Parallel Search for Maximal Independence Given Minimal Dependence

Paul Beame[1] and Michael Luby[2]

TR-89-003

February, 1989

We consider the problem of finding a maximal independent set fast in parallel when the independence system is presented as an explicit list of minimal dependent sets. Karp and Wigderson [KW] were the first the find an *NC* algorithm for the special case when the size of each minimal dependent set is at most two, and subsequent work by Luby [Lu1], by Alon, Babai and Itai [ABI] and Goldberg and Spencer [GS] have introduced substantially better algorithms for this case. On the other hand, no previous work on this problem extends even to the case when the size of each minimal dependent set is at most three. We provide a randomized *NC* algorithm for the case when the size of each minimal dependent set is at most a constant, and we conjecture that this algorithm is a randomized *NC* algorithm for the general case.

[1]Computer Science Department, FR-35, University of Washington, Seattle, Washington 98195, research supported by NSF grant CCR-8858799. - beame@cs.washington.edu

[2]International Computer Science Institute, Berkeley, California on leave of absence from the Univeristy of Toronto, research partially supported by N.S.E.R.C. of Canada operating grant A8092. - luby@icsi.berkeley.edu

Parallel Search for Maximal Independence given Minimal Dependence

Paul Beame *
Computer Science Department, FR-35
University of Washington
Seattle, Washington 98195
beame@cs.washington.edu

Michael Luby†
International Computer Science Institute, Suite 600
1947 Center Street
Berkeley, California 94704
luby@icsi.berkeley.edu

January 27, 1989

Abstract

We consider the problem of finding a maximal independent set fast in parallel when the independence system is presented as an explicit list of minimal dependent sets. Karp and Wigderson [KW] were the first to find an NC algorithm for the special case when the size of each minimal dependent set is at most two, and subsequent work by Luby [Lu1], by Alon, Babai and Itai [ABI] and by Goldberg and Spencer [GS] have introduced substantially better algorithms for this case. On the other hand, no previous work on this problem extends even to the case when the size of each minimal dependent set is at most three. We provide a randomized NC algorithm for the case

*Research supported by NSF grant CCR-8858799.

†on leave of absence from the University of Toronto, research partially supported by N.S.E.R.C. of Canada operating grant A8092.

when the size of each minimal dependent set is at most a constant, and we conjecture that this algorithm is a randomized NC algorithm for the general case.

1 Introduction

Let $G = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$. A set $I \subseteq V$ is *independent* if for all $e \in E$, the two endpoints v and w of e are not both in I. An independent set I is *maximal* if for all $v \in V - I$, $I \cup \{v\}$ is not an independent set, i.e. for every $v \in V - I$ there is some $w \in I$ such that $(v, w) \in E$. The *maximal independent set given an undirected graph problem* is to find a maximal independent set in an undirected graph G presented by an explicit list for V and E. Although there is a trivial greedy linear time algorithm, it required quite a bit of effort to develop a fast parallel algorithm (an efficient NC algorithm) for the maximal independent set given an undirected graph problem [KW, Lu1, ABI, GS, Lu2].

The work in this paper studies a natural generalization of the maximal independent set for undirected graphs problem, the *maximal independent set given c-size minimal dependent sets problem.* The input to the problem is $H = (V, E)$ and the output is a maximal independent set in H. Here, E is a collection of m subsets of V, each of size at most c. (The maximal independent set given an undirected graph problem is the special case where $c = 2$.) A set $I \subseteq V$ is *independent* if for all $e \in E$ there is at least one $v \in e$ such that $v \notin I$. An independent set I is *maximal* if for every $v \in V - I$, $I \cup \{v\}$ is not an independent set, i.e. there is some $e \in E$ such that $e \subseteq I \cup \{v\}$. In this paper we suggest a very simple parallel randomized algorithm for this problem. Although we conjecture the algorithm has expected running time polynomial in $\log n$ even for the most general case of problem instances (when c is allowed to be n), we are able to only prove that this is the case when the maximum edge dimension c is restricted to be a constant.

The maximal independent set given minimal dependent sets problem can be thought of in two ways. The first is to view H is a hypergraph, where E is the set of hyperedges and c is the maximum dimension of any hyperedge in E. This is the terminology that we use in the remainder of the paper. The second is to view H as an independence system, where V is the ground set and E is an explicit list of the minimal dependent sets. Karp, Upfal and Wigderson [KUW] investigate the complexity of a related problem. They consider the parallel complexity of finding maximal independent sets in an independence system on a ground set of size n but

where the only way to gain information about the independence system is via calls to an oracle that accepts as input a subset of V and answers whether or not it is independent. Besides the obvious difference in the input specifications, the main difference between our problem and their problem is how the time for finding the maximal independent set is expressed. [KUW] express the time simply in terms of the number of elements in the ground set, whereas we express time in terms of the size of the ground set plus a measure of the complexity of the independence system, i.e. in terms of the size of the description of all the minimal dependent sets. [KUW] prove that any *deterministic* parallel algorithm with p processors and access to the oracle takes time $\Omega\left(\frac{n}{\log(np)}\right)$. On the other hand, they show that there is a *randomized* parallel algorithm with n processors and access to the oracle that takes time $O(\sqrt{n})$. In contrast, we make a conjecture on the time to find a maximal independent set expressed in terms of our measure of the complexity of the independence system: we conjecture that there is a randomized parallel algorithm using $O(n+m)$ processors with running time polynomial in $\log(n+m)$, where n is the size of the ground set and m is the number of minimal dependent sets. (Our main result is that this conjecture is true with a polynomial number of processors if the size of each minimal dependent set is restricted to be a constant.) In light of these results, it would be interesting to consider proving lower bounds on the time complexity of finding a maximal independent set in the oracle model introduced in [KUW] expressed in terms of a natural measure of the complexity of the independence system (e.g. in terms of the number of minimal dependent sets in the independence system) instead of simply in terms of the number of elements in the ground set.

2 Notation

Let $H = (V, E)$, where $|V| = n$ and $|E| = m$. E is a collection of m subsets of V, each such subset is called a *hyperedge*, or more simply an *edge*. The *dimension* of an edge $e \in E$, $dim(e)$, is $|e|$. For all $s \subseteq V$, for all $j = 1, \ldots, n$, let

$$N_j(s) = \{t \subseteq V : t \cup s \in E \text{ and } t \cap s = \emptyset \text{ and } |t| = j\},$$

and let

$$d_j(s) = |N_j(s)|^{\frac{1}{j}}$$

be the *normalized degree* of s with respect to dimension $|s|+j$ edges. For example, $N_1(\{v\})$ is the set of $\{w\}$ such that $\{v, w\}$ is a dimension two edge and $d_1(\{v\})$ is the number of dimension two edges that include v, $N_1(\{v, w\})$ is the set of

$\{z\}$ such that $\{v, w, z\}$ is a dimension three edge and $d_1(\{v, w\})$ is the number of dimension three edges that include both v and w, whereas $N_2(\{v\})$ is the set of $\{w, z\}$ such that $\{v, w, z\}$ is a dimension three edge and $d_2(\{v\})$ is the square root of the number of dimension three edges that include v. For all $j = 2, \ldots, n$, let

$$\Delta_j = \max_{s \subseteq V \text{ and } |s|<j} \{d_{j-|s|}(s)\}$$

be the *maximum normalized degree* with respect to dimension j edges. Finally, we let

$$\Delta = \max_{j=2,\ldots,n} \{\Delta_j\}$$

be the maximum normalized degree. All of these definitions are with respect to H implicitly. When necessary, we explicitly specify the hypergraph to which the definitions apply, e.g. $\Delta_j(H)$ and $\Delta(H)$ are defined to be Δ_j and Δ, respectively, with respect to H.

3 Overall Algorithm Structure

In this section we describe the structure for all the algorithms and introduce two randomized parallel algorithms for the maximal independent set for hypergraphs problem. We conjecture that at least one of these two algorithms runs in expected time polynomial in $\log n$ for the general case. In the next section we prove that the second of these two algorithms does have this expected running time when the maximum dimension of an edge is restricted to three.

All of the algorithms share the following overall structure on input $H = (V, E)$. They work in stages, and at the beginning of each stage the pair (I, H'), where I is an independent set in H and $H' = (V', E')$ is a subhypergraph of H, satisfies the following invariant: If I' is a maximal independent set in H' then $I \cup I'$ is a maximal independent set in H. Before the beginning of the first stage, I is initialized to the empty set and H' is initialized to H. At the end of the algorithm I is a maximal independent set in H and H' is empty. At each stage vertices are added to I and H' is pruned so that the invariant remains true. In particular, the vertices added to I are removed from both V' and from the edges in E' in which they are contained. As vertices are removed from an edge the dimension of the edge is reduced. Thus, it is possible that some of the edge constraints become redundant and can be removed. In particular, for all pairs of edges $e, e' \in E'$, if $e \subset e'$ then the constraint imposed by e is strictly stronger than the constraint imposed by e' and e' is removed from E'. When an edge is reduced to a dimension

one edge $e = \{v\}$, v is removed from V' (because the constraint imposed by e ensures that v can never be added to I) and e is removed from E'.

The difference between the algorithms is the procedure for finding the vertices to add to I in a stage. The *permutation algorithm*, which is a natural generalization of the permutation algorithm described in [Lu1], chooses these vertices as follows. (As shown in [Lu1], the expected running time of the permutation algorithm is provably polynomial in $\log n$ when the maximum dimension of an edge is two.) A random ordering (or permutation) π of V' is chosen such that each possible ordering of the vertices is equally likely. This can be implemented by having each vertex $v \in V'$ independently and randomly choose $\pi(v)$ uniformly from a suitably large range of integers so that it is unlikely that two vertices choose the same number (e.g. the range from $1, \ldots, n^4$ would guarantee that the probability there is a pair of vertices that choose the same number is at most $\frac{1}{n^2}$) and then use the value of $\pi(v)$ to determine the relative ordering of v with respect to the other vertices in V'. For each $v \in V'$, let $b(v)$ be a boolean variable that is initialized to **true**. For each $e \in E'$, find $v \in e$ such that $\pi(v) = \max_{w \in e}\{\pi(w)\}$ and set $b(v) =$ **false**. The vertices added to I in the stage are those such that $b(v)$ remains **true**.

The *maximum degree algorithm* chooses the vertices to add to I in a stage as follows. Let $p = \frac{1}{8\Delta(H')}$. Each vertex $v \in V'$ independently and randomly chooses $b(v)$ to be **true** with probability p and $b(v)$ to be **false** with probability $1-p$. For each edge $e \in E'$, if $b(v)$ is chosen to be **true** for all $v \in e$ then $b(v)$ is reset to **false** for all $v \in E'$. The vertices added to I in the stage are those such that $b(v)$ remains **true**.

We conjecture that the expected number of stages for at least one of these two algorithms is polynomial in $\log n$. In fact, the two algorithms in many ways are remarkably similar. We remark why this conjecture might be true for the maximum degree algorithm. Let H' be the hypergraph at the beginning of a stage. We can prove that $\Delta(H')$ decreases to zero after a number of stages polynomial in $\log n$, *if we ignore the migration of edges from one dimension to a smaller dimension due to loss of vertices that are added to the independent set.* (Our belief is that this migration should only make the algorithm work faster, however we do not know how to prove this.)

4 Dimension Three Hypergraphs

In this section we describe and analyze the maximum degree algorithm for finding a maximal independent set in hypergraphs with maximum edge dimension three.

The following *Main Algorithm* gives the overall structure; the only part of this algorithm specific to the maximum degree algorithm for hypergraphs with edges of dimension at most three is how the procedure *FindInd* is implemented.

Main Algorithm

$I \leftarrow \emptyset$
$H' = (V', E') \leftarrow H = (V, E)$
comment: *A stage is an interation of the while loop*
while $H' \neq \emptyset$ **do**
  call $FindInd(H', I')$
  $I \leftarrow I \cup I'$
  comment: *Remove vertices in I' from H'*
  $V' \leftarrow V' - I'$
  for all $e \in E'$ **do** $e \leftarrow e - I'$ **end for**
  comment: *Remove redundant edges*
  for all $e, e' \in E'$ **do**
    if $e \subset e'$ **then** $E' \leftarrow E' - e'$ **end if**
  end for
  comment: *Remove dimension one edges from H'*
  for all $e = \{v\} \in E'$ *of dimension one* **do**
    $E' \leftarrow E' - \{e\}$
    $V' \leftarrow V' - \{v\}$
  end for
end while

Procedure *FindInd* given below is used to choose the vertices I' that are to be added to I during a stage of the algorithm when H is a hypergraph with all edges of dimension at most three.

Procedure FindInd(H', I')

comment: *The next two loops are to compute* Δ
for all $v \in V'$ **do**
$d_1(\{v\}) \leftarrow |N_1(\{v\})|$
$d_2(\{v\}) \leftarrow |N_2(\{v\})|^{\frac{1}{2}}$
end for
for all $v, w \in V', v \neq w$ **do**
$d_1(\{v, w\}) \leftarrow |N_1(\{v, w\})|$
end for
$\Delta \leftarrow \max_{v,w \in V'}\{d_1(\{v\}), d_2(\{v\}), d_1(\{v, w\})\}$
$p \leftarrow \frac{1}{8\Delta}$
comment: *Choice Step*
for all $v \in V'$ **do independently choose**

$$b(v) = \begin{cases} \textbf{true} & \textit{with probability } p \\ \textbf{false} & \textit{with probability } 1-p \end{cases}$$

end for
comment: *Elimination Step*
for all $e \in E'$ **do**
if $b(v) =$ **true** *for all* $v \in e$ **then do**
for all $v \in e$ **do** $b(v) \leftarrow$ **false end for**
end if
end for
$I' \leftarrow \{v \in V' \mid b(v) = \textbf{true}\}$

In the following all quantities are defined with respect to H' during an execution of procedure *FindInd*. For all $v \in V'$ we say v *is chosen* if $b(v)$ is set to **true** in the *Choice Step* of *FindInd*, and in this case we say that event C_v occurs. We say v *is eliminated* if $b(v)$ is reset to **false** in the *Elimination Step* of *FindInd*, and in this case we say that event E_v occurs. Let A_v be the event that v is chosen but not eliminated, i.e. $A_v = C_v \wedge \neg E_v$.

Lemma 1

(a) For any vertex v, $\Pr[E_v | C_v] \leq \frac{1}{4}$.

(b) For any pair of distinct vertices v *and* w *such that* $\{v, w\} \notin E'$,

$$\Pr[E_v \vee E_w | C_v \wedge C_w] \leq \frac{1}{2}.$$

Proof:

(a) If vertex v is chosen, then it is eliminated if there is some edge e such that $v \in e$ and all members of e are chosen. The probability that there is such a two dimensional edge is

$$\Pr[\exists \{x\} \in N_1(\{v\}) \text{ such that } C_x] \leq |N_1(\{v\})|p \leq \frac{d_1(\{v\})}{8\Delta} \leq \frac{1}{8}.$$

The probability that there is such a three dimensional edge is

$$\Pr[\exists \{x,y\} \in N_2(\{v\}) \text{ such that } C_x \wedge C_y] \leq |N_2(\{v\})|p^2 \leq \frac{d_2(\{v\})^2}{64\Delta^2} \leq \frac{1}{64}.$$

Thus for any vertex v, $\Pr[E_v|C_v] \leq \frac{1}{8} + \frac{1}{64} \leq \frac{1}{4}$.

(b) Suppose both v and w are chosen. Vertex v or vertex w is eliminated if there is some edge e such that all members of e are chosen and either $v \in e$ or $w \in e$. There are five cases to consider:

(1) e is a two dimensional edge containing v but not w,

(2) e is a two dimensional edge containing w but not v,

(3) e is a three dimensional edge containing v but not w,

(4) e is a three dimensional edge containing w but not v and

(5) e is a three dimensional edge containing both v and w.

We upper bound the sum of the probability of these five events. By the proof of part (a), an upper bound on (1) is $\frac{1}{8}$, an upper bound on (2) is $\frac{1}{8}$, an upper bound on (3) is $\frac{1}{64}$ and an upper bound on (4) is $\frac{1}{64}$. The probability of (5) is

$$\Pr[\exists \{x\} \in N_1(\{v,w\}) \text{ such that } C_x] \leq |N_1(\{v,w\})|p \leq \frac{d_1(\{v,w\})}{8\Delta} \leq \frac{1}{8}.$$

Thus, the sum of the five probabilities is at most $\frac{1}{2}$.

□

Lemma 2

(a) For any vertex v such that $d_1(\{v\}) \geq \epsilon\Delta$,

$$\Pr[\exists\{x\} \in N_1(\{v\}) \text{ such that } A_x] \geq \frac{\epsilon}{16}.$$

(b) For any vertex v such that $d_2(\{v\}) \geq \epsilon\Delta$,

$$\Pr[\exists\{x,y\} \in N_2(\{v\}) \text{ such that } A_x \wedge A_y] \geq \frac{\epsilon^2}{256}.$$

(c) For any pair of distinct vertices v and w such that $d_1(\{v,w\}) \geq \epsilon\Delta$,

$$\Pr[\exists\{x\} \in N_1(\{v,w\}) \text{ such that } A_x] \geq \frac{\epsilon}{16}.$$

Proof:

(a) By the principle of inclusion-exclusion,

$$\Pr[\exists\{x\} \in N_1(\{v\}) \text{ such that } A_x] \geq$$

$$\sum_{\{x\} \in N_1(\{v\})} \Pr[A_x] - \sum_{\{x\},\{y\} \in N_1(\{v\}) \wedge \{x\} \neq \{y\}} \Pr[A_x \wedge A_y].$$

By Lemma 1 part (a), for each $\{x\} \in N_1(\{v\})$,

$$\Pr[A_x] = \Pr[C_x] \Pr[\neg E_x | C_x] \geq \frac{3}{32\Delta}.$$

For each pair of distinct $\{x\}$ and $\{y\}$ in $N_1(\{v\})$,

$$\Pr[A_x \wedge A_y] = \Pr[C_x \wedge \neg E_x \wedge C_y \wedge \neg E_y] \leq \Pr[C_x \wedge C_y] \leq \frac{1}{64\Delta^2}.$$

Thus,

$$\Pr[\exists\{x\} \in N_1(\{v\}) \text{ such that } A_x] \geq \frac{|N_1(\{v\})|}{\Delta}\left(\frac{3}{32} - \frac{|N_1(\{v\})|}{64\Delta}\right).$$

Because $\epsilon\Delta \leq |N_1(\{v\})| \leq \Delta$, this last quantity is at least $\frac{\epsilon}{16}$.

(b) By the principle of inclusion-exclusion,

$$\Pr[\exists\{x,y\} \in N_2(\{v\}) \text{ such that } A_x \wedge A_y] \geq$$

$$\sum_{\{x,y\}\in N_2(\{v\})} \Pr[A_x \wedge A_y] - \sum_{\{x,y\},\{w,z\}\in N_2(\{v\})\wedge\{x,y\}\neq\{w,z\}} \Pr[A_x \wedge A_y \wedge A_w \wedge A_z].$$

By Lemma 1 part (b), for each $\{x,y\} \in N_2(\{v\})$,

$$\Pr[A_x \wedge A_y] = \Pr[C_x \wedge C_y] \Pr[\neg(E_x \vee E_y) | C_x \wedge C_y] \geq \frac{1}{128\Delta^2}.$$

For each pair of distinct $\{x,y\}$ and $\{w,z\}$ in $N_2(\{v\})$,

$$\Pr[A_x \wedge A_y \wedge A_w \wedge A_z] \leq \Pr[C_x \wedge C_y \wedge C_w \wedge C_z].$$

The quantity

$$\sum_{\{x,y\},\{w,z\}\in N_2(\{v\})\wedge\{x,y\}\neq\{w,z\}} \Pr[C_x \wedge C_y \wedge C_w \wedge C_z]$$

can be partitioned into the following quantities:

(1) All four vertices x, y, w, and z are distinct. There are at most $|N_2(\{v\})|^2$ such terms, and each term is at most $\frac{1}{4096\Delta^4}$.

(2) Among the four vertices there are three distinct vertices, renamed to be x, y and z. We can enumerate all possible triples as follows. For each $\{x,y\} \in N_2(\{v\})$, we can choose $\{z\} \in N_1(\{v,x\})$ or $\{z\} \in N_1(\{v,y\})$. With these choices there are exactly two ways that the same triple can be chosen and, for each possible $\{v,x\}$, $|N_1(\{v,x\})| \leq \Delta$. Consequently the number of terms is at most $|N_2(\{v\})|\Delta$. Each term is at most $\frac{1}{512\Delta^3}$.

Thus,

$$\Pr[\exists\{x,y\} \in N_2(\{v\}) \text{ such that } A_x \wedge A_y]$$

is at least

$$\frac{|N_2(\{v\})|}{\Delta^2}\left(\frac{1}{128} - \frac{|N_2(\{v\})|}{4096\Delta^2} - \frac{1}{512}\right).$$

Because $\epsilon^2\Delta^2 \leq |N_2(\{v\})| \leq \Delta^2$, this quantity is at least $\frac{\epsilon^2}{256}$.

(c) Exactly the same argument as for proof of part (a), replacing $N_1(\{v\})$ by $N_1(\{v,w\})$.

□

Lemma 3 *Let H' be the input to FindInd.*

(a) For any vertex v with $d_1(\{v\}) \geq \epsilon\Delta$,

$$\Pr[\textit{ all edges containing } v \textit{ are removed during this stage }] \geq \frac{\epsilon}{16}.$$

(b) For any vertex v such that $d_2(\{v\}) \geq \epsilon\Delta$,

$$\Pr[\textit{ all edges containing } v \textit{ are removed during this stage }] \geq \frac{\epsilon^2}{256}.$$

(c) For any pair of distinct vertices v and w such that $d_1(\{v,w\}) \geq \epsilon\Delta$,

$$\Pr[\textit{ all edges containing both } v \textit{ and } w \textit{ are removed during this stage }] \geq \frac{\epsilon}{16}.$$

Proof:

(a) All edges containing v are removed during this stage if there is some two dimensional edge $e = \{v, x\}$ such that A_x. By Lemma 2 part (a), this happens with probability at least $\frac{\epsilon}{16}$.

(b) All edges containing v are removed during this stage if there is some three dimensional edge $e = \{v, x, y\}$ such that $A_x \wedge A_y$. By Lemma 2 part (b), this happens with probability at least $\frac{\epsilon^2}{256}$.

(c) All edges containing v and w are removed during this stage if there is some three dimensional edge $e = \{v, w, x\}$ such that A_x. By Lemma 2 part (c), this happens with probability at least $\frac{\epsilon}{16}$.

□

The rest of the analysis would be straightforward except for the fact that dimension two edges can be created during the execution of the algorithm. We now outline this straightforward analysis, assuming that no new dimension two edges are ever created. (The actual analysis is similar in spirit but more complicated due to the creation of dimension two edges.) Let X be the set of all vertices and all pairs of vertices in V. The cardinality of X is $O(n^2)$. For each $s \in X$, we define the *degree of* s with respect to H' as follows:

(a) For all $\{x\} \in X$, the *degree of* $\{x\}$ is the maximum of $d_1(\{x\})$ and $d_2(\{x\})$.

(b) For all $\{x, y\} \in X$, the *degree of* $\{x, y\}$ is $d_1(\{x, y\})$.

Since $\Delta(H) \leq n$, for all $s \in X$ and during all stages the degree of s is at most n. We divide the execution of the algorithm into epochs. The stages belonging to the i^{th} epoch (for $i = 1, \ldots, \lceil \log n \rceil + 1$) are those such that at the beginning of the stage $\frac{n}{2^{i-1}} \geq \Delta(H') > \frac{n}{2^i}$. (Some epochs may contain no stages, and when $\Delta(H') < 1$ then the current stage is the final stage). Fix an epoch i. Let Δ' be the value of Δ at the beginning of the first stage in epoch i. For all $j \geq 1$, let H^j be H' at the beginning of the j^{th} stage after the start of epoch i and let $\Delta^j = \Delta(H^j)$. Let

$$X^j = \{s \in X : \text{ degree of } s \text{ in } H^j \geq \frac{\Delta'}{2}\}.$$

Because for now we are making the (false) assumption that no new edges are ever created during the running of the algorithm, $\Delta^1 \geq \Delta^2 \geq \ldots$. For every $s \in X^j$, it is easy to see (using Lemma 3 and the fact that the degree of s is at least $\frac{\Delta'}{2} = \frac{\Delta^1}{2} \geq \frac{\Delta^j}{2}$) that the probability the degree of s decreases to at most one during stage j in epoch i is a constant. Thus on average some constant fraction of the L^j are not members of L^{j+1}. Because $|L^1| \leq |X|$ and $|X|$ is $O(n^2)$, the expected number of stages before a stage j is reached in epoch i such that L^j is empty is $O(\log n)$. The smallest j such that L^j is empty initiates a new epoch. Thus, the expected number of stages in each epoch is $O(\log n)$ and there are $O(\log n)$ epochs for a total expected number of stages during the entire algorithm of $O(\log^2 n)$.

The analysis just given is correct except for the fact that it ignores the creation of new dimension two edges as follows: When exactly one vertex of the three vertices of a dimension three edge is added to the independent set then a new dimension two edge is created. Thus the number of dimension two edges can actually increase during the execution of the algorithm. The real problem is that these additional dimension two edges might actually cause the value of $\Delta(H')$ to increase from one stage to the next. Lemma 4 given below is used to prove that this behavior is limited.

We now briefly outline the proof that the algorithm has expected running time polynomial in $\log n$. Recall that $\Delta_2(H')$ and $\Delta_3(H')$ are the maximum normalized degree with respect to dimension two edges and dimension three edges, respectively, in H' (see the Notation section). All of the statements in this paragraph hold with suitably high probability (e.g. $O(1/n^2)$). Lemma 4 can be used to show that as long as $\Delta_2(H')$ is bigger than $\Delta_3(H')$ then $\Delta_2(H')$ does not grow too rapidly, and furthermore that if $\Delta_2(H')$ is more than a polynomial in $\log n$ factor bigger than $\Delta_3(H')$ then (using also Lemma 3) $\Delta_2(H')$ decreases in a polynomial in $\log n$ number of stages until it is within a polynomial in $\log n$ factor of $\Delta_3(H')$.

Finally, using Lemma 3, it is possible to show that after polynomial in $\log n$ stages where $\Delta_2(H')$ is at most a polynomial in $\log n$ factor larger than $\Delta_2(H')$ the value of $\Delta_3(H')$ decreases to zero. Putting all of this together we get that after a polynomial in $\log n$ number of stages $\Delta_3(H')$ is zero. Once $\Delta_3(H')$ is zero, after another $\log^2 n$ stages the algorithm has found a maximal independent set in H. We give these details in the final version of the paper.

Lemma 4 *Fix H' at the beginning of a stage. For any $c \geq 0$, for any $v \in V'$, if $\Delta \geq t \geq \frac{\Delta}{\log^c n}$ and $t \geq \max\{d_2(\{v\}), \max_{w \in V'}\{d_1(\{v,w\})\}\}$ then the probability is at most $\frac{\log n}{n^2}$ that during the stage more than $(5c + 10.5)t \log n$ additional two dimensional edges are produced at v.*

Proof: Fix an $v \in V'$. Let $X = \cup N_2(\{v\})$, i.e. X is the set of all $x \in V'$ such that x is in some dimension three edge that also includes v. For each $x \in X$ we let a_x be the maximum number of dimension two edges that could be created at v if x were added to the independent set. This number a_x is just $d_1(\{v,x\})$ and by assumption this is at most t. Also, $\sum_{x \in X} a_x \leq 2t^2$ because each dimension three edge containing v is accounted for twice in the sum on the left. We overestimate the number of dimension two edges produced at v by viewing the total number of edges produced as being the sum of $|X|$ independent random variables $\{Y_x : x \in X\}$ such that $Y_x = a_x$ with probability $\frac{1}{8\Delta(H')}$ and $Y_x = 0$ otherwise.

We break-up the range of possible values of a_x into $\log t$ groups, $[1,2], (2,4], \ldots, (t/2, t]$. Since $\sum_{x \in X} a_x \leq 2t^2$ there are at most $\frac{2t^2}{2^i}$ vertices x with a_x in the range $(2^i, 2^{i+1}]$. Thus the distribution of $\sum_{x \in X} Y_x$ is bounded above by

$$Y = 2B(2t^2, \frac{1}{8\Delta}) + 4B(t^2, \frac{1}{8\Delta}) + \cdots + tB(4t, \frac{1}{8\Delta})$$

where $B(m,p)$ is binomial distribution which is the sum of m independent variables with individual probability p. The expected value of this distribution is $\frac{4t^2 \log t}{8\Delta} \leq \frac{1}{2} t \log n$.

A group corresponding to the range $(m/2, m]$ contributes $\frac{4t^2}{m} B(m, \frac{1}{8\Delta}) \leq tB(m, \frac{1}{8\Delta})$ to the total. We now consider the distribution $B(m, \frac{1}{8\Delta})$. Now

$$\Pr[B(m, \frac{1}{8\Delta}) \geq s] \leq \binom{m}{s} \left(\frac{1}{8\Delta}\right)^s \leq \left(\frac{me}{8s\Delta}\right)^s < \left(\frac{m}{2s\Delta}\right)^s.$$

We break up consideration of the tail probabilities into 3 cases.

1. For $m/\Delta \geq 2\log n$ we choose $s = m/\Delta$ and get $\Pr[B(m, \frac{1}{\Delta}) \geq s] \leq 2^{-s} \leq 2^{-2\log n} = 1/n^2$.

2. For $m/\Delta \leq \sqrt{\log n}$ choose $s = \frac{5\log n}{\log\log n}$ and obtain

$$\begin{aligned}\Pr[B(m, \frac{1}{8\Delta}) \geq s] < (10\sqrt{\log n}\log\log n)^{-s} &< 2^{-(\frac{1}{2}\log\log n - \log\log\log n)s} \\ &\leq 2^{-\frac{2}{5}s\log\log n} \\ &= 2^{-2\log n} = 1/n^2.\end{aligned}$$

3. For $\sqrt{\log n} < m/\Delta < 2\log n$ choose $s = 2\log n$ and obtain $\Pr[B(m, \frac{1}{8\Delta}) \geq s] < 2^{-s} = 2^{-2\log n} = 1/n^2$.

Assume that each of the binomial distributions being summed takes on a value no more than its corresponding s value. This happens with probability at least $1 - \frac{\log t}{n^2} \geq 1 - \frac{\log n}{n^2}$. For each group corresponding to the first case we see that the distribution is no more than m/Δ and the contribution of its group is at most $(4t^2/m)(m/\Delta) = 4t^2/\Delta \leq 4t$. Thus the total contribution of the first case is no more than $4t\log t \leq 4t\log n$. Since $\Delta \leq t\log^c n$ there are at most $(c+.5)\log\log n$ groups corresponding to the second case. In each of these groups the contribution is no more than $t\frac{5\log n}{\log\log n}$ for a total contribution of at most $5(c+.5)t\log n$. Finally there are at most $.5\log\log n$ groups corresponding to the third case and in each of these the contribution is at most $(4t^2/m)2\log n$. Now since $m/\Delta \geq \sqrt{\log n}$, the contribution of each of these groups is at most $(4t^2/\Delta)\sqrt{\log n} \leq 4t\sqrt{\log n}$. Thus the total contribution from groups corresponding to the third case is no more than $2t\sqrt{\log n}\log\log n < t\log n$ for n sufficiently large. Adding up the contributions of the three cases we get a total Y value of at most $(5c+10.5)t\log n$. □

5 Constant Dimension Hypergraphs

For hypergraphs with dimension at most c, the algorithm is unchanged in overall structure and all that is affected is the computation of Δ and p in *FindInd*. Δ now requires $O(n^c)$ processors and $O(\log n)$ time and the probability p is chosen to be $(2^{c+1}\Delta)^{-1}$ instead of $\frac{1}{8\Delta}$. Corresponding to Lemma 3 above we have for any set X of vertices of size up to $c-1$ the probability that all edges containing X are removed during a stage is at least $\frac{1}{4}(\epsilon/2^{c+1})^j$ if $d_j(X) \geq \epsilon\Delta$. Similarly, an analog of Lemma 4 holds but now we must consider all the possibilities of higher dimensional

edges being partially selected and generating new edges of lower dimension. The overall analysis follows the same structure as the dimension 3 case. Details are left for the full paper.

6 References

[**ABI**] Alon, N., L. Babai, A. Itai, "A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem", *Journal of Algorithms*, 7, pp. 567–583, 1986.

[**GS**] Goldberg, M., T. Spencer, "A New Parallel Algorithm for the Maximal Independent Set Problem", preliminary version in proceedings of 28th Annual Symposium on Foundations of Computer Science, 1987, pp. 161–165, to appear in *SIAM J. on Computing.*

[**KUW**] Karp, R.M., Upfal, E. and Wigderson, A., "Are Search and Decision Problems Computationally Equivalent?", Proc. 17th ACM Symposium on Theory of Computing, 1985, pp. 464–475.

[**KW**] Karp, R.M., A. Wigderson, *A Fast Parallel Algorithm for the Maximal Independent Set Problem*, Proc. 16th ACM Symposium on Theory of Computing, 1984, pp. 266–272.

[**Lu1**] Luby, M., "A Simple Parallel Algorithm for the Maximal Independent Set Problem", *SIAM J. Comput.*, vol. 15, no. 4, November, 1986, pp. 1036-1053.

[**Lu2**] Luby, M., "Removing Randomness in Parallel Computation Without a Processor Penalty", preliminary version in proceedings of 29th Annual Symposium on Foundations of Computer Science, 1988, pp. 162–173, to appear in special issue of *JCSS.*