

# Parallel maximal independent set algorithms for graphs and hypergraphs

Jessica Shi, Yiqiu Wang, Zeyuan Shang  
6.854 Advanced Algorithms, Fall 2018

## 1. Introduction

### 1.1. Background and motivation

Given an undirected graph  $G$ , the *maximal independent set problem* (MIS) asks for an independent subset of vertices  $U \subseteq V(G)$  such that all vertices in  $V(G) \setminus U$  have a neighbor in  $U$ . The *lexicographically first MIS problem* fixes a random ordering of vertices, and returns the maximal independent set given by a standard greedy sequential algorithm, where vertices are considered in order and added to the MIS if they do not violate the independent property of the set. This is a fundamental problem in parallel computing, since we can envision tasks as vertices and an edge as the constraint that the two corresponding tasks cannot run in parallel; in this sense, a MIS is precisely a maximal set of tasks that can be run in parallel.

One current area of research involves parallel MIS algorithms. In particular, parallel lexicographically first MIS algorithms differ from parallel MIS algorithms in that the former can be adapted to return precisely the same MIS as a sequential algorithm (namely, the sequential greedy algorithm). The latter makes no such guarantee. This may be desirable in situations where one is choosing between parallel and sequential algorithms depending on the platform, but wants the same result to be returned regardless of which platform is selected.

One well-known parallel algorithm for MIS is Luby's [8] randomized algorithm, which takes  $O(\log |V(G)|)$  time on  $O(|E|)$  processors.<sup>1</sup> Recently, Blelloch *et al.* [4] achieved a parallel algorithm for lexicographically first MIS, with linear work and polylogarithmic ( $O(\log^2 |V(G)|)$ ) span with high probability (excepting a negligible number of orderings). Fischer and Noever [5] improved the span to be logarithmic ( $O(\log |V(G)|)$ ) with high probability, and showed that this bound is tight.

We can also consider the MIS and lexicographically first MIS problems in hypergraphs. One well-known parallel algorithm for MIS on hypergraphs is Karp *et al.*'s [7] randomized algorithm, which takes  $O(\sqrt{|V(G)|})$  time on a polynomial number of processors. Recently, Bercea *et al.* [3] developed a parallel algorithm for hypergraphs satisfying  $|E(G)| \leq |V(G)|^{\frac{\log \log |V(G)|}{\log \log \log |V(G)|}}$  in  $O(|V(G)|^{\rho(1)})$  time and a polynomial number of processors. Note that Bercea *et al.* achieve this bound by using a previous parallel MIS algorithm, by Beame and Luby [2], as a subroutine.

There has been little work on parallel lexicographically first MIS algorithms for hypergraphs. We hypothesize that parallelizing the sequential greedy algorithm for MIS on hypergraphs will give a work-efficient algorithm for lexicographically first MIS with  $O(|V(G)|^{\rho(1)})$  span with high probability. Note that this is an extension of Blelloch *et al.*'s [4] work, since they applied a similar parallelization for the graphs case. Throughout this paper, we refer to this algorithm either as the

---

<sup>1</sup>See Section 2.3 for a brief introduction on parallel algorithm definitions, if you have no previous background in parallel algorithms.

lexicographically first MIS algorithm for hypergraphs, or as an extension of Blelloch *et al.*'s [4] algorithm to hypergraphs.

## 1.2. Our results

In this paper, we provide an overview of parallel MIS and parallel lexicographically first MIS algorithms for graphs and hypergraphs. While we have been unable to prove bounds for the extension of Blelloch *et al.*'s [4] algorithm to hypergraphs, we implement this algorithm and compare its performance to that of Karp *et al.*'s [7] classic parallel MIS algorithm for hypergraphs. We also compare this to a previous implementation [11] of Beame and Luby's [2] parallel MIS algorithm for hypergraphs.

Note that while we include these implementation details to justify work on parallel lexicographically first algorithms in hypergraphs, this paper is primarily a reading paper.

## 1.3. Outline

In Section 2, we introduce preliminary notations and definitions. In Section 3, we discuss parallel MIS algorithms for graphs, including Luby's [8] algorithm and Blelloch *et al.*'s [4] algorithm. In Section 4, we discuss parallel MIS algorithms for hypergraphs, including Karp *et al.*'s [7] algorithm and an extension of Blelloch *et al.*'s [4] algorithm to hypergraphs. We also implement these algorithms and compare their performances in practice.

# 2. Preliminaries and definitions

## 2.1. Graph preliminaries

Let  $G$  denote the input graph/hypergraph to each of our algorithms, and let  $V(G)$  and  $E(G)$  denote the set of vertices and the set of edges of  $G$  respectively. Note that in a graph, unordered pairs of vertices form edges, while in a hypergraph, any unordered subset of vertices form edges. Thus, hypergraphs consist of vertices  $V(G)$  and edges  $E(G) \subseteq \mathcal{P}(V(G))$ , where  $\mathcal{P}(S)$  denotes the powerset of a set  $S$ .<sup>2</sup>

For graphs, we will take  $n$  to be the number of vertices, and  $m$  to be the number of edges. For hypergraphs, we simply take  $n$  to be the size of the graph, including both edges and vertices. Note that we take  $G$  to be *simple*, that is to say, undirected with no self-loops and no multiple edges.

For a vertex  $v \in V(G)$ , we define the *neighborhood* of  $v$ , denoted by  $N(v)$ , to be the set of vertices in  $V(G) \setminus \{v\}$  adjacent to  $v$ . For a subset of vertices  $V \subseteq V(G)$ , we define the *neighborhood* of  $V$ , denoted by  $N(V)$ , to be the set of vertices in  $V(G) \setminus V$  adjacent to  $V$ .

Given a subset of vertices  $V \subseteq V(G)$ , we let  $G[V]$  denote the *vertex-induced subgraph* of  $G$  on  $V$ . The vertex-induced subgraph is the graph consisting of the set of vertices  $V$  along with any edges within  $V$  that appear in  $G$ . Similarly, given a subset of edges  $E \subseteq E(G)$ , we let  $G[E]$  denote the *edge-induced subgraph* of  $G$  on  $E$ . The edge-induced subgraph is the graph consisting of the set of edges  $E$  along with any vertices within  $E$ .

## 2.2. Maximal independent set

An *independent set* (or *stable set*) is a set of vertices  $V \subseteq V(G)$  such that  $G[V]$  contains no edges. A *maximal independent set* (*MIS*) is an independent set  $V$  such that for all  $v \in V(G) \setminus V$ ,  $V \cup \{v\}$  is not

---

<sup>2</sup>A *k-uniform* hypergraph is a hypergraph where every edge consists of  $k$  vertices. While this is a popular convention, we consider general hypergraphs here, where there is no restriction on the lengths (cardinalities) of the edges.

an independent set. In other words, we cannot add more vertices to  $V$  to form a larger independent set. Note that *maximal* is distinct from *maximum*, and  $V$  is not necessarily the largest independent set of  $G$ .

In a sequential setting, there is a simple greedy algorithm to find the MIS of a graph/hypergraph  $G$ . Namely, we fix some random ordering of our vertices  $V(G)$ , and we construct a MIS by adding each vertex in order to our MIS if it does not violate the independence condition of our MIS. More formally,

---

**Algorithm 1** Greedy sequential

---

```

1: procedure MIS( $G$ )
2:    $M \leftarrow \emptyset$  ▷ MIS
3:   for each  $v \in V(G)$  in a fixed random order do
4:     Add  $v$  to  $M$  if  $M \cup \{v\}$  is an independent set
   return  $M$ 

```

---

Note that the MIS that the greedy sequential algorithm returns is known as the *lexicographically first MIS*. Importantly, this algorithm will be the basis of Blelloch *et al.*'s [4] parallel algorithm for lexicographically first MIS on graphs.

### 2.3. Parallel algorithm preliminaries

A deep understanding of parallel algorithms is not necessary to understand the content of this paper. However, we give here a general overview of parallel algorithms, to hopefully provide enough basic background for the bounds that we discuss.

A parallel algorithm is an algorithm that can be executed over different processors (usually concurrently), and then synchronized and combined to give a final result.

There are many models for analyzing parallel algorithms; in this paper, we use the concurrent-read concurrent-write (CRCW) parallel random access machine (PRAM) model for analyzing parallel algorithms (also known as concurrent random-access machine). Importantly, this means that we have an unbounded number of random access machine (RAM) processors, with an unbounded amount of shared memory. Multiple processors are allowed to read and write to our shared memory at the same time.<sup>3</sup>

The *work* of a parallel algorithm is the amount of time that the algorithm would take to run given only one processor. The *span* of a parallel algorithm is the amount of time that the algorithm would take to run given an infinite number of processors. These two notions encapsulate the runtime of a parallel algorithm. While there are other important preliminaries regarding parallel algorithms, these are sufficient for the purposes of this paper.

### 2.4. Parallel MIS definitions

We now introduce some notation specific to parallel MIS algorithms; note that while some of this notation is not necessarily standard, it will make our analysis in Sections 3 and 4 easier.

At every “step” of our algorithms (where steps will be clearly defined when we discuss specific algorithms), we call a vertex  $v$  *active* if it can be added to our MIS; that is to say, adding  $v$  to our

---

<sup>3</sup>Note that only one processor is allowed to write to a particular memory cell at any given time, and behavior is undefined if one processor attempts to write to a cell and another process attempts to read/write that same cell in parallel.

MIS does not complete an edge. For graphs, this means that no vertex  $u \in N(v)$  is already in our MIS.

Similarly, we call a vertex  $v$  *dead* if it cannot be added to our MIS; that is to say, adding  $v$  to our MIS does complete an edge. For graphs, this means that for some  $u \in N(v)$ ,  $v$  is already in our MIS. Note that in our algorithms, we never have to consider dead vertices; we can simply delete dead vertices at the end of each step, since once a vertex dies, it can never join our MIS.

We let  $AN(v)$  denote the *active neighbors* of  $v$ , that is to say, the subset of active vertices in  $N(v)$ . We let  $ad(v)$  denote the *active degree* of  $v$ , that is to say,  $|AN(v)|$  (the number of neighbors of  $v$  which are active).

Colloquially, we will also say that an active vertex  $v$  *kills*<sup>4</sup> an active vertex  $u \in AN(v)$  when  $v$  is added to our MIS, and adding both  $u$  and  $v$  to our MIS would result in an edge. The idea here is that  $u$  dies as a result of  $v$ . For graphs, this simply occurs when  $v$  is added to our MIS before  $u$  is added to our MIS.

### 3. Parallel MIS in graphs

In this section, we discuss Luby's [8] algorithm for parallel MIS in graphs, and we give the proof that Luby's algorithm has work  $O(m)$  and span  $O(\log n)$  in expectation. Note that Luby's algorithm actually takes work  $O(m)$  and span  $O(\log n)$  with high probability, but we omit this extension for the purposes of this paper.

We also introduce Blelloch *et al.*'s [4] algorithm for parallel lexicographically first MIS in graphs. Blelloch *et al.* showed that their algorithm, which is a simple parallelization of the sequential greedy algorithm, has work  $O(m)$  and span  $O(\log^2 n)$  with high probability. Fischer and Noever [5] showed that this algorithm has work  $O(m)$  and span  $O(\log n)$  with high probability. We omit both of these proofs for the purposes of this paper.

#### 3.1. Intuition

There is one main intuition for how parallel MIS in graphs works; note that here, we will speak quite loosely about what it means for a vertex to be added to our MIS.

Consider some step of our algorithm, and any active vertex  $v$ . Note that we cannot add  $v$  and any active neighbor  $u \in AN(v)$  to our MIS in the same parallel step; clearly, we have a conflict if  $v$  and  $u$  are both in our MIS. As such, we must consider  $v$  and  $u \in AN(v)$  in distinct steps, and the span of our algorithm is necessarily restricted (in some way) by the neighbors  $AN(v)$ .

We see that the active degree of  $v$  is important. Here, we have two cases.

If we choose an active vertex  $v$  with low active degree to be in our MIS, this essentially means that we can add more vertices to our MIS at the same time as we add  $v$ . In other words, there are not many  $u \in AN(v)$  that we cannot add to our MIS at the same time as  $v$ . The general idea here is that by adding  $v$ , we can process many vertices in the same step as  $v$ .

If we choose an active vertex  $v$  with high active degree to be in our MIS, this essentially means that  $v$  kills many vertices in that step, since all active neighbors of  $v$  become dead. The general idea here is that by adding  $v$ , the number of vertices that we have left to consider in future steps decreases significantly.

In both cases, we see that we progress meaningfully in our algorithm, and this will be the main intuition for bounding the spans of our parallel algorithms.

---

<sup>4</sup>No vertices were harmed in the making of this paper.

### 3.2. Luby's algorithm (MIS)

Luby's [8] algorithm for parallel MIS in graphs involves assigning random priorities to vertices and adding all vertices with no earlier priority neighbors to our MIS. We then delete all neighbors of the vertices in our MIS, and repeat until all vertices have been considered (either deleted or added to our MIS).

Note that throughout this section, we make liberal use of various well-written notes regarding Luby's algorithm. These include notes from Shun [12], Lynch [9], Haeupler [6], and Miller [10].

---

#### Algorithm 2 Luby [8]

---

```

1: procedure MIS( $G$ )
2:    $A \leftarrow V(G)$  ▷ Active vertices
3:    $M \leftarrow \emptyset$  ▷ MIS
4:   while  $A \neq \emptyset$  do
5:     Each vertex  $v \in A$  chooses a value  $r_v$  uniformly at random from  $\{1, 2, \dots, n^c\}$ 
6:     ▷ where  $c$  is a constant defined in Lemma 3.2
7:      $M' \leftarrow \emptyset$ 
8:     for (parallel) each  $v \in A$  do
9:       if  $r_v > r_u$  for all  $u \in N(v) \cap A$  then
10:        Add  $v$  to  $M'$ 
11:      $M \leftarrow M \cup M'$ 
12:      $A \leftarrow A \setminus (M' \cup N(M'))$ 
return  $M$ 

```

---

Note that lines 5 and 9 can be executed in parallel as well. As such, the span of this algorithm is given simply by the number of iterations in our while loop (lines 4 to 12); we refer to each iteration of our while loop as a step.

We now show that 1) if Luby's algorithm terminates, then it gives a MIS of  $G$  and 2) Luby's algorithm has work  $O(m)$  and span  $O(\log n)$  in expectation.

**Theorem 3.1.** *If Luby's algorithm terminates, then it returns a MIS of  $G$ .*

*Proof.* First, we note that an invariant of the algorithm is that  $M$  is independent. This is because a vertex  $v$  is added to  $M$  only if  $v$  is active and none of its active neighbors is added to  $M$  in that step (by line 9). Moreover, as soon as  $v$  is added to  $M$ , all of its neighbors die (and are never considered to be added to  $M$  in the future).

Now,  $M$  is maximal because the only way a vertex  $v$  is removed from our active set  $A$  is if either  $v$  is already added to  $M$ , or a neighbor of  $v$  is added to  $M$  ( $v$  is killed). Thus, if our algorithm terminates, then the only vertices not in  $M$  are vertices that have a neighbor in  $M$ . This is precisely the definition of maximality.

Thus, if Luby's algorithm terminates, then it returns a MIS of  $G$ . □

In order to show our second point, namely the work and span of Luby's algorithm, we first introduce several lemmas. The version of the proof that we use here was originally given by Yves *et al.* [13].

**Remark 1.** The main idea here is to show that each step of our algorithm reduces the number of edges in  $G[A]$  by half in expectation. The intuition for this is similar to what we discussed in Section 3.1; importantly, we have a balance between the active degree of any active vertex  $v$  and the probability that  $v$  is added to our MIS at the given step.

If  $v$  has low active degree, then it has a higher probability of being added to our MIS (since there are few values  $r_u$  for  $u \in AN(v)$  that  $r_v$  has to be greater than). This means that while adding  $v$  to our MIS does not remove many edges from  $G[A]$ , these edges have a higher probability of being removed. Thus, the expected number of edges removed from  $G[A]$  remains significant (in a loose sense).

If  $v$  has high active degree, then it has a lower probability of being added to our MIS. However, adding  $v$  to our MIS removes many edges from  $G[A]$ , so again, the expected number of edges removed from  $G[A]$  remains significant.

**Lemma 3.2.** *With high probability, at each step of Luby's algorithm, every (active) vertex chooses a different random value from the random values that its (active) neighbors choose.*

*Proof.* First, note that for every  $(u, v) \in E(G[A])$ , the probability that  $u$  and  $v$  choose the same random value is given by

$$Pr[r_u = r_v] = \frac{1}{n^c}.$$

Thus, by the union bound, the probability that any two adjacent vertices in  $G[A]$  choose the same random value is given by

$$Pr[\exists (u, v) \in E(G[A]) \text{ s.t. } r_u = r_v] \leq \sum_{(u, v) \in E(G[A])} Pr[r_u = r_v] \leq \frac{m}{n^c}.$$

Thus, we see that for sufficiently large constant  $c$ , the probability that any two adjacent vertices choose the same random value is negligible.  $\square$

From this, in the rest of our analysis, it is sufficient to assume that all adjacent active vertices have chosen distinct random values.

**Lemma 3.3.** *For each step of Luby's algorithm, the expected number of edges in  $G[A]$  at the end of that step is at most half the number of edges in  $G[A]$  at the start of that step.*

*Proof.* Consider a step of Luby's algorithm. We first introduce the idea of *preemptively killing* a vertex.<sup>5</sup> A vertex  $v$  *preemptively kills* a vertex  $u$  if and only if

1.  $(u, v)$  is an active edge in  $G[A]$
2. for all  $w \in AN(v)$ ,  $r_v > r_w$
3. for all  $w \in AN(u) \setminus \{v\}$ ,  $r_v > r_w$ .

Note that the first two conditions say that  $v$  kills  $u$ , since  $v$  is added to our MIS at this step (before  $u$  is added to the MIS). However, the idea of killing a vertex is not strict enough for the purposes of counting the number of edges that die at this step, because there are multiple vertices that could

---

<sup>5</sup>Shun [12] calls this “preemptively removing.”



have killed  $u$  (namely, any vertex in  $AN(u)$ ); thus, we say that a vertex  $v$  preemptively kills  $u$  if it kills  $u$  and, out of all vertices that could kill  $u$ , has chosen the largest random value.<sup>6</sup>

Now, we see that any active vertex  $u$  can be preemptively killed at most once. Also, if  $v$  preemptively kills  $u$ , then we say that  $v$  has preemptively killed all active edges adjacent to  $u$  (since those edges necessarily die once  $u$  dies). By definition, any active edge  $(u, v)$  can be preemptively killed at most twice (once if  $u$  is preemptively killed, and once if  $v$  is preemptively killed).

We can now lower bound the number of active edges that die at this step as follows. If we define an indicator variable  $X_{(u,v)}$  to be 1 if  $v$  preemptively kills  $u$  and 0 otherwise, then

$$2 \cdot E[\# \text{ edges killed}] \geq \sum_{(u,v) \in E(G[A])} Pr[X_{(u,v)} = 1] \cdot ad(u) + Pr[X_{(v,u)} = 1] \cdot ad(v).$$

Now, in order for  $v$  to preemptively kill  $u$ ,  $v$  must have chosen a random value greater than all of its active neighbors and greater than all of  $u$ 's active neighbors (excepting  $v$ ). Thus,  $Pr[X_{u,v} = 1] \geq 1/(ad(u) + ad(v))$ ; note that this is an inequality, because  $u$  and  $v$  may share neighbors.<sup>7</sup> This gives us

$$2 \cdot E[\# \text{ edges killed}] \geq \sum_{(u,v) \in E(G[A])} \frac{ad(u)}{ad(u) + ad(v)} + \frac{ad(v)}{ad(u) + ad(v)} = |E(G[A])|.$$

We have that the expected number of edges killed in this step is at least  $|E(G[A])|/2$ . Thus, the expected number of edges in  $G[A]$  at the end of this step is at most half the number of edges in  $G[A]$  at the start of this step, as desired.  $\square$

**Theorem 3.4.** *Luby's algorithm has work  $O(m)$  and span  $O(\log n)$  in expectation.*

*Proof.* Each step of Luby's algorithm does  $|E(G[A])|$  work, and  $E(G[A])$  reduces by half in expectation during each step. Thus, in total, Luby's algorithm does  $O(m)$  work in expectation.

Since  $E(G[A])$  reduces by half in expectation during each step, and we terminate at most one step after no edges remain (when there are no edges remaining, the next step adds all active vertices to our MIS, and we are done), we see that the number of steps is given by  $O(\log m) = O(\log n)$ . Moreover, as mentioned previously, each step has span 1. Thus, in expectation, Luby's algorithm has span  $O(\log n)$ .  $\square$

Note that we can actually show that Luby's algorithm has span  $O(\log n)$  with high probability, by applying a Chernoff bound [9, 6, 10]. We omit this proof from this paper, since it is somewhat out of scope from an algorithms perspective.

### 3.3. Blleloch *et al.*'s algorithm (Lexicographically first MIS)

Blleloch *et al.* [4] showed that a simple parallelization of the sequential greedy algorithm for MIS can be executed in  $O(m)$  total work and  $O(\log^2 n)$  span with high probability. Fischer and Noever [5] showed that this parallelization actually takes  $O(\log n)$  span with high probability; as such, the theoretical bounds on Blleloch *et al.*'s lexicographically first MIS algorithm are precisely the same as the bounds on Luby's algorithm.

<sup>6</sup>Importantly, this definition undercounts the number of ways in which  $u$  could die at this step.

<sup>7</sup>This is also where we use Lemma 3.2, since we are assuming that every active vertex chooses a different random value than its active neighbors.

We give Blelloch *et al.*'s algorithm here, but we will not formally prove the work and span bounds. The general intuition for Blelloch *et al.*'s initial bounds is precisely that in Section 3.1; one of the main ideas is that after processing a certain number of vertices, the remaining vertices will have low degree with high probability (if they had high degree, then they have a high probability of being killed while processing previous vertices), and as such we can process many of these vertices in the same step.

Note that Fischer and Noever's bound is much more involved, and we omit a discussion of their work from this paper.

---

**Algorithm 3** Blelloch *et al.* [4]

---

```

1: procedure MIS( $G$ )
2:    $A \leftarrow V(G)$ , where  $A$  is in a fixed random order           ▷ Active vertices
3:    $M \leftarrow \emptyset$                                            ▷ MIS
4:   while  $A \neq \emptyset$  do
5:     Let  $A'$  be the set of vertices in  $A$  with no earlier neighbors in  $A$ , based on the ordering
6:     Add  $A'$  to  $M$ 
7:     Remove  $A'$  and  $N(A')$  from  $A$ 
   return  $M$ 

```

---

Note that the parallelization comes primarily from lines 5 and 7, which can each be executed in parallel with constant span.<sup>8</sup>

## 4. Parallel MIS in hypergraphs

In this section, we discuss Karp *et al.*'s [7] algorithm for parallel MIS in hypergraphs, and we give the proof that Karp's algorithm has work  $O(n^2)$  and span  $O(\sqrt{n})$  in expectation.<sup>9</sup>

We also introduce an extension to Blelloch *et al.*'s [4] algorithm, namely a simple parallelization of the greedy sequential algorithm for MIS in hypergraphs. We discuss some intuition for why bounds for this algorithm could be obtainable, although we have been unable to prove any non-trivial bounds.

### 4.1. Karp *et al.*'s algorithm (MIS)

Karp *et al.*'s [7] algorithm for parallel MIS in hypergraphs involves choosing a random ordering of vertices and finding the last vertex such that adding said vertex and all prior vertices to the MIS forms an independent set. We add precisely these vertices, delete all dead vertices, and repeat until all vertices have been considered (either deleted or added to our MIS).

Note that this is quite similar to a greedy sequential algorithm, but the randomization at the start of every step allows us to obtain the requisite bounds for the span of this algorithm.

---

<sup>8</sup>To be more explicit, for line 5, we can assign each vertex in  $A$  to a different processor. On each processor corresponding to vertex  $v$ , we can then check on  $|N(v)|$  more processors whether each neighbor of  $v$  has an earlier priority than  $v$  considering the ordering in  $A$ . This takes constant time per processor if we store  $A$  in a proper way, such as in a hash table with vertices as keys and index order as values. As such, line 5 has constant span. A similar argument can be applied to line 7. In total, the span of this algorithm is simply the number of iterations of the while loop (lines 4 to 7), which Blelloch *et al.* [4] and Fischer and Noever [5] bound.

<sup>9</sup>Note that as mentioned in Section 2,  $n$  in the context of hypergraphs refers to the size of the hypergraph, including vertices and edges.



---

**Algorithm 4** Karp *et al.* [7]

---

```
1: procedure MIS( $G$ )
2:    $A \leftarrow V(G)$  ▷ Active vertices
3:    $M \leftarrow \emptyset$  ▷ MIS
4:   while  $A \neq \emptyset$  do
5:     Choose a random ordering of vertices in  $A$ , say  $a_1, \dots, a_{|A|}$ 
6:      $T \leftarrow \emptyset$ 
7:     for (parallel)  $i = 1, \dots, \min(\lfloor \sqrt{n} \rfloor, |A|)$  do
8:       if  $M \cup \{a_1, \dots, a_i\}$  is an independent set in  $G$  then
9:         Add  $i$  to  $T$ 
10:     $t \leftarrow \max T$ 
11:    Add  $a_1, \dots, a_t$  to  $M$ 
12:    Remove  $a_1, \dots, a_t$  from  $A$ 
13:    for (parallel) each  $v \in A$  do
14:      if  $M \cup \{v\}$  is not an independent set in  $G$  then
15:        Remove  $v$  from  $A$ 
return  $M$ 
```

---

Note that line 5 can be executed in parallel with  $O(n^{3/2})$  expected work and  $O(1)$  span with high probability. [1]<sup>10</sup> Moreover, checking if a set of vertices forms an independent set takes at most  $O(n)$  work and  $O(1)$  span.<sup>11</sup> Thus, the span of this algorithm is given by the number of iterations of our while loop (lines 4 to 15); again, we refer to each iteration of our while loop as a step.

We now show that 1) Karp *et al.*'s algorithm gives a MIS of  $G$  and 2) Karp *et al.*'s algorithm has work  $O(n^2)$  and span  $O(\sqrt{n})$  in expectation.

**Theorem 4.1.** *Karp et al.'s algorithm returns a MIS of  $G$ .*

*Proof.* As in the proof of Theorem 3.1, note that an invariant of this algorithm is that  $M$  is independent. This is because a sequence of vertices are added to  $M$  only if they form an independent set with  $M$  (by line 8).

Now,  $M$  is maximal because the only way a vertex  $v$  is removed from our active set  $A$  is if either  $v$  is already added to  $M$ , or adding  $v$  to  $M$  would cause an edge to appear in  $M$  ( $v$  is killed). Thus, the only vertices not in  $M$  are vertices that, when added to  $M$ , would cause an edge to appear in  $M$ . This is precisely the definition of maximality.

Thus, Karp *et al.*'s algorithm returns a MIS of  $G$ . □

We now show the work and span of Karp *et al.*'s algorithm.

---

<sup>10</sup>This is somewhat involved, but essentially, we can execute the Fisher-Yates shuffle (Knuth shuffle) in parallel. First, each element of  $A$  chooses a random number in  $\{1, \dots, c|A|^2 \log |A|\}$ , where  $c$  is a constant. By the same argument as in Lemma 3.2, with high probability all elements in  $A$  will choose a distinct number. Then, Alon, Azar, and Vishkin [1] showed that we can sort our randomly chosen numbers in parallel with  $O(n^{3/2})$  work and  $O(1)$  span with a randomized algorithm. We omit the details of this here, since it is out of scope of this paper and since the analysis is fairly involved. Overall, though, this means that we can find a random permutation in  $O(n^{3/2})$  work and  $O(1)$  span with high probability.

<sup>11</sup>This is somewhat tricky, but there are certain ways of storing our graph that will give us these theoretical bounds. For example, we could have objects corresponding to vertices, and each edge could consist of pointers to the relevant vertices. Then, we can mark all vertices that appear in our set of vertices (in parallel), and iterate through all edges to see if an edge consists of solely marked vertices (in parallel). This setup gives us the desired work and span bounds.

**Remark 2.** The main idea here is to view the work done in each step sequentially. Fix some step of our algorithm. In this step, we essentially randomly choose vertices out of our active vertices  $A$  to add to our MIS, until we reach one that violates the independence condition of our MIS. Then, we delete from  $A$  all vertices that, if added to our MIS, would violate the independence condition (in other words, we delete all dead vertices).

As we add vertices, there is a probability that the next vertex we consider can actually be added to our MIS. If this probability remains high for a sufficient number of additions, we will (with high probability) add a significant (in a loose sense) number of vertices to our MIS. If this probability becomes low at some point, we see that the expected number of dead vertices remaining in  $A$  becomes high (this is clear because the probability that the next randomly chosen vertex violates the independence condition is high).

Thus, in both cases, at each step a significant number of vertices are removed from  $A$ , either because they are added to our MIS or because they die. This will give our desired bound on the expected number of steps of Karp *et al.*'s algorithm.

**Lemma 4.2.** *For each step of Karp *et al.*'s algorithm, if we let  $n_A$  denote the number of vertices in  $A$  at the start of the step, then the expected number of vertices that we delete from  $A$  by the end of the step is at least  $e^{-1}\sqrt{n_A} + O(1)$ .*

*Proof.* Consider a step of Karp *et al.*'s algorithm. First, note that  $a_1, \dots, a_{n_A}$  is our random ordering of vertices in  $A$ . As mentioned in Remark 2, we define  $p_j$  to be the conditional probability that we can add  $a_j$  to our MIS, given that  $a_1, \dots, a_{j-1}$  are already added to our MIS.

Let  $M_s$  denote the event that we can add  $a_1, \dots, a_s$  to our MIS. Then,

$$\Pr[M_s] = \prod_{j=1}^s p_j.$$

Moreover, the expected number of dead vertices remaining in  $A$  after we have added  $a_1, \dots, a_s$  to our MIS is given by<sup>12</sup>

$$E[\# \text{ dead vertices in } A \text{ after } M_s] = (1 - p_{s+1}) \cdot (n_A - s).$$

Now, let us fix  $s$  (we will set a specific value for  $s$  later in this proof). A lower bound on the expected number of elements that we delete from  $A$  in this step is given by the probability that we add the first  $s$  elements of  $A$  to our MIS, and then delete all killed vertices from this addition. In other words,

$$\begin{aligned} E[\# \text{ deleted vertices from } A] &\geq \Pr[M_s] \cdot (s + E[\# \text{ dead vertices in } A \text{ after } M_s]) \\ &= \prod_{j=1}^s p_j \cdot (s + (1 - p_{s+1}) \cdot (n_A - s)). \end{aligned}$$

We will now show how we can fix  $s$  such that this expected value is  $\geq e^{-1}\sqrt{n_A} + O(1)$ . As mentioned in Remark 2, we do this using casework on the probability that we can successively add new vertices to our MIS.

---

<sup>12</sup>We can think of this as the probability that the randomly chosen next vertex cannot be added to our MIS (see Remark 2), times the number of vertices remaining in  $A$  ( $n_A - s$ ).

**Case 1** For all  $j \in \{1, \dots, \lfloor \sqrt{n_A} \rfloor + 1\}$ ,  $p_j > e^{-1/\lfloor \sqrt{n_A} \rfloor}$  (the consecutive probabilities that we can continue to add vertices to our MIS are all high). Then, set  $s = \lfloor \sqrt{n_A} \rfloor$ .

As such,

$$\prod_{j=1}^s p_j \geq \prod_{j=1}^s e^{-1/\lfloor \sqrt{n_A} \rfloor} \geq e^{-1},$$

so

$$\begin{aligned} E[\# \text{ deleted vertices from } A] &\geq e^{-1} \cdot (s + (1 - p_{s+1}) \cdot (n_A - s)) \\ &\geq e^{-1} \cdot (\lfloor \sqrt{n_A} \rfloor + (1 - p_{s+1}) \cdot (n_A - s)) \\ &\geq e^{-1} \cdot \lfloor \sqrt{n_A} \rfloor, \text{ since } (1 - p_{s+1}) \cdot (n_A - s) \geq 0 \\ &= e^{-1} \sqrt{n_A} + O(1). \end{aligned}$$

**Case 2** For some  $j \in \{1, \dots, \lfloor \sqrt{n_A} \rfloor + 1\}$ ,  $p_j \leq e^{-1/\lfloor \sqrt{n_A} \rfloor}$  (at some point there is a low probability that we can continue to add vertices to our MIS). Then, let  $s$  be the minimum  $j$  such that  $p_{j+1} \leq e^{-1/\lfloor \sqrt{n_A} \rfloor}$  (such that  $p_{j+1}$  is low).

As in Case 1, we have

$$\prod_{j=1}^s p_j \geq \prod_{j=1}^s e^{-1/\lfloor \sqrt{n_A} \rfloor} \geq e^{-1}.$$

Moreover,

$$\begin{aligned} E[\# \text{ deleted vertices from } A] &\geq e^{-1} \cdot (s + (1 - p_{s+1}) \cdot (n_A - s)) \\ &\geq e^{-1} \cdot (s + (1 - e^{-1/\lfloor \sqrt{n_A} \rfloor}) \cdot (n_A - s)) \\ &\geq e^{-1} \cdot n_A (1 - e^{-1/\lfloor \sqrt{n_A} \rfloor}) \\ &= e^{-1} \sqrt{n_A} + O(1), \end{aligned}$$

where the last equality follows from a Taylor series expansion.<sup>13</sup>

We now see that in all cases,

$$E[\# \text{ deleted vertices from } A] \geq e^{-1} \sqrt{n_A} + O(1),$$

as desired. □

---

<sup>13</sup>The details are somewhat unimportant, but we'll leave them here for completeness. Note that it suffices to show  $n_A - n_A \cdot e^{-1/\sqrt{n_A}} = \sqrt{n_A} + O(1)$ .

By our Taylor series expansion, we have

$$e^{-1/\sqrt{n_A}} = 1 - \frac{1}{\sqrt{n_A}} + \frac{1}{2n_A} + \sum_{i=3}^{\infty} \frac{(-1)^i}{i! \cdot n_A^{i/2}}.$$

Thus,

$$\begin{aligned} n_A - n_A \cdot e^{-1/\sqrt{n_A}} &= n_A - n_A + \sqrt{n_A} - \frac{1}{2} + \sum_{i=3}^{\infty} \frac{(-1)^{i+1}}{i! \cdot n_A^{(i-2)/2}} \\ &= \sqrt{n_A} + O(1), \end{aligned}$$

since the infinite sum converges by the ratio test.

**Theorem 4.3.** *Karp et al.’s algorithm has work  $O(n^2)$  and span  $O(\sqrt{n})$  in expectation.*

*Proof.* By Lemma 4.2, since the number of vertices in  $A$  decreases by at least  $e^{-1}\sqrt{n_A} + O(1)$  in each step, we see that we need at most  $O(\sqrt{n})$  steps.

An easy (and fun) way to see this is that if we let  $f(x) = x - \sqrt{x}$ , the number of steps  $s$  that we need is given by the number of repeated applications of  $f$  on  $n$  that will bring us to 1, that is to say  $f^s(n) = 1$ . Note that the inverse function is  $g(x) = \frac{1}{2} \cdot (x + \sqrt{4x + 1} + 1)$ . Thus, an equivalent question is to find  $s$  such that  $h^s(1) = n$  where  $h(x) = x + \sqrt{x}$ .<sup>14</sup>

Let us define  $s_i$  to be  $h^{s_i}(1) = i^2$ . We can bound the difference  $s_{i+1} - s_i$ . Importantly, since  $(i+1)^2 - i^2 = 2i + 1$ , we see that  $s_{i+1} - s_i \leq 3$  (more directly, this is because  $h^3(i^2) \geq i^2 + 3i \geq (i+1)^2$ ). Thus, since it takes us a constant number of applications of  $h$  to reach consecutive squares, we see that  $s_i = O(i)$ . Thus,  $s = O(\sqrt{n})$ , as desired.

Now, as mentioned previously, each step of Karp et al.’s algorithm does  $O(n^{3/2})$  work in expectation. Since we have  $O(\sqrt{n})$  steps in expectation, in total we have  $O(n^2)$  work in expectation, as desired.

Moreover, each step of Karp et al.’s algorithm has  $O(1)$  span in expectation. Thus, in expectation, Karp et al.’s algorithm has span  $O(\sqrt{n})$ .  $\square$

## 4.2. Lexicographically first MIS

One of the initial motivations for this paper was to discuss a simple parallelization of the sequential greedy algorithm for MIS for hypergraphs, as an extension of Blelloch et al.’s [4] work. While we were unable to produce similar bounds, we include this algorithm here<sup>15</sup> and we discuss how it performs in practice compared to Karp’s algorithm in Section 4.3.

Intuitively, this algorithm should parallelize well, in the same sense that Blelloch et al.’s algorithm parallelizes. Let us say that we have processed some vertices, and the next vertex that we are adding to our MIS, in order, is  $v$ . There are several cases.

If adding  $v$  kills many other vertices, then we will have deleted a significant number of vertices from our active vertices  $A$ . Consider the case where adding  $v$  does not kill many other vertices.

If  $v$  has very few distinct active vertices distributed among edges of small length, then this means that with high probability, we can process many vertices at the same time as we process  $v$  (since with high probability, the vertices that come next will not be killed by adding  $v$ ).

If  $v$  has many distinct active vertices distributed among edges of small length, then this means that none (we use the term “none” loosely) of those vertices could have been picked in our earlier processing, or else  $v$  would have died. Thus, this scenario only occurs with low probability.

In all, we see that roughly, there is some intuition for why we should be able to parallelize this algorithm in a significant sense. However, since this intuition is quite loose, it is possible that there exists a counter-example, in the form of a specific hypergraph that would perform poorly under the greedy algorithm even with randomization. We have been unable to make progress on this problem.

<sup>14</sup>We can be rather loose regarding constants since we only wish to find a big-O bound for  $s$ .

<sup>15</sup>Note that there are multiple ways to parallelize the sequential greedy algorithm. Importantly, one way that most directly extends Blelloch et al.’s work is to add all vertices in  $A$  that do not complete an edge in our MIS with some vertices that appear before said vertex. However, since this extension is somewhat more complicated to formulate and for the purposes of thinking of theoretical bounds, we introduce a somewhat simpler extension where we only add all vertices up until the first vertex that completes an edge in our MIS. One intuition for doing this is that it also closely resembles Karp’s algorithm, without the randomization at the start of each step. We will see in Section 4.3 that this version does give us decent runtimes and outperforms Karp’s algorithm.

---

**Algorithm 5** Lexicographically first MIS in hypergraphs

---

```
1: procedure MIS(G)
2:    $A \leftarrow V(G)$ , where  $A$  is in a fixed random order ▷ Active vertices
3:    $M \leftarrow \emptyset$  ▷ MIS
4:   while  $A \neq \emptyset$  do
5:     Find the first  $v \in A$  s.t.  $M \cup \{u \in A \mid u \text{ is before } v\} \cup \{v\}$  is not independent
6:      $A' \leftarrow \{u \in A \mid u \text{ is before } v\}$ 
7:     Add  $A'$  to  $M$ 
8:     Remove  $A'$  from  $A$ 
9:     Remove all  $w \in A$  such that  $w \cup M$  is not independent
   return  $M$ 
```

---

Note that again, the parallelization comes from lines 5 and 9, each of which can be executed in parallel to some degree. Since we have not computed the span of this algorithm, we omit more details of this parallelization here.

### 4.3. Implementation

We generated random hypergraphs<sup>16</sup> to test three parallel MIS algorithms for hypergraphs, namely Beame and Luby’s [2] algorithm<sup>17</sup> (which was implemented by Shun [11] in an unpublished framework called Hygra), Karp *et al.*’s [7] algorithm, and our lexicographically first MIS algorithm based on Blelloch *et al.*’s [3] work.

Note that we do not test these algorithms on real world examples. This is because this paper is primarily a reading paper, and we were interested in general timing results for the purposes of seeing if the idea of a parallel lexicographically first MIS algorithm for hypergraphs is feasible. The random hypergraphs do cover a wide range of vertex and edge configurations, and while exact timing varies since our algorithms are randomized,<sup>18</sup> they are generally reflective of certain trends.

Information regarding the generated hypergraphs on which we test these algorithms are in Table 1. Note that  $\Delta$  refers to the maximum cardinality of each edge.

**Table 1.** Generated hypergraphs.

Hypergraph	$ V $	$ E $	$\Delta$
rhg-1	10,000	5,000	1,000
rhg-2	5,000	10,000	1,000
rhg-3	10,000	10,000	5,000
rhg-4	5,000	5,000	4,000
rhg-5	10,000	10,000	100
rhg-6	50,000	50,000	1,000
rhg-7	100,000	50,000	1,000

---

<sup>16</sup>These hypergraphs were generated using Shun’s [11] unpublished hypergraph framework, Hygra.

<sup>17</sup>We do not discuss this algorithm in this paper, but it is the basis of Bercea *et al.*’s [3] algorithm, which runs with  $O(n^{\rho(1)})$  span under certain conditions on the number of edges in the hypergraph. This algorithm is mentioned in Section 1.1.

<sup>18</sup>In particular, there are outliers where certain algorithms, such as Karp *et al.*’s algorithm on rhg-3 in parallel, are noticeably slow. This is most likely due to randomness.

We refer to Shun’s [11] implementation of Beame and Luby’s algorithm as “Hygra”. We implemented Karp *et al.*’s algorithm and our lexicographically first MIS algorithm over the Hygra framework, and we refer to these algorithms as “Karp” and “LF” respectively. Since Hygra is unpublished, we have not published our code externally. However, we have attached our code for the purposes of this assignment as supplementary material.<sup>19</sup> Note that all code followed directly from the pseudocode given in the previous sections; the framework for storing hypergraphs is precisely the framework from Hygra, so as such, we omit these details from this paper.

We performed these tests on an unloaded Thinkpad-T460 laptop with a i5-6300U CPU @ 2.40GHz with 4 threads and 8GB RAM. The timings that we receive are listed in Table 2; we first performed single-threaded tests, which appear on the left side of the table, and then we performed 4-threaded tests, which appear on the right side of the table.

**Table 2.** Timing information, in seconds.

Hypergraph	Hygra Seq	LF Seq	Karp Seq	Hygra Par	LF Par	Karp Par
rhg-1	0.123	0.0129	0.0104	0.0518	0.0172	0.126
rhg-2	0.224	0.0255	0.0167	0.0913	0.0188	0.0898
rhg-3	0.881	0.0947	0.0587	0.373	0.0555	0.209
rhg-4	0.303	0.0369	0.0218	0.145	0.0196	0.0916
rhg-5	0.0264	0.0042	0.00922	0.0159	0.00599	0.137
rhg-6	1.29	0.247	0.271	0.0601	0.186	0.86
rhg-7	1.41	0.17	0.297	0.678	0.204	1.57

We observe that sequentially, Hygra is significantly slower than LF and Karp. LF and Karp perform similarly on average, but for large hypergraphs, LF is noticeably faster.

In parallel, LF has a visible advantage over both Hygra and Karp, in particular by a maximum of 22x over Karp on rhg-5, and 7x over Hygra on rhg-4. Therefore, although proving the theoretical efficiency of our lexicographically first MIS algorithm is still an open question, it actually performs well in practice with good parallelism.

## 5. Conclusion

In this paper, we have provided an overview of various parallel MIS algorithms for graphs and hypergraphs. We have also shown in Section 4.3 that a simple parallelization of the sequential greedy algorithm for MIS in hypergraphs outperforms classical parallel algorithms for MIS in hypergraphs. While we have been unable to prove theoretical bounds, we hypothesize that the span of this algorithm is  $O(|V(G)|^{o(1)})$  with high probability. In the future, we hope to make progress on deriving a bound for this algorithm.

## 6. Acknowledgements

We would like to thank Julian Shun, for providing the idea for this project and for his invaluable guidance. We would also like to thank David Karger and Aleksander Mądry, for their support throughout 6.854.

<sup>19</sup>Note that we have excluded the Hygra framework, so the code as-is will not run.



## References

- [1] N. Alon, Y. Azar, and U. Vishkin. Tight complexity bounds for parallel comparison sorting. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science, SFCS '86*, pages 502–510, Washington, DC, USA, 1986. IEEE Computer Society.
- [2] P. Beame and M. Luby. Parallel search for maximal independence given minimal dependence. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '90*, pages 212–218, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.
- [3] I. O. Bercea, N. Goyal, D. G. Harris, and A. Srinivasan. On computing maximal independent sets of hypergraphs in parallel. *ACM Trans. Parallel Comput.*, 3(1):5:1–5:13, Jan. 2017.
- [4] G. E. Blelloch, J. T. Fineman, and J. Shun. Greedy sequential maximal independent set and matching are parallel on average. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12*, pages 308–317, New York, NY, USA, 2012. ACM.
- [5] M. Fischer and A. Noever. Tight analysis of parallel randomized greedy MIS. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '18*, pages 2152–2160, Philadelphia, PA, USA, 2018. Society for Industrial and Applied Mathematics.
- [6] B. Haeupler. Algorithmic superpower randomization: Lecture 20. <http://www.cs.cmu.edu/~haeupler/15859F15/docs/lecture20.pdf>, 2015.
- [7] R. M. Karp, E. Upfal, and A. Wigderson. The complexity of parallel search. *J. Comput. Syst. Sci.*, 36(2):225–253, Apr. 1988.
- [8] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC '85*, pages 1–10, New York, NY, USA, 1985. ACM.
- [9] N. Lynch. Class on design and analysis of algorithms, lecture 19 notes. [https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-notes/MIT6\\_046JS15\\_lec19.pdf](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/lecture-notes/MIT6_046JS15_lec19.pdf), 2015.
- [10] G. Miller. Lecture 32: Luby’s algorithm for maximal independent set. <http://www.cs.cmu.edu/afs/cs/academic/class/15750-s17/ScribeNotes/lecture32.pdf>, 2017.
- [11] J. Shun. Hygra.
- [12] J. Shun. Notes on simple analysis for parallel maximal independent set and maximal matching algorithms. <https://people.csail.mit.edu/jshun/simple-analysis.pdf>, n.d.
- [13] M. Yves, J. M. Robson, S.-D. Nasser, and A. Zemmari. An optimal bit complexity randomized distributed mis algorithm (extended abstract). In *Proceedings of the 16th International Conference on Structural Information and Communication Complexity, SIROCCO'09*, pages 323–337, Berlin, Heidelberg, 2010. Springer-Verlag.