Fast Maximal Independent Sets on Dynamic Graphs

I. Preliminaries

To prove the correctness of our algorithms, it is essential to understand some fundamental definitions. These definitions are crucial as they outline the key properties that our algorithms must maintain during edge insertions and deletions to ensure accurate and deterministic results.

A. *Preliminary 1: Independent Set Property*

An independent set Δ in a graph $G = (V, E)$ is defined as a set of vertices such that there does not exist any pair of vertices $x, y \in \Delta$ where $(x, y) \in E$. Mathematically, for a graph $G = (V, E)$, an independent set Δ satisfies:

$$\forall x, y \in \Delta, (x, y) \notin E$$

This means that for every pair (x, y) where x and y belong to Δ, there is no edge between them.

B. *Preliminary 2: Maximal Independent Set Property*

A maximal independent set (MIS) Δ_m in a graph $G = (V, E)$ is defined as an independent set where no other vertex from V can be added to Δ_m without violating the independence property. Mathematically, for a graph $G = (V, E)$, an independent set Δ_m is maximal if and only if:

$$\forall v \in V \setminus \Delta_m, \exists u \in \Delta_m, (u, v) \in E$$

which states that for every v that is not a member of Δ_m, there exists a u in Δ_m such that (u, v) is an edge in E, indicating that v cannot be added to Δ_m while maintaining independence.
In other words, every non-MIS member has at least one of its neighbours in the MIS.

These preliminaries will be used to prove the correctness of our algorithms in the following appendices.

Appendix A
Proofs

A. *Proof of correctness for insertions*

In all the given proofs below, we are checking two parameters which are:
1. *Independency*: $\forall x, y \in \Delta_m^{t+}, (x, y) \notin E$.
2. *Maximality*: $\forall w \in V \setminus \Delta_m^{t+}, \exists t \in \Delta_m^{t+}$ such that $(w, t) \in \Delta_m^{t+}$.

For all **Lemmas**, we assume $G = (V, E)$ is a simple graph and $\Delta_m^{t-} \subseteq V$ is a maximal independent set (MIS) in G, provided as input. A copy of Δ_m^{t-} is made as Δ_m^{t+}, and updates are performed over Δ_m^{t+}, which is returned as the updated MIS. In each of the **Lemmas**, we aim to prove that the updated Δ_m^{t+} holds the required properties.

Lemma 1. *For an incoming edge (u, v), if both u and v belong to Δ_m^{t+}, removing either vertex and checking the possibility of their neighbours becoming Δ_m^{t+} members preserves independence and maximality.*

Proof. Consider an incoming edge (u, v) where both u and v belong to Δ_m^{t+}. To maintain the independence property, we must remove either u or v from Δ_m^{t+} because if we do not remove either vertex, there will be an edge $(u, v) \in E$ with both u and v in Δ_m^{t+}, which contradicts Preliminary I-A.

To preserve maximality, we then check whether any neighbour N_x of the removed vertex x (either u or v) can be included in the new independent set without violating independence. Since u and v were in Δ_m^{t+}, none of their neighbours were previously included in Δ_m^{t+} due to the independence property.

If a neighbour N_x of the removed vertex x can be included in Δ_m^{t+} without violating the independence property (i.e., N_x does not have any neighbour in Δ_m^{t+}), then it is added to Δ_m^{t+}. This ensures that the new set remains maximal. By iterating through all neighbours of the removed vertex and potentially including them in Δ_m^{t+}, we can say that the resulting set maintains both independence and maximality. □

Lemma 2. *For an incoming edge (u, v), if either u or v is a member of Δ_m^{t+}, we refrain from making any updates, preserving independence and maximality.*

Proof. Since Δ_m^{t+} is an independent set, we know that for any $x, y \in \Delta_m^{t+}$, $(x, y) \notin E$.

Consider the scenario where $u \in \Delta_m^{t+}$ and the new edge (u, v) is added to the graph. This addition does not violate the independence property of Δ_m^{t+} because $v \notin \Delta_m^{t+}$. The newly added edge (u, v) means u has a new neighbour v, but since v is not in Δ_m^{t+}, the independence property is preserved. Similarly, if $v \in \Delta_m^{t+}$ and $u \notin \Delta_m^{t+}$, the addition of the edge (u, v) does not violate the independence property.

From Preliminary I-B, for every vertex $w \in V \setminus \Delta_m^{t+}$, there exists a vertex $t \in \Delta_m^{t+}$ such that $(w, t) \in E$. Consider the scenario where $u \in \Delta_m^{t+}$. In this case, v cannot be added to Δ_m^{t+} without violating the independence of the set because v, like other non-Δ_m^{t+} members, has at least one neighbour in Δ_m^{t+}, which is u. Therefore, even after the edge insertion, $\forall w \in V \setminus \Delta_m^{t+}$, $\exists t \in \Delta_m^{t+}$ such that $(w, t) \in E$. Consequently, both properties are conserved during this step. □

Lemma 3. *For an incoming edge (u, v), if neither u nor v is in Δ_m^{t+}, no updates are necessary, and independence and* *maximality are preserved.*

Proof. Considering the cases where both $u, v \notin \Delta_m^{t+}$. From Preliminary I-B, we know that at least one of their neighbours is in Δ_m^{t+}. Hence, they cannot be added to Δ_m^{t+}, which maintains the independence property of Δ_m^{t+}.

Even after the insertion of the edge (u, v), neither u nor v can be added to Δ_m^{t+} without violating the independence property. Therefore, this still maintains the fact that $\forall w \in V \setminus \Delta_m^{t+}, \exists t \in \Delta_m^{t+}$ such that $(w, t) \in E$, which satisfies Preliminary I-B. ☐

B. Proof of Correctness for Deletions

In all the proofs below, we will verify two properties as we did for the insertion cases:

1. *Independency*: $\forall x, y \in \Delta_m^{t+}, (x, y) \notin E$.
2. *Maximality*: $\forall w \in V \setminus \Delta_m^{t+}, \exists t \in \Delta_m^{t+}$ such that $(w, t) \in E$.

Let $N(u) \subseteq V$ and $N(v) \subseteq V$ denote the sets of neighbors of vertices u and v in G, respectively.

Lemma 4. *For a deleted edge (u, v), if either u or v is in Δ_m^{t+}, we check the possibility of adding the other one to Δ_m^{t+}.*

Proof. Let us assume $u \in \Delta_m^{t+}$ and let $N(v) \subseteq V$ denote the set of neighbours of vertex v in G. Considering the relationship between $N(v)$ and Δ_m^{t+}, we have two cases to analyze:

Case 1: $\exists w \in N(v) \cap \Delta_m^{t+} \Rightarrow v \notin \Delta_m^{t+}$ (to maintain independence).

If there exists a vertex $w \in N(v) \cap \Delta_m^{t+}$, then adding v to Δ_m^{t+} would violate the independence property, as $(v, w) \in E$. In this scenario, we refrain from modifying Δ_m^{t+}, ensuring its continued independence.

Case 2: $N(v) \cap \Delta_m^{t+} = \emptyset \Rightarrow v \in \Delta_m^{t+}$ (to achieve maximality).

If $N(v) \cap \Delta_m^{t+} = \emptyset$, meaning none of v's neighbours is in Δ_m^{t+}, then adding v to Δ_m^{t+} would not violate independence, since v has no edges with any current Δ_m^{t+} members. According to the definition of Δ_m^{t+}, for every vertex $v \notin \Delta_m^{t+}$, there must exist a neighbour $u \in \Delta_m^{t+}$ such that $(u, v) \in E$. The absence of any Δ_m^{t+} neighbour for v contradicts the maximality property. Therefore, we must add v to Δ_m^{t+} to ensure it remains maximal.

Therefore, when none of the neighbours of v is a member of Δ_m^{t+} after the edge deletion, we must add v to Δ_m^{t+} to maintain maximality. ☐

Lemma 5. *For a deleted edge (u, v), if neither u nor v is in Δ_m^{t+}, we do not modify Δ_m^{t+}.*

Proof. Since neither u nor v is in Δ_m^{t+}, we leverage the property from Preliminary I-B:

$$\forall x \notin \Delta_m^{t+}, \exists y \in \Delta_m^{t+} \text{ such that } (x, y) \in E$$

This means at least one neighbor of each of u and v must be in Δ_m^{t+} (i.e., $N(u) \cap \Delta_m^{t+} \neq \emptyset$ and $N(v) \cap \Delta_m^{t+} \neq \emptyset$). Therefore, adding either u or v to the MIS would violate the independence property because they have neighbours in Δ_m^{t+}. Hence, to preserve the property of independence, we do not modify the Δ_m^{t+}.

Additionally, removing an edge between u and v (neither of whom is in Δ_m^{t+}) does not affect the maximality property. This is because the relationship between existing Δ_m^{t+} members and non-members remains unchanged. Thus, for all $x \notin \Delta_m^{t+}$, there still exists $y \in \Delta_m^{t+}$ such that $(x, y) \in E$, maintaining the maximality of Δ_m^{t+}. ☐

Lemma 6. *For a deleted edge (u, v), there is no case where both $u \in \Delta_m^{t+}$ and $v \in \Delta_m^{t+}$.*

Proof. From Preliminary I-A, we know that for any two vertices x and y in Δ_m^{t+}, $(x, y) \notin E$ (i.e., Δ_m^{t+} is independent).

Given that (u, v) is a deletion edge, it implies that the edge (u, v) was previously present in the graph G and is now being removed.

If both u and v were in Δ_m^{t+}, there would be an edge between them, contradicting the independence property. Since this situation violates the fundamental property of an MIS, the scenario where both $u \in \Delta_m^{t+}$ and $v \in \Delta_m^{t+}$ cannot occur. ☐

Appendix B
Time complexity analysis

A. Parallel Dynamic MIS Update for Insertions in the incremental setting (P-DMI_{INC})

Algorithm 1 P-$DMI_{\text{INC}}(G, B, \Delta_m^{t-})$

Require: G (Graph), B (Batches of updates), Δ_m^{t-} (input MIS)
Ensure: Δ_m^{t+} (Updated MIS after deletions)

$\Delta_m^{t+} \leftarrow \Delta_m^{t-}$
for $b_i \in B$ **do**
  for $(u, v) \in b_i$ **do**
    $v_{\text{in}}, u_{\text{in}} \leftarrow$ false
    for $w \in \Delta_m^{t+}$ **parallel do**
      if $u = w$ **then**
        $u_{\text{in}} \leftarrow$ true
      if $v = w$ **then**
        $v_{\text{in}} \leftarrow$ true
    if $u_{\text{in}} \land v_{\text{in}}$ **then**
      $vertex \leftarrow \min(u, v)$
      Δ_m^{t+}.erase(vertex)
      $N \leftarrow$ getNeighbours$(G, vertex)$
      for $N_i \in N$ **do**
        $shouldWeAdd \leftarrow$ true
        $NofN_i \leftarrow$ getNeighbours(G, N_i)
        for $N_j \in NofN_i$ **do**
          for $w \in \Delta_m^{t-}$ **do**
            if $N_j = w$ **then**
              $shouldweAdd \leftarrow$ false
        if $shouldWeAdd$ **then**
          Δ_m^{t+}.push_back(N_i)

In the sequential case, the time complexity of the insertion algorithm for one batch can be expressed as $T_{\text{seq}}(n, m, k) =$

$O(k^2 \cdot m \cdot n)$, where n represents the number of vertices in the Δ_m^{t+}, m represents the number of edges in the batch, and k represents the average degree of a vertex.

The "for" loop in Line 2 iterates over the batches, and for each batch, the inner "for" loop (Line 3) iterates over the m edges. For each edge, the nested loop (Line 5) checks the Δ_m^{t+} membership of neighbouring vertices, which requires iterating over the entire set of size n. In the worst case, if the case occurs where the edge is inserted between two texts Δ_m^{t+} members, we must check the neighbours of all neighbours of the vertex to be removed to ensure a maximal set. Therefore, the overall complexity in the sequential case is $O(k^2 \cdot m \cdot n)$.

Assuming p threads are available to check the membership of vertices in Δ_m^{t+} concurrently. The complexity will be $T_{\text{par}}(n,m,k) = O\left(k^2 \cdot m \cdot \left(\frac{n}{p} + \log p\right)\right)$, where n denotes the number of vertices in the graph, m signifies the number of edges in the batch, and k is the average degree of a vertex. Each thread processes $\frac{n}{p}$ vertices, and the final step of thread synchronization introduces an additional overhead of $\log p$, assuming an efficient parallel reduction operation.

B. *Parallel Dynamic MIS Update for Insertions in the batched setting ($P\text{-}DMI_{BAT}$)*

Algorithm 2 $P\text{-}DMI_{\text{BAT}}$(G, B, Δ_m^{t-})

Require: G (Graph), B (Batches of updates), Δ_m^{t-} (input MIS)
Ensure: Δ_m^{t+} (Updated MIS)
1: $\Delta_m^{t+} \leftarrow \Delta_m^{t-}$
2: **for** $b_i \in B$ **do**
3:   **for** $(u,v) \in b_i$ **parallel do**
4:     $v_{\text{in}}, u_{\text{in}} \leftarrow$ false
5:     **for** $w \in \Delta_m^{t+}$ **do**
6:       **if** $u = w$ **then**
7:         $u_{\text{in}} \leftarrow$ true
8:       **if** $v = w$ **then**
9:         $v_{\text{in}} \leftarrow$ true
10:     **if** $u_{\text{in}} \land v_{\text{in}}$ **then**
11:       $vertex \leftarrow \min(u,v)$
12:       Δ_m^{t+}.mark(vertex) ▷ Mark as -1
13:       $N \leftarrow$ getNeighbours$(G, vertex)$
14:       **for** $N_i \in N$ **do**
15:         $shouldWeAdd \leftarrow$ true
16:         $NofN_i \leftarrow$ getNeighbours(G, N_i)
17:         **for** $N_j \in NofN_i$ **do**
18:           **for** $w \in \Delta_m^{t+}$ **do**
19:             **if** $N_j = w$ **then**
20:               $shouldweAdd \leftarrow$ false
21:         **if** $shouldWeAdd$ **then**
22:           Δ_m^{t+}.push_back(N_i)

In contrast to Algorithm 1, where multiple threads concurrently process a single edge at a time. Algorithm 2 partitions the batch of updates among the available threads, each handling a subset independently. Each thread in the parallel case processes $\frac{m}{p}$ edges independently, resulting in a parallel time complexity of $T_{\text{par}}(n,m,k) = O(k^2 \cdot \frac{m}{p} \cdot n)$. Since the updates occur in shared memory, synchronization barriers are unnecessary, eliminating the $\log p$ term. Thus, $P\text{-}DMI_{\text{BAT}}$ efficiently utilizes available threads to achieve significant parallel speedup.

C. *Parallel Dynamic MIS Update for Insertions in the batched setting using TVB data structure $P\text{-}DMI_{TVB}$*

Algorithm 3 $P\text{-}DMI_{\text{TVB}}$(T, B)

Require: T^- (TVB data structure storing both graph and Δ_m^{t-} membership of vertices), B (Batches of updates)
Ensure: T^+ (Updated TVB storing both graph and Δ_m^{t+} membership of vertices)
1: $T^+ \leftarrow T^-$
2: **for** $b_i \in B$ **do**
3:   **for** $(u,v) \in b_i$ **parallel do**
4:     $v_{\text{in}}, u_{\text{in}} \leftarrow$ false
5:     **if** $T^+[u].Membership$ **then**
6:       $u_{\text{in}} \leftarrow$ true
7:     **if** $T^+[v].Membership$ **then**
8:       $v_{\text{in}} \leftarrow$ true
9:     **if** $u_{\text{in}} \land v_{\text{in}}$ **then**
10:       $vertex \leftarrow \min(u,v)$
11:       $T^+[vertex].Membership \leftarrow$ false
12:       $N \leftarrow$ getNeighbours$(T, vertex)$
13:       **for** $N_i \in N$ **do**
14:         $shouldWeAdd \leftarrow$ true
15:         $NofN_i \leftarrow$ getNeighbours(G, N_i)
16:         **for** $N_j \in NofN_i$ **do**
17:           **if** $T^+[N_j].Membership$ **then**
18:             $shouldWeAdd \leftarrow$ false
19:         **if** $shouldWeAdd$ **then**
20:           $T^+[N_i].Membership \leftarrow$ true

TVB stores Δ_m^{t+} membership of vertices, eliminating the search space bottleneck (Lines 5-19 of Algorithm 2). Consequently, with TVB, the time complexity of the $P\text{-}DMI_{\text{TVB}}$ algorithm is reduced to $T(m,k) = O(k^2 \cdot \frac{m}{p})$. However, the second bottleneck, involving the evaluation of potential neighbours to be added to Δ_m^{t+} after a vertex removal, remains unresolved. This process involves two nested "for" loops, making it costly for graphs with high average degrees and dense structures (Lines 13-20 of Algorithm 3).

D. *Parallel Dynamic MIS Update for Insertion in the batched setting using $TVBL$ data structure $P\text{-}DMI_{TVBL}$*

The tuples used in $TVBL$ store the information of the vertex in the form of (x, (y, z) tuples, where x points to the adjacency list, y is the membership in Δ_m, and z is the level information, respectively. Algorithm 4 is based on the $TVBL$ data structure where we can see that we have only one for loop (Line 13) in the if condition (Line 9). This is because the need for iteration over neighbours of neighbours is now

Algorithm 4 $P\text{-}DMI_{\text{TVBL}}(T, B)$

Require: T^{-} (TVBL data structure storing both graph, Δ_m^{t-} membership of vertices and levels of the nodes), B (Batches of updates)
Ensure: T^{+} (Updated TVBL storing both graph, Δ_m^{t+} membership of vertices and levels of the nodes)

$T^{+} \leftarrow T^{-}$
for $b_i \in B$ **do**
  for $(u, v) \in b_i$ parallel **do**
    $v_{\text{in}}, u_{\text{in}} \leftarrow$ false
    if $T^{+}[u].Membership$ **then**
      $u_{\text{in}} \leftarrow$ true
    if $T^{+}[v].Membership$ **then**
      $v_{\text{in}} \leftarrow$ true
    if $u_{\text{in}} \wedge v_{\text{in}}$ **then**
      $vertex \leftarrow \min(u, v)$
      $T^{+}[vertex].Membership \leftarrow$ false
      $N \leftarrow$ getNeighbors$(T, vertex)$
      for $N_i \in N$ **do**
        if $T^{+}[N_i].Level = 1$ **then**
          $T^{+}[N_i].Membership \leftarrow$ true

eliminated by the level information available in the $TVBL$. Thus resulting in the single "for" loop with the overall time complexity of $T(m, k) = O(k \cdot \frac{m}{p})$.

E. Parallel Dynamic MIS Update for Deletion in the batched setting using TVB data structure $P\text{-}DMD_{TVB}$

Algorithm 5 $P\text{-}DMD_{\text{TVB}}$(T, B)

Require: T (TVB data structure storing both graph and Δ_m^{t-}), B (Batches of updates)
Ensure: Δ_m^{t+} (Updated MIS after deletion of edges)

for $b_i \in B$ **do**
  for $(u, v) \in b_i$ **parallel do**
    if $T[v].Membership$ **then**
      $N_u \leftarrow$ getNeighbors(T, u)
      $shouldWeAdd \leftarrow$ true
      for $n \in N_u$ **do**
        if $T[n].Membership$ **then**
          $shouldWeAdd \leftarrow$ false
      if $shouldWeAdd$ **then**
        $T[u].Membership \leftarrow$ true
    if $T[u].Membership$ **then**
      $N_v \leftarrow$ getNeighbors(T, v)
      $shouldWeAdd \leftarrow$ true
      for $n \in N_v$ **do**
        if $T[n].Membership$ **then**
          $shouldWeAdd \leftarrow$ false
      if $shouldWeAdd$ **then**
        $T[v].Membership \leftarrow$ true

Algorithm 5's parallel implementation shares similarities with $P\text{-}DMI_{\text{BAT}}$ approach through parallelization across the edges within each batch, as this algorithm also follows batch dynamic setting. The edge deletion cases are checked using a TVB data structure, which allows direct membership verification of neighbours by indexing. This eliminates the need for an iteration through the entire Δ_m^{t+} array to check the membership of every neighbour. Under a parallel implementation, the time complexity of this algorithm can be expressed as $T(m, k) = O\left(k \cdot \frac{m}{p}\right)$.

F. Parallel Dynamic MIS Update for Deletion in the batched setting using $TVBL$ data structure $P\text{-}DMD_{TVBL}$

Algorithm 6 $P\text{-}DMD_{\text{TVBL}}$(T, B)

Require: T (TVBL data structure storing graph, Δ_m^{t-} and levels of the nodes), B (Batches of updates)
Ensure: Δ_m^{t+} (Updated MIS after deletion of edges)

1: **for** $b_i \in B$ **do**
2:   **for** $(u, v) \in b_i$ **parallel do**
3:     **if** $T[v].Membership$ **then**
4:       **if** $T[u].Level = 1$ **then**
5:         $T[v].Membership \leftarrow$ true
6:     **if** $T[u].Membership$ **then**
7:       **if** $T[v].Level = 1$ **then**
8:         $TVBL[v].Membership \leftarrow$ true

By utilizing the $TVBL$ data structure, the usage of the iterations to check the membership of all neighbours of a non-Δ_m^{t+} vertex that needs to be added to the set is eliminated. This can done by checking the level (z coordinate). If the level is 1 (as seen in Lines 4 and 7), we add that vertex to Δ_m^{t+}. This reduces the time complexity of the deletion algorithm to $T(m) = O\left(\frac{m}{p}\right)$.

Appendix C
Probability of Conflicting Updates in a 2-Hop Neighbourhood

Problem Setup and Assumptions

In our parallel MIS update algorithm, a potential race condition can occur if two threads attempt to update vertices that are within a 2-hop distance (i.e., two edges apart) in the same batch of updates. To calculate the probability of this occurrence under the assumption that edges are distributed uniformly at random across the batches, we define the following parameters:

- $|V|$: Total number of vertices in the graph.
- $|E|$: Total number of edges in the graph.
- k: Average degree of the graph, meaning each vertex has approximately d neighbors.
- θ_{U}: Size of a batch of updates, or the number of edges updated in a single parallel operation.
- T: Number of threads.

Step 1: Calculate the Expected Number of 2-Hop Neighborhoods Affected per Thread

Vertices Directly Affected by Each Thread: Each thread in a batch accesses $\frac{\theta_{\mathrm{U}}}{T}$ edges on average. Since each edge connects two vertices, each thread directly interacts with approximately $2 \times \left(\frac{\theta_{\mathrm{U}}}{T}\right)$ vertices.

Total 2-Hop Neighborhoods Covered by Each Thread: For each vertex v directly accessed by a thread, the 2-hop neighbourhood includes all vertices that are within two edges of v. In a graph with average degree k, each vertex's 2-hop neighbourhood contains approximately k^2 number of vertices.

Therefore, each thread covers an approximate total of:

$$N_{\text{2-hop}} = 2 \times \frac{\theta_{\mathrm{U}}}{T} \times k^2$$

vertices in 2-hop neighbourhoods across all vertices it interacts with.

Step 2: Probability of a 2-Hop Conflict between Two Threads

The likelihood of two threads encountering a race condition depends on whether the vertices they attempt to update overlap within these 2-hop neighbourhoods.

Probability of a Single Thread Conflict: For each thread, the probability of selecting a vertex in the graph is $\frac{N_{\text{2-hop}}}{|V|}$.

Probability of Conflict between Two Threads: Since two threads might select overlapping neighbourhoods, we calculate the probability that any two threads' selected 2-hop neighbourhoods intersect. For simplicity, let's approximate this as the probability that two threads independently select overlapping neighbourhoods. For a random, independent selection, this probability can be estimated as:

$$P_{\text{conflict}} = \left(\frac{N_{\text{2-hop}}}{|V|}\right)^2$$

Expected Number of Conflicts Across All Threads: With T threads, the number of unique thread pairs is $\binom{T}{2} = \frac{T(T-1)}{2}$. The expected number of conflicts is then:

$$E_{\text{conflicts}} = \frac{T(T-1)}{2} \times \left(\frac{N_{\text{2-hop}}}{|V|}\right)^2$$

Step 3: Substitute Parameters and Simplify

Plugging in $N_{\text{2-hop}}$:

$$E_{\text{conflicts}} = \frac{T(T-1)}{2} \times \left(2 \times \frac{\theta_{\mathrm{U}}}{T} \times k^2\right)^2$$

Simplifying the expression:

$$E_{\text{conflicts}} = \frac{T(T-1)}{2} \times 4 \times \frac{\theta_{\mathrm{U}}^2}{T^2} \times d^4$$

Further simplifying:

$$E_{\text{conflicts}} = \frac{2(T-1)}{T} \times \frac{\theta_{\mathrm{U}}^2 k^4}{|V|^2}$$

Step 4: Interpretation and Approximation

For large graphs where $|V|$ is significantly larger than both θ_{U} and k,

$E_{\text{conflicts}}$ will be small, effectively minimizing race conditions.

For instance, in a graph with $|V| = 10^6$, $\theta_{\mathrm{U}} = 10^4$, $k = 10$, and $T = 8$:

$$E_{\text{conflicts}} \approx \frac{2 \times 7}{8} \times \frac{10^4}{10^6} \approx 0.0175$$

Thus, with these parameters, the expected number of conflicts is around 1.75%, aligning with empirical results and suggesting that race conditions remain rare and manageable in practical applications.