FINDING LARGE INDEPENDENT SETS IN GRAPHS AND HYPERGRAPHS*

HADAS SHACHNAI† AND ARAVIND SRINIVASAN‡

Abstract. A basic problem in graphs and hypergraphs is that of finding a *large* independent set–one of guaranteed size. Understanding the parallel complexity of this and related independent set problems on hypergraphs is a fundamental open issue in parallel computation. Caro & Tuza (*J. Graph Theory*, Vol. 15, pp. 99–107, 1991) have shown a certain lower bound $\alpha_k(H)$ on the size of a maximum independent set in a given k-uniform hypergraph H, and have also presented an efficient sequential algorithm to find an independent set of size $\alpha_k(H)$. They also show that $\alpha_k(H)$ is the size of the maximum independent set for various hypergraph families. Here, we show that an RNC algorithm due to Beame & Luby (in *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 212–218, 1990) finds an independent set of expected size $\alpha_k(H)$, and also derandomize it for certain special cases. (An intriguing conjecture of Beame & Luby implies that understanding this algorithm better may yield an RNC algorithm to find a maximal independent set in hypergraphs, which is among the outstanding open questions in parallel computation.) We also present lower bounds on independent set size for *non-uniform* hypergraphs using this algorithm. For graphs, we get an NC algorithm to find independent sets of size essentially that guaranteed by the general (degree-sequence based) version of Turán's theorem.

Key words. Independent Sets, Parallel Algorithms, Randomized Algorithms.

AMS subject classifications. 05C65, 60C05, 68R10, 68W10, 68W20.

1. Introduction.

Finding large/maximal independent sets in (hyper)graphs, defined formally below, is a fundamental problem in parallel combinatorial optimization. An outstanding open question in parallel computation is whether a maximal independent set in a given hypergraph can be found in $(R)NC$ [13]. The work of Karp and Wigderson [15] on finding maximal independent sets in graphs in NC, was a breakthrough that inspired several graph-theoretic NC algorithms, and also led to a rich theory of derandomization. The corresponding problems on hypergraphs have applications, e.g., to feasible communication in channelized cellular telephone systems [19], but seem much harder than in the case of graphs. In this work we consider an RNC algorithm for finding large independent sets in hypergraphs due to Beame and Luby [3],[1] and derandomize it for various special cases. We show that this algorithm finds independent sets that have been shown to exist for uniform hypergraphs via a *sequential* algorithm in [5]; we also use the algorithm to give lower bounds on independent set size for certain families of non-uniform hypergraphs. For graphs, our derandomization yields the first NC algorithm to find independent sets of size essentially that guaranteed by the degree-sequence version of Turán's theorem [28].

*Preliminary versions of parts of this work appeared in: (i) A. Srinivasan, New approaches to covering and packing problems, *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 567–576, 2001; and (ii) H. Shachnai and A. Srinivasan, Finding large independent sets of hypergraphs in parallel, *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pp. 163–168, 2001.

† Department of Computer Science, The Technion, Haifa 32000, Israel. E-mail: `hadas@cs.technion.ac.il`. Part of this work was done while the author was on a leave at Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974.

‡ Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA. Most of this work was done while at Bell Laboratories, Lucent Technologies; part of this work was supported by the National Science Foundation under Grant No. 0208005. E-mail: `srin@cs.umd.edu`.

[1] We describe the algorithm in Section 2.1.

Recall that a *hypergraph* $H = (V, E)$ consists of a vertex set V and a collection E of subsets of V; each element of E is called a (hyper)edge. We will only consider finite hypergraphs here, and will throughout denote the number of vertices and edges in a given hypergraph by n and m respectively. An independent set (IS) in H is a subset S of V that does not contain any edge. S is a *maximal independent set* (MIS) if S is an IS, and if no proper superset of S is an IS. It is easy to find an MIS sequentially, but efficient parallel algorithms appear much harder. There has been much work on finding MISs in parallel in graphs (see, e.g., [8, 15, 17, 21]), and in hypergraphs (e.g., [1, 3, 6, 16, 14, 12]); as mentioned above, it is a major open question in parallel computation whether these can be found in RNC on hypergraphs. Since an MIS can be much smaller than a maximum independent set (as in the case of a star graph), it is also of much interest to find ISs that have a guaranteed size. What is known in this context? Recall that given a hypergraph $H = (V, E)$, the *degree* of a vertex is the number of edges that it lies in; H is called k-uniform if all the edges have exactly k elements. Also, for any integer $\ell \geq 0$ and real r,

$$\binom{r}{\ell} \equiv \frac{r(r-1)\cdots(r-l+1)}{\ell!}. \tag{1.1}$$

Caro and Tuza showed in [5] that for any $k \geq 2$, any k-uniform hypergraph H contains an IS of size at least

$$\alpha_k(H) = \sum_{v \in V} \frac{1}{\binom{d(v) + 1/(k-1)}{d(v)}}; \tag{1.2}$$

here and from now on, $d(v)$ denotes the degree of $v \in V$. They showed that this bound is tight for a large class of hypergraphs and also gave a *sequential* algorithm that finds an IS of size $\alpha_k(H)$ in $O(km+n)$ steps. Note that when $k = 2$, the bound in (1.2) reduces to

$$\alpha_2(G) = \sum_{v \in V} \frac{1}{d(v)+1}, \tag{1.3}$$

which is the classical Turán bound for graphs [28]. To give the reader a better feel for (1.2), we point out that

$$\binom{d(v) + 1/(k-1)}{d(v)} = \Theta((d(v))^{1/(k-1)}). \tag{1.4}$$

This can be shown as follows. We set in (1.1) $r = d(v) + 1/(k-1)$ and $\ell = d(v)$ to get

$$\binom{d(v) + 1/(k-1)}{d(v)} = \prod_{i=1}^{d(v)} (1 + \frac{1}{i(k-1)}). \tag{1.5}$$

By a standard expansion of $\ln(1+x)$ for $x \geq 0$ (see, e.g., [24]),

$$\exp(x - x^2/2) \leq 1 + x \leq \exp(x). \tag{1.6}$$

(Here and in what follows, $\exp(x)$ denotes e^x.) Applying (1.6) to (1.5) we have (1.4).

REMARK 1. To handle the case where $d(v) = 0$, the r.h.s. of (1.4) should actually be $\Theta((d(v)+1)^{1/(k-1)})$. However, since vertices of degree zero are irrelevant and can always be added to our independent set, we will always assume that vertex-degrees are non-zero.

Our main results are as follows; these are spelled out in more detail in Section 1.2. Beame & Luby have presented a simple and elegant RNC algorithm for finding an IS in a given hypergraph H [3]; intriguingly, they also conjecture that repeatedly finding and removing such ISs, followed by obvious updates to H, finds an MIS in RNC. (The work of [3] also presents another, more involved, candidate RNC algorithm for finding an MIS in a given hypergraph. By the analysis of [16], this indeed yields an RNC algorithm for MIS in hypergraphs with constant-sized edges.) Thus, it is of interest to understand this algorithm better. We analyze the performance of this algorithm, and show that it finds an IS of expected size $\alpha_k(H)$ for k-uniform hypergraphs, for any $k \geq 2$. We also analyze its performance for certain families of non-uniform hypergraphs. Finally, we derandomize the algorithm for certain special cases; in particular, in the case of graphs G, this yields an NC algorithm to find ISs of size essentially $\alpha_2(G)$.

1.1. Related Work. The following parallel algorithms are known in the case of graphs. Spencer [26] gave an RNC algorithm that yields an IS of expected size $\alpha_2(G)$ in graphs G. Goldberg and Spencer [9] presented an NC algorithm that finds an IS of size at least $\lceil n^2/(2m+n) \rceil$ in any graph G, where n and m denote the number of vertices and edges in G respectively. This bound equals $\alpha_2(G)$ when G is regular; in all other cases, $\alpha_2(G)$ is larger. (In fact, it is not hard to construct graph families for which $\alpha_2(G) = \Theta(n)$ and $n^2/(2m+n)$ is just $\Theta(1)$: see item (i) in Section 1.2.) Alon, Babai and Itai [1] studied the MIS problem on hypergraphs: they gave an NC algorithm that finds an IS of size $c(n^k/m)^{1/(k-1)}$, $0 < c < 1$, in k-uniform hypergraphs with $k \geq 2$ being any constant. The work of Karger and Koller [12] generalized this to arbitrary k.

1.2. Main Results. The RNC algorithm of [3] that we shall analyze, will be defined in Section 2.1; we will refer to it as algorithm $\mathcal{A}$. We show that $\mathcal{A}$ finds an IS of expected size $\alpha_k(H)$ in k-uniform hypergraphs, and also present related results and extensions. Our main contributions are as follows.

(i) The expected size of the IS produced by $\mathcal{A}$ is larger than the size of the IS found by other parallel algorithms for this problem. For instance, $\alpha_k(H)$ is always at least as large as the bound $c(n^k/m)^{1/(k-1)}$ of [1, 12]. Furthermore, one can construct families of k-uniform hypergraphs for which $\alpha_k(H)$ is $\Theta(n)$ while $(n^k/m)^{1/(k-1)}$ is $\Theta(1)$. For instance, in the case of constant k, consider hypergraphs H where:
- the vertex set is partitioned into two equal-sized sets V_1 and V_2, and
- H contains as (hyper)edges all k-sized subsets of V_1 and $n/(2k)$ many pairwise-disjoint k-sized subsets of V_2.

It is easy to check that for such families of hypergraphs, $\alpha_k(H) = \Theta(n)$ and $(n^k/m)^{1/(k-1)} = \Theta(1)$.

(ii) We show how our analysis extends to certain families of non-uniform hypergraphs; we are not aware of any such bounds relating to large independent sets in this non-uniform case.

(iii) Regarding derandomized versions, we have the following. Let C be a suitably large positive constant. Given a graph or hypergraph, let $\epsilon = (n+m)^{-C}$.

For graphs G, we show how to construct an independent set of size at least $(1-\epsilon)\alpha_2(G)$, in NC. For hypergraphs where all the vertex-degrees are at most $O(\lg(n+m))$, we present NC algorithms to construct ISs of size at least $(1-\epsilon)$ times the size guaranteed by our above-mentioned RNC algorithms.

(iv) All of our results extend without any change in the processor or time complexity to their weighted analogs. In the weighted analogs, we are given a non-negative weight for each vertex, and wish to find an IS of large total weight. The only way that we are aware of to extend arbitrary IS algorithms to their weighted analogs is by a suitable (polynomial) blowup in the size of the hypergraph, leading to a loss in efficiency.

One key facilitator of our results is a simple way of generating random permutations that provides sufficient stochastic independence to conduct our analysis (see Lemma 2.1).

2. The RNC Algorithm.

2.1. Algorithms and Tools. The algorithm $\mathcal{A}$ of [3] that we shall analyze, is as follows.[2] Randomly permute the vertices; add a vertex $v \in V$ to the IS iff there is no edge $e \in E$ such that $v \in e$ and v is *last* among the vertices of e in the random order. It is easy to verify that we produce a valid IS in this fashion. Spencer independently considered this algorithm in the case of graphs, and showed that the expected size of the IS found is $\alpha_2(G)$ [26]. (Actually, the algorithm of [3] adds v to the IS iff there is no edge $e \in E$ such that $v \in e$ and v is *first* among the vertices of e in the random order. It is easy to verify that this is equivalent to our description of $\mathcal{A}$ a few lines above; our description is given so that we get a direct generalization of the algorithm of [26].)

Our main analysis tool will be the FKG inequality [7]; for our purposes, we now recall a special case of the inequality. The reader is referred to [2] for more about the inequality. Given a vector $\vec{Y} = (Y_1, Y_2, \ldots, Y_\ell)$ of *independent* random variables $Y_i \in \{0,1\}$ and an event F that is completely determined by the Y_i's, call F *increasing* iff the following holds: for any $\vec{a}$ such that F holds when $\vec{Y} = \vec{a}$, F also holds when $\vec{Y} = \vec{b}$ for any $\vec{b}$ that co-ordinatewise dominates $\vec{a}$ (i.e., $a_i \leq b_i$ for all i). Then, for any collection of *increasing* events $F_1, F_2, \ldots, F_t$, the FKG inequality shows that

$$\mathrm{Prob}\Big(\bigwedge_{i=1}^{t} F_i\Big) \geq \prod_{i=1}^{t} \mathrm{Prob}(F_i). \tag{2.1}$$

A specific way of implementing algorithm $\mathcal{A}$ is given in Figure 2.1. As will be seen in the proof of Lemma 2.1, the method of generating random permutations that we adopt, provides sufficient independence to employ tools such as the FKG inequality.

It is readily seen that $\mathcal{A}$ can be implemented in RNC. For each edge, we first choose the vertex u in it of highest X_u value; removing duplicates from this multiset of chosen vertices (e.g., through sorting) yields the set of vertices that will *not* lie in our IS. This is easily done on a CREW PRAM in $O(\lg(m+n))$ steps, using $(\sum_{e_i \in E} |e_i|) \leq mn$ processors. Also, since this algorithm is simple and just uses some basic primitives, it should be easy to implement in other parallel/distributed settings.

[2] The algorithm is called the *permutation algorithm* in [3].

Algorithm $\mathcal{A}$:
Independently for all $v \in V$ do:
  Sample $X_v \in [0,1)$ using the uniform distribution
    on $[0,1)$.
Define a permutation π of the vertices in which
$\pi(v) < \pi(u)$ *iff* $X_v < X_u$.
$I := \emptyset$;
for all $v \in V$ do
  {
    $j_v = true$.
    For all $e \in E$ such that $v \in e$
      if $\pi(v) = \max_{u \in e} \pi(u)$ then $j_v = false$.
    If j_v then $I := I \cup \{v\}$;
  }
return(I).

FIG. 2.1. *The algorithm $\mathcal{A}$*

The following notation will be used frequently in the rest of this paper: given a hypergraph H, denote by B_v the event that vertex v is in the final IS produced by our algorithm.

REMARK 2. In case generating random reals poses a problem, we can take up the following standard alternative for producing random permutations; this will also be useful in Section 3. Our modified algorithm $\mathcal{A}'$ is as follows. Let $K > 2$ be an arbitrary constant. Instead of choosing the X_v from $[0,1)$, $\mathcal{A}'$ will independently select each X_v uniformly at random from the set $\{0, 1, \ldots, n^K - 1\}$; for any vertex v, we now add v to the IS iff there is no edge e such that $v \in e$, and such that $X_v \geq X_u$ for all $u \in e$. It is easy to verify that we will still produce an IS in this way. Furthermore, let $\mathcal{E}$ be the event that all the X_v are distinct. First, note that for any distinct v and w, $\mathrm{Prob}(X_v = X_w) = 1/n^K$; so, by a union bound,

$$\mathrm{Prob}(\overline{\mathcal{E}}) \leq n^2 \cdot 1/n^K = o(1).$$

Next, if we condition on $\mathcal{E}$, the permutation π of $\mathcal{A}'$ is distributed as a random permutation. In particular, let $\mathrm{Prob}(B_v)$ and $\mathrm{Prob}'(B_v)$ denote the probability of event B_v occurring, when we use algorithms $\mathcal{A}$ and $\mathcal{A}'$ respectively. Then,

$$\mathrm{Prob}'(B_v) \geq \mathrm{Prob}(\mathcal{E}) \cdot \mathrm{Prob}'(B_v \mid \mathcal{E}) = \mathrm{Prob}(\mathcal{E}) \cdot \mathrm{Prob}(B_v) \geq (1 - n^{2-K}) \cdot \mathrm{Prob}(B_v). \tag{2.2}$$

Thus, if $K > 2$, the expected size of the IS constructed by $\mathcal{A}'$ is essentially as large as that constructed by $\mathcal{A}$.

2.2. Analysis of the performance of $\mathcal{A}$. We now analyze the performance of Algorithm $\mathcal{A}$. Our basic tool will be Lemma 2.1. Before presenting the lemma, we recall that a *linear hypergraph* is one in which every pair of distinct edges share at most one vertex. Linear hypergraphs have been studied in the context of parallel construction of MISs, and NC algorithms are known for the MIS problem on linear hypergraphs [18, 27]. Hence, we also see how well algorithm $\mathcal{A}$ does on linear hypergraphs: Lemma 2.1 also helps provide an exact bound on our algorithm's performance in this case.

LEMMA 2.1. *Suppose a vertex v lies in edges $e_1, e_2, \ldots, e_t$, whose respective cardinalities are k_1, k_2, ..., k_t. Then,*

$$Prob(B_v) \geq \int_0^1 [\prod_{i=1}^{t}(1 - x^{k_i - 1})] \ dx.$$

Furthermore, this inequality becomes an equality in the case of linear hypergraphs.

Proof. Recall the random variables X_u from Figure 2.1. The main idea behind our proof is that the computations become tractable once we condition on the value of X_v. As we will see, the fact that the X_u's are independent will help us much: this way of introducing independence into a choice of permutations helps us use tools such as the FKG inequality. Let $x \in [0, 1)$ be arbitrary, and define, for all $u \neq v$, the random variable $Y_u = 1$ if $X_u > x$, and $Y_u = 0$ otherwise. For each edge e, define the random variable $C(e)$ to be 1 if $\max_{u:u \in e} X_u > x$, and $C(e)$ to be 0 otherwise. Then,

$$\mathrm{Prob}(B_v | X_v = x) = \mathrm{Prob}([\bigwedge_{e:v \in e} (C(e) = 1)] | X_v = x).$$

Now, even conditional on $X_v = x$, the random variables Y_u are independent with $\mathrm{Prob}(Y_u = 1) = 1 - x$. Also, conditional on $X_v = x$, each $C(e)$ is determined by the values of the Y_u, and is increasing as a function of the Y_u. Thus, by the FKG inequality,

$$\mathrm{Prob}(B_v | X_v = x) \geq \prod_{i=1}^{t} \mathrm{Prob}(C(e_i) = 1 | X_v = x) = \prod_{i=1}^{t}(1 - x^{k_i - 1}). \tag{2.3}$$

The first part of the lemma now follows from the fact that

$$\mathrm{Prob}(B_v) = \int_0^1 \mathrm{Prob}(B_v | X_v = x) \ dx.$$

It is also easy to check that the inequality in (2.3) becomes an equality in the case of linear hypergraphs. Hence, the inequality of this lemma is in fact an equality for linear hypergraphs. ∎

Applying the linearity of expectation, we get the following theorem on the expected quality of the IS produced by $\mathcal{A}$ for an arbitrary hypergraph:

THEOREM 2.2. *Suppose we are given an arbitrary hypergraph $H = (V, E)$ with a weight $w_v \geq 0$ for each vertex v. Suppose each vertex v lies in $d(v)$ edges, whose cardinalities are $k_{v,1}, k_{v,2}, \ldots, k_{v,d(v)}$. Then, the expected weight of the IS produced by $\mathcal{A}$ is at least*

$$\sum_v (w_v \cdot \int_0^1 [\prod_{i=1}^{d(v)} (1 - x^{k_{v,i} - 1})] \ dx);$$

in the case of linear hypergraphs, this lower bound is an exact bound on the expected weight.

The next theorem considers the performance of $\mathcal{A}$ for unweighted uniform hypergraphs, and shows that the expected size of the IS produced is at least as large as $\alpha_k(H)$:

THEOREM 2.3. *For any $k \geq 2$ and any k-uniform hypergraph H, $\mathcal{A}$ finds an IS of expected size at least $\alpha_k(H)$.*

Proof. We will use the following identity from [10], which holds for any non-negative integer d and any real x that does not lie in the set $\{-d, -d+1, \ldots, 0\}$:

$$\sum_{l=0}^{d} \binom{d}{l} \frac{(-1)^l}{x+l} = \frac{1}{x\binom{d+x}{d}}. \tag{2.4}$$

Specialized to k-uniform hypergraphs, Lemma 2.1 shows that

$$\begin{aligned}
\mathrm{Prob}(B_v) &\geq \int_0^1 (1 - x^{k-1})^{d(v)}\, dx \\
&= \sum_{l=0}^{d(v)} (-1)^l \binom{d(v)}{l} \int_0^1 x^{(k-1)l}\, dx \\
&= \sum_{l=0}^{d(v)} (-1)^l \binom{d(v)}{l} \frac{1}{1+(k-1)l} \\
&= \binom{d(v) + 1/(k-1)}{d(v)}^{-1},
\end{aligned}$$

by (2.4).

Summing over all the vertices and applying the linearity of expectation completes the proof. ∎

REMARK 3. If an edge is just a singleton $\{v\}$, then vertex v cannot lie in any IS. Hence, whenever we calculate $\mathrm{Prob}(B_v)$ below, we assume without loss of generality that all edges that v lies in have size at least 2.

Any algorithm that works for k-uniform hypergraphs and whose output solution is a non-increasing function of each vertex-degree (as is the function $\alpha_k(H)$), can be immediately extended to give the same guarantee for hypergraphs in which all edges have size *at least* k. We simply replace each edge by an arbitrary subset of it of size k, to achieve this. Thus, our results such as Theorem 2.3 also hold for hypergraphs with at least k vertices in each edge. However, in various families of non-uniform hypergraphs we can do better than this simple approach, as we demonstrate in Theorem 2.4.

We need some notation. Suppose each vertex v lies in $D(j, v)$ edges of cardinality $k(j, v)$, for $j = 1, 2, \ldots, a(v)$. (In other words, vertex v lies in edges of $a(v)$ different sizes: $k(j, v)$, for $j = 1, 2, \ldots, a(v)$. If H is a uniform hypergraph, then $a(v) \equiv 1$.) As mentioned in Remark 3, we assume that $k(j, v) > 1$ for all j, v. Define

$$f(v) = \min_{j=1,2,\ldots,a(v)} [D(j, v)]^{-1/(k(j,v)-1)},$$

and let $b(v) = \min_j (k(j, v) - 1)$. Then, Theorem 2.4 shows that given a weight $w(v)$ for each vertex, $\mathcal{A}$ produces an IS of expected total weight at least $\Omega(\sum_v (w(v)/a(v)^{1/b(v)}) \cdot f(v))$. We prove this by lower-bounding the quantity guaranteed by Theorem 2.2.

Our proof of Theorem 2.4 shows that the quantity guaranteed by Theorem 2.2 is $O(\sum_v w(v)f(v))$; so our "closed-form" bound $\Omega(\sum_v (w(v)/a(v)^{1/b(v)}) \cdot f(v))$ approximates the guarantee of Theorem 2.2 well when $a(v)$ is "small" or $b(v)$ is "large".

THEOREM 2.4. *In a hypergraph $H = (V, E)$, let the notation $a(v)$, $b(v)$ and $f(v)$ be as above. Then, given weights $w(v) \geq 0$ for the vertices, the expected weight of the IS produced by $\mathcal{A}$ is at least $\Omega(\sum_v (w(v)/a(v)^{1/b(v)}) \cdot f(v))$.*

Proof. Fix a vertex v. For notational simplicity, let $a = a(v)$, $b = b(v)$, $k(j) = k(j,v)$, $D(j) = D(j,v)$, and $f = f(v)$. By Theorem 2.2, it suffices to show that

$$\beta \doteq \int_0^1 [\prod_{j=1}^{a} (1 - x^{k(j)-1})^{D(j)}] \, dx \geq \Omega(f/a^{1/b}). \tag{2.5}$$

Let $s = s(v) = \text{argmin}_{j=1,2,\ldots,a} [D(j)]^{-1/(k(j)-1)}$. The bound (1.4) and the proof of Theorem 2.3 help show that

$$\beta \leq \int_0^1 (1 - x^{k(s)-1})^{D(s)} \, dx = \Theta(f);$$

thus, (2.5) is tight to within a constant factor if, e.g., $a(v)$ is bounded by an absolute constant (in particular, (2.5) is a tight bound for hypergraphs of constant maximum degree), or if $b(v) = \Omega(\log n)$.

We now prove (2.5). Let $t = f/(2 \cdot a^{1/b})$, and note that $t \in [0, 1/2]$. Define $\gamma \doteq \prod_{j=1}^{a} (1 - t^{k(j)-1})^{D(j)}$. Note that $1 - z \geq \exp(-2z)$ for $0 \leq z \leq 1/2$. Thus we have

$$\begin{aligned}
\gamma &\geq \exp(-O(\sum_{j=1}^{a} D(j) t^{k(j)-1})) \\
&\geq \exp(-O(\sum_{j=1}^{a} D(j) \frac{f^{k(j)-1}}{a})) \quad \text{(by the definition of } b) \\
&= \exp(-O(\frac{1}{a} \cdot \sum_{j=1}^{a} D(j) f^{k(j)-1})) \\
&\geq \exp(-O(\frac{1}{a} \cdot \sum_{j=1}^{a} 1)) \quad \text{(by the definition of } f) \\
&= \Omega(1).
\end{aligned}$$

Thus,

$$\beta \geq \int_0^t [\prod_{j=1}^{a} (1 - x^{k(j)-1})^{D(j)}] \, dx \geq \int_0^t \gamma \, dx = t\gamma \geq \Omega(t),$$

establishing (2.5). ∎

3. NC algorithms. Recall algorithm $\mathcal{A}'$ of Section 2.1, which is presented soon after defining algorithm $\mathcal{A}$. Note in particular from (2.2) that the expected size of the IS constructed by $\mathcal{A}'$ is essentially as large as that constructed by $\mathcal{A}$. We now take up the task of derandomizing algorithm $\mathcal{A}'$. We will employ an "automata-fooling" approach of [22, 23, 12, 20]; see [25] for a different perspective on this approach through

Logspace-computable statistical tests. Specializing this approach for our purposes, we have the following. Suppose we have h finite-state automata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_h$ with respective state-sets $S_1, S_2, \ldots, S_h$, such that $S_i \cap S_j = \emptyset$ if $i \neq j$. Each S_i is partitioned into $n+1$ layers, numbered $0, 1, \ldots, n$. Layer 0 has a unique state s_i, which is also the start-state of $\mathcal{A}_i$. All transitions are only from one layer to the layer numbered one higher; there are no transitions from layer n. Outgoing arcs from a state are numbered by an integer in the range $\{0, 1, \ldots, 2^d - 1\}$, for some integer d. Given a word $\gamma_1\gamma_2 \ldots \gamma_n$ where each γ_i is a d-bit string, each $\mathcal{A}_i$ moves from its start-state s_i to some state in layer n of S_i, in the obvious way. (In case some γ_i does not correspond to a valid transition, $\mathcal{A}_i$ moves to a unique "reject" state at which it remains from then on.) Now suppose we are given a parameter $\epsilon \in (0, 1)$. Let $R = r + 2^d + 1/\epsilon$, where $r = \sum_i |S_i|$. Then, there is an explicit deterministic parallel algorithm to construct a certain multiset T whose elements are from the set $\{0, 1, \ldots, 2^d - 1\}^n$ such that the total cardinality of T is at most poly(R) [22, 23, 12, 20]; this algorithm uses poly(R) processors and runs in polylog(R) time. The key property of T is as follows. Given a state t in layer n of S_i, let $p_1(i,t)$ be the probability of reaching state t, if we choose $\gamma_1\gamma_2 \ldots \gamma_n$ uniformly at random from $\{0, 1, \ldots, 2^d - 1\}^n$; let $p_2(i,t)$ be this probability if $\gamma_1\gamma_2 \ldots \gamma_n$ is chosen uniformly at random from T. Then, T has the useful property that

$$\forall(i,t),\ |p_1(i,t) - p_2(i,t)| \leq \epsilon. \tag{3.1}$$

We will show how to adapt the above framework to derandomize some of our results. In our applications, automaton $\mathcal{A}_i$ will basically correspond to running $\mathcal{A}'$ and deciding if a vertex i is put in the IS or not by $\mathcal{A}'$; in particular, we will have $h = n$. Fix an automaton $\mathcal{A}_i$. The transitions from layer $j-1$ to j in it, will correspond to the various choices of the value X_j. (Since X_j is chosen at random from the set $\{0, 1, \ldots, n^K - 1\}$, the value d will be $\Theta(\lg n)$.) Now suppose we can construct a layered automaton $\mathcal{A}_i$ of this type with poly$(n+m)$ states, such that the final state entered by $\mathcal{A}_i$ determines whether $\mathcal{A}'$ puts the vertex i in the IS or not. Then we can choose ϵ to be of the form $(n+m)^{-C}$, where C is a suitably large positive constant; by (3.1), we will have a polynomial-sized sample space T constructible in NC, such that sampling from T produces an IS of sufficiently large expected size. By trying out all points in T, we will achieve the desired NC–derandomization.

Thus, the question boils down to the following: for what families of hypergraphs do *polynomial-sized* layered automata of the above type exist? We show that this property holds for two families of hypergraphs:

(H1) graphs, and

(H2) hypergraphs where, for some constant c, all the vertex-degrees are bounded by $c \cdot \lg(n+m)$.

Fix a vertex i. By renumbering the vertices (just for the sake of this automaton), we will assume that $i = 1$. The automaton $\mathcal{A}_i$ for both these families of hypergraphs will have the following structure. Layer 0 has the unique start-state s_i. Layer 1 has n^K states numbered $s_{1,0}, s_{1,1}, \ldots, s_{1,n^K-1}$; $\mathcal{A}_i$ (i.e., $\mathcal{A}_1$ after our renumbering) will transit from s_i to state $s_{1,\ell}$ in layer 1 iff $X_i = \ell$; this way, the automaton can be made to remember the value of X_i. Layer n has two states "accept$_{i,n}$" and "reject$_{i,n}$": for a given choice of all the random variables X_v, $\mathcal{A}_i$ will enter these states if $\mathcal{A}'$ respectively puts i, or does not put i, in the IS. Consider any other layer j of $\mathcal{A}_i$ (i.e., $2 \leq j \leq n-1$). This layer has a state "reject$_{i,j}$" as well as a polynomial number of other states. All transitions from reject$_{i,j}$ are to the state reject$_{i,j+1}$ in the next layer. The invariant we wish to maintain is that:

(P): For $j = 2, 3, \ldots, n$, $\mathcal{A}_i$ enters state reject$_{i,j}$ for a given sequence of values for the X_v iff, just by inspecting the value of X_1 (i.e., X_i) and those of $X_2, X_3, \ldots, X_j$, we have with probability one that $\mathcal{A}'$ will not put vertex i in the IS.

We may use the non-reject states to remember some useful information, to assist us in maintaining the invariant **(P)**.

The automata-construction is simple in the case of graphs: $\mathcal{A}_i$ simply has two states ok-so-far$_{i,j}$ and reject$_{i,j}$ in layer j, for $j = 2, 3, \ldots, n$, and the states ok-so-far$_{i,n}$ and accept$_{i,n}$ are identical. As above, all transitions from reject$_{i,j}$ go to reject$_{i,j+1}$. There is a transition from state $s_{1,\ell}$ in layer 1 to reject$_{i,2}$ iff vertex 2 is a neighbor of i, and if $X_2 \leq X_1 (= \ell)$; otherwise, we transit from $s_{1,\ell}$ to ok-so-far$_{i,2}$. In general, there is a transition from ok-so-far$_{i,j-1}$ to reject$_{i,j}$ iff vertex j is a neighbor of i, and if $X_j \leq X_i$; otherwise, we will transit from ok-so-far$_{i,j-1}$ to ok-so-far$_{i,j}$. It is easy to see that this construction satisfies **(P)**, and so we are done in the case of graphs.

For the family of hypergraphs (H2), the situation is not much more difficult. Consider $\mathcal{A}_i$, and once again renumber the vertices so that $i = 1$. Let $e_1, e_2, \ldots, e_t$ be the sets in the given hypergraph that contain the vertex i, and define $e_{\ell,j} = e_\ell \cap \{1, 2, \ldots, j\}$. To maintain the invariant **(P)**, it suffices to know which of the following three disjoint cases holds for each e_ℓ, after reading the values of $X_1, X_2, \ldots, X_j$:
(i) $e_{\ell,j} = e_\ell$, and $X_u \leq X_i$ for all $u \in e_{\ell,j}$;
(ii) $e_{\ell,j} \neq e_\ell$, and $X_u \leq X_i$ for all $u \in e_{\ell,j}$;
(iii) $X_u > X_i$ for some $u \in e_{\ell,j}$.
Note that $\mathcal{A}_i$ should enter state reject$_{i,j}$ iff case (i) holds for some ℓ. Otherwise, case (ii) or case (iii) holds for each ℓ. Thus, in addition to state reject$_{i,j}$, we only need $2^t \leq 2^{c \cdot \lg(n+m)} = (n+m)^c$ states to remember which case holds for each ℓ. Given these semantics for the states, it is also easy to see how the state-transitions should occur. Thus, we get a polynomial-size bound for S_i for the family of hypergraphs (H2) also.

Constant-Degree Hypergraphs. When the maximum degree of any vertex is bounded by some constant $d > 1$, we propose an alternative approach for finding a random permutation of the vertices. The resulting algorithm yields an IS of weight at least $(1-\epsilon)$ times the expected value presented in Theorem 2.2; ϵ here denotes an arbitrary positive constant.

Our approach uses algorithm $\mathcal{A}$, as given in Figure 2.1. Recall that B_v is the event that vertex v is in the final IS produced by $\mathcal{A}$. We rewrite $\text{Prob}(B_v)$ as follows. Suppose that the vertex v lies in edges $e_{v,1}, e_{v,2}, \ldots, e_{v,d(v)}$, where $d(v) \leq d$; as mentioned in Remark 3, we assume that all of these edges have size at least 2. Let $[k]$ denote the set $\{1, 2, \ldots, k\}$. Denote by $C_{v,u}$ the event "$X_v \geq X_u$"; then, by inclusion-exclusion,

$$\text{Prob}(B_v) = 1 + \sum_{S \subseteq [d(v)],\ |S| \geq 1} (-1)^{|S|} \cdot \text{Prob}\Big(\bigwedge_{i \in S} [\forall u \in e_{v,i},\ C_{v,u}]\Big)$$

(3.2)

$$= 1 + \sum_{S \subseteq [d(v)],\ |S| \geq 1} (-1)^{|S|} \cdot \text{Prob}\Big(\forall u \in \Big[\Big(\bigcup_{i \in S} e_{v,i}\Big) \backslash \{v\}\Big],\ C_{v,u}\Big)$$

(3.3)

$$= 1 + \sum_{S \subseteq [d(v)],\ |S| \geq 1} (-1)^{|S|} \cdot \Big(1 + \Big|\Big(\bigcup_{i \in S} e_{v,i}\Big) - \{v\}\Big|\Big)^{-1}; \tag{3.4}$$

(3.4) follows from the fact that each X_u, $u \in \bigcup_{i\in S} e_{v,i}$, is equally likely to be $\max\{X_w :\ w \in \bigcup_{i\in S} e_{v,i}\}$.

Denote by S_n the set of all permutations of $[n]$. A family of permutations $\mathcal{F} \subseteq S_n$ is defined in [4] to be *approximately min-wise independent with relative error* $\delta > 0$ if, for all $X \subseteq [n]$ and $x \in X$, we have for a randomly chosen permutation $\pi \in \mathcal{F}$ that

$$|\mathrm{Prob}(\min\{\pi(X)\} = \pi(x)) - \frac{1}{|X|}| \leq \frac{\delta}{|X|}.$$

We abbreviate the above property by (n,δ)-amw. We will use the property that for any n and $\delta > 0$, there is an explicitly constructible permutation family $F(n,\delta)$ that satisfies property (n,δ)-amw, and has cardinality $n^{O(\lg(1/\delta))}$ [11]. Note that given such a family $F(n,\delta)$, we can generate a family of permutations of the same size[3] that satisfies

$$\frac{1-\delta}{|X|} \leq \mathrm{Prob}(\max\{\pi(X)\} = \pi(x)) \leq \frac{1+\delta}{|X|}. \tag{3.5}$$

Thus, w.l.o.g, when referring to $F(n,\delta)$, we assume that F satisfies (3.5). We show below that for an appropriate constant $\delta = \delta(\epsilon, d)$, the expected size of an IS produced by using a random element of $F(n,\delta)$ is at least $(1-\epsilon)$ times the value guaranteed by Theorem 2.2. Thus, it suffices to choose a random permutation from the explicit polynomial-sized set $F(n,\delta)$. We can then apply parallel exhaustive search on the polynomial-sized $F(n,\delta)$ to find a "good" permutation in NC.

THEOREM 3.1. *Consider the family of hypergraphs with maximum degree at most d, for any given constant $d > 0$. Then, given any constant $\epsilon > 0$, there is an NC algorithm to find in these hypergraphs an IS of total weight at least $(1-\epsilon)$ times the expected weight guaranteed by Theorem 2.2.*

Proof. Let $\delta > 0$ be a constant (to be determined). We denote by $\mathrm{Prob}(B_v|F(n,\delta))$ the probability that vertex v is in the final IS produced by $\mathcal{A}$, conditioned on the random choice of π from $F(n,\delta)$; then, it is sufficient to show that for any $v \in V$, $\mathrm{Prob}(B_v|F(n,\delta))$ is at least $(1-\epsilon)$ times the value guaranteed by Lemma 2.1. The statement of the theorem will then follow from the linearity of expectation. Denote by V_S the set of vertices that lie in the subset of edges $e_{v,i_1}, \ldots, e_{v,i_{|S|}}$. Then, by inclusion-exclusion,

$$\mathrm{Prob}(B_v|F(n,\delta)) = \sum_{S\subseteq[d(v)]} (-1)^{|S|}\cdot \mathrm{Prob}(v \text{ is last in } \pi \text{ among the vertices in } V_S \mid F(n,\delta)).$$

Using (3.4) and (3.5), we get that

$$\begin{aligned}
\mathrm{Prob}(B_v|F(n,\delta)) &\geq 1 + \sum_{S\subseteq[d(v)]:\ |S|\geq 1} (-1)^{|S|}\cdot (1 + |(\bigcup_{i\in S} e_{v,i}) - \{v\}|)^{-1} \\
&\quad -\delta \sum_{S\subseteq[d(v)]:\ |S|\geq 1} (1 + |(\bigcup_{i\in S} e_{v,i}) - \{v\}|)^{-1} \\
&\geq \mathrm{Prob}(B_v) - \delta \sum_{S\subseteq[d(v)]:\ |S|\geq 1} (1/2) \\
&= \mathrm{Prob}(B_v) - \delta(2^{d-1} - \frac{1}{2}).
\end{aligned}$$

[3]This is done by taking, for any permutation π chosen from $\mathcal{F}$, the reverse permutation π', that is, $\pi'(i) = n + 1 - \pi(i)$, $\forall i$.

(The second inequality follows from our assumption that all the edges $e_{v,i}$ have size at least 2.) It follows from Lemma 2.1 that $\text{Prob}(B_v) \geq \int_0^1 (1-x)^{d(v)}\, dx = \frac{1}{d(v)+1} \geq \frac{1}{d+1}$. Hence, taking $\delta = \frac{2\epsilon}{(d+1)(2^d-1)}$ and summing over all $v \in V$ we get the statement of the theorem.

■

4. Concluding Remarks. A question that remains open is to obtain a full derandomization of our RNC algorithms. Any progress on the classical MIS problem on hypergraphs would also be most interesting.

Acknowledgments. We thank Noga Alon for valuable discussions, and for pointing out to us the results of Caro & Tuza. Thanks to Andrei Broder and Mike Mitzenmacher for fruitful discussions. We also thank the SPAA 2001 program committee member(s) and referee(s), as well as the journal referee, for their helpful comments.

REFERENCES

[1] N. Alon, L. Babai and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. of Algorithms*, 7, pp. 567–583, 1986.

[2] N. Alon and J. H. Spencer. The Probabilistic Method, Second Edition. Wiley-Interscience, 2000.

[3] P. Beame and M. Luby. Parallel search for maximal independence given minimal dependence. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 212–218, 1990.

[4] A. Z. Broder, M. Charikar, A. M. Frieze and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60 (2000), 630–659.

[5] Y. Caro and Z. Tuza. Improved lower bounds on k-independence. *J. Graph Theory*, 15, pp. 99–107, 1991.

[6] E. Dahlhaus, M. Karpinski, and P. Kelsen. An efficient parallel algorithm for computing a maximal independent set in a hypergraph of dimension 3. *Information Processing Letters*, 42(6):309–314, 1992.

[7] C. M. Fortuin, J. Ginibre, P. N. Kasteleyn, Correlational Inequalities for Partially Ordered Sets, *Communications of Mathematical Physics*, 22, pp. 89–103, 1971.

[8] M. Goldberg and T. Spencer. A new parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 18:419–427, 1989.

[9] M. Goldberg and T. Spencer. An efficient parallel algorithm that finds independent sets of guaranteed size. *SIAM J. Disc. Math.*, 6:443–459, 1993.

[10] M. Hofri. Analysis of Algorithms. Oxford University Press, 1995.

[11] P. Indyk. A small approximately min-wise independent family of hash functions. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 454–456, 1999.

[12] D. R. Karger and D. Koller. (De)randomized construction of small sample spaces in NC. *Journal of Computer and System Sciences*, 55 (1997), pp. 402–413.

[13] R. M. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In *Handbook of Theoretical Computer Science, Volume A*, J. van Leeuwen, Editor, Elsevier, New York, pages 871–941, 1990.

[14] R. M. Karp, E. Upfal, and A. Wigderson. The complexity of parallel search. *Journal of Computer and System Sciences*, 36:225–253, 1988.

[15] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. *J. Assoc. Comput. Mach.*, 32:762–773, 1985.

[16] P. Kelsen. On the parallel complexity of computing a maximal independent set in a hypergraph. In *Proc. ACM Symposium on Theory of Computing*, pages 339–350, 1992.

[17] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15 (1986), pp. 1036–1053.

[18] T. Łuczak and E. Szymańska. A Parallel Randomized Algorithm for Finding a Maximal Independent Set in a Linear Hypergraph. *J. Algorithms*, 25:311–320, 1997.

[19] R. J. McEliece and K. N. Sivarajan. Performance limits for channelized cellular telephone systems. *IEEE Trans. Info. Theory*, 40(1):21–34, 1994.

[20] S. Mahajan, E. A. Ramos, and K. V. Subrahmanyam. Solving some discrepancy problems in NC. *Algorithmica*, 29:371–395, 2001.

[21] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge Univ. Press, 1995.

[22] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12:449–461, 1992.

[23] N. Nisan. $RL \subseteq SC$. In *Proc. ACM Symposium on Theory of Computing*, pages 619–623, 1992.

[24] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*, Addison Wesley, 1996.

[25] D. Sivakumar. Algorithmic derandomization via complexity theory. In *Proc. ACM Symposium on Theory of Computing*, pages 619–626, 2002.

[26] J. H. Spencer. The probabilistic lens: Sperner, Turán and Brégman revisited. In *A Tribute to Paul Erdős* (A. Baker, B. Bollobás, A. Hajnal, Eds.), Cambridge Univ. Press, pp. 391–396, 1990.

[27] E. Szymańska. Derandomization of a Parallel MIS Algorithm in a Linear Hypergraph. In *Proc. Fourth International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 39–52, 2000.

[28] P. Turán. On the theory of graphs. *Colloq. Math.*, 3, pp. 19–30, 1954.