thatchaphol Saranurak. Separations between oblivious and adaptive adversaries for natural dynamic graph problems. In Kasper Green Larsen and Barna Saha, editors, *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 5669–5687. SIAM, 2026.
5. 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. doi:10.1145/2933057.2933083.
6. Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 370–381. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00031.
7. Yuhao Du and Hengjie Zhang. Improved algorithms for fully dynamic maximal independent set. *arXiv preprint arXiv:1804.08908*, 2018. arXiv:1804.08908.
8. 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.
9. Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018. arXiv:1804.01823.
10. Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set, maximum flow and maximum matching. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 86–91. SIAM, 2021. doi:10.1137/1.9781611976496.10.

# A Tightness of the Incremental Algorithm

We now present worst-case examples demonstrating the tightness of the analysis of $O(\Delta)$ algorithm 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.

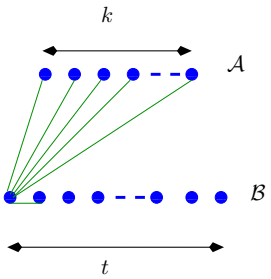


**Figure 1** Worst case example for arbitrary removal.

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 $\Delta$ 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 Lemma 6, 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 $O(\Delta)$ algorithm in incremental (and hence fully dynamic) setting.

▶ **Theorem 12.** *For each value of $n \leqslant m \leqslant \binom{n}{2}$ and $1 \leqslant \Delta \leqslant n$, there exists a sequence of $m$ edge insertions where the degree of each vertex is bounded by $\Delta$ for which $O(\Delta)$ algorithm 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 $\mathcal{M}$.

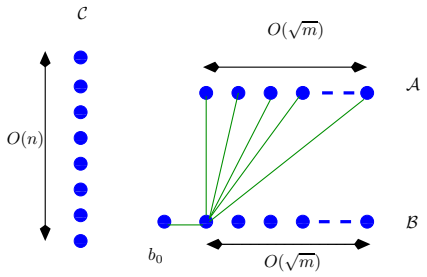


**Figure 2** Tightness Example for degree biased removal.

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 the 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 Lemma 6, 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 13.** *For each value of $n \leqslant m \leqslant \binom{n}{2}$, 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 the 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, $|\mathcal{B}| \leqslant \sqrt{m}$ 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 $\Delta$. As evident from the previous example in the absence of such a restriction, the amortized time can be raised to $\Delta$, implying the significance of the degree biased removal in the incremental algorithm.