A parallel algorithmic version of the Local Lemma.

Noga Alon *
Department of Mathematics
Raymond and Beverly Sackler Faculty of Exact Sciences
Tel Aviv University, Tel Aviv, Israel

Abstract

The Lovász Local Lemma is a tool that enables one to show that certain events hold with positive, though very small probability. It often yields existence proofs of results without supplying any efficient way of solving the corresponding algorithmic problems. J. Beck has recently found a method for converting some of these existence proofs into efficient algorithmic procedures, at the cost of loosing a little in the estimates. His method does not seem to be parallelizable. Here we modify his technique and achieve an algorithmic version that can be parallelized, thus obtaining deterministic NC^1 algorithms for several interesting algorithmic problems.

*Research supported in part by a United States Israel BSF Grant and by a Bergmann Memorial Grant

1 Introduction

In a typical application of the probabilistic method we try to prove the existence of a combinatorial structure (or a substructure of a given structure) with certain prescribed properties. To do so, we show that a randomly chosen element from an appropriately defined sample space satisfies all the required properties with positive probability. In most applications, this probability is not only positive, but is actually high and frequently tends to 1 as the parameters of the problem tend to infinity. In such cases, the proof usually supplies an efficient randomized algorithm for producing a structure of the desired type, and in many cases this algorithm can be derandomized and converted into an efficient deterministic one. By efficient we mean here, as usual, an algorithm whose running time -(or expected running time, in case we consider randomized algorithms)- is polynomial in the length of the input.

There are, however, certain examples, where one can prove the existence of the required combinatorial structure by probabilistic arguments that deal with rare events; events that hold with positive probability which is exponentially small in the size of the input. Such proofs usually yield neither randomized nor deterministic efficient procedures for the corresponding algorithmic problems.

A class of examples demonstrating this phenomenon is the class of results proved by applying the Lovász Local Lemma. The exact statement of this lemma (for the symmetric case) is the following.

Lemma 1.1 *Let $A_1, \ldots, A_n$ be events in an arbitrary probability space. Suppose that the probability of each of the n events is at most p, and suppose that each event A_i is mutually independent of all but at most b of the other events A_j. If $ep(b+1) < 1$ then with positive probability none of the events A_i holds.*

Many applications of this lemma can be found in [14], [2], [4], [8], [7], [10], [11], [9]. For several years there has been no known method of converting the proofs of any of these examples into an efficient algorithm. Very recently J. Beck [12] found such a method that works for many of these examples with a little loss in the constants. Beck demonstrated his method by considering the problem of hypergraph 2-coloring. A hypergraph $H = (V, E)$ is 2-*colorable* if there is a two coloring of V so that no edge in E is monochromatic. The following result is due to Erdös and Lovász. Its

derivation from the Local Lemma is very simple [14].

Theorem 1.2 *If $e(d+1) < 2^{n-1}$ then any hypergraph H in which every edge has at least $n \geq 2$ vertices and no edge intersects more than d other edges is 2-colorable.*

Recall that a hypergraph is *n-uniform* if each of its edges contains precisely n vertices. Observe that there is no loss of generality in assuming that H is n-uniform in Theorem 1.2, since otherwise we can simply replace each edge of H by a subset of size n of it. Let H be an n-uniform hypergraph with N edges satisfying the assumptions in Theorem 1.2, and suppose, for simplicity, that n, d are fixed. Can we find a proper two coloring of H (i.e., a vertex coloring in which no edge is monochromatic) efficiently? Beck showed that indeed we can, in case d is somewhat smaller, say $d < O(2^{n/1000})$. In this case there is a randomized as well as a deterministic algorithm whose running time is polynomial in N for finding a proper two coloring. Beck's method does not seem to provide a parallel efficient algorithm (i.e., an algorithm that runs in poly-logarithmic time using a polynomial number of processors). Here we modify his algorithm and obtain a version which is parallelizable. We first describe the randomized version of the algorithm and then comment briefly on the possibilities to derandomize and parallelize it. Let us denote, as usual, the binary entropy function by $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$.

Theorem 1.3 *Suppose $n \geq 2, d$ are fixed and suppose that for some $\alpha > 0$*

$$4ed^3 < 2^{n(1-H(\alpha))} \tag{1}$$

and

$$2e(d+1) < 2^{\alpha n}. \tag{2}$$

Then there is a randomized algorithm that finds a proper 2-coloring of any given n-uniform hypergraph H with N edges in which no edge intersects more than d others in expected running time $N^{O(1)}$. This algorithm can be derandomized and parallelized, providing a deterministic algorithm that finds a proper coloring in time $O(\log N)$ using $N^{O(1)}$ processors.

We note that for large n, any $d \leq 2^{n/8}$ satisfies the above (by taking an appropriate $\alpha > 1/8$). We also note that by assuming that d is smaller, say that $d < 2^{n/500}$, the expected running time can be reduced to almost linear in N. In addition, if indeed $d < 2^{n/500}$ the assumption that n, d are fixed can be omitted.

2 The basic algorithm

Let H be as in Theorem 1.3. Here is the randomized algorithm for finding a proper 2-coloring of H. In the First Pass we color all the vertices of H, randomly and independently by two colors, where each point is colored either red or blue with equal probability. Call an edge *bad* if at most αn of its points are red or at most αn of its points are blue. The probability of a fixed edge to be bad is clearly at most $2\sum_{i\leq \alpha n}\binom{n}{i}/2^n \leq 2\cdot 2^{(H(\alpha)-1)n}$. Put $p = 2\cdot 2^{(H(\alpha)-1)n}$ and let B denote the set of all bad edges.

Let G be the dependency graph for the problem, i.e., the graph whose vertices are the edges of H in which two are adjacent iff they intersect. Observe that if S is an independent set in G then the probability that $S \subset B$ is at most $p^{|S|}$, since these $|S|$ events are mutually independent. Let us call a set of vertices C of G a $1,2$-tree if C is the set of vertices of a connected subgraph in the square of G. I.e., C is a $1,2$-tree if the $A_i \in C$ are such that drawing an arc between $A_i, A_j \in C$ if their distance in G is either 1 or 2 the resulting graph is connected.

Lemma 2.1 *The probability that every $1,2$-tree in G all of whose vertices belong to B has size at most $d\log(2N)$ is at least $1/2$.*

Proof Call $T \subseteq G$ a $2,3$-tree if the $A_i \in T$ are such that all their mutual distances in G are at least two and so that, drawing an arc between $A_i, A_j \in T$ if their distance in G is either 2 or 3 the resulting graph is connected. We first bound the number of $2,3$-trees of size u in G. Consider the graph on the set of vertices of G in which two vertices are adjacent if their distance in G is either 2 or 3. Every $2,3$-tree on a set T of vertices of G must contain a tree on T in this new graph. The new graph has maximum degree smaller than $D = d^3$. It is well known (see [17]) that an infinite D-regular rooted tree contains precisely $\frac{1}{(D-1)u+1}\binom{Du}{u}$ rooted subtrees of size u, and this easily implies that the number of trees of size u containing one specific given vertex in any graph with maximum degree at most D does not exceed this number, which is smaller than $(eD)^u$.

For any particular $2,3$-tree T of size u we know that $\Pr[T \subseteq B] \leq p^u$. Hence the expected number of $2,3$-trees $T \subseteq B$ of size u is at most $N(eDp)^u$. As $eDp < 0.5$ by (1), if $u = \log(2N)$ this term is at most $1/2$. Thus with probability at least $1/2$ there is no $2,3$-tree of size bigger than $\log(2N)$ all of whose vertices are in B. We actually want to bound the size of any $1,2$-tree C of G

all of whose vertices lie in B. A maximal $2,3$-tree T in such a C must have the property that every $A_i \in C$ is a neighbor (in G) of an $A_j \in T$. (This is because otherwise there is an $A_l \in C$ which is not a neighbor of any set in T and yet it is of distance 2 or 3 from some $A_j \in T$. Such an A_l can be added to T, contradicting its maximality). There are less than d neighbors A_i of any given A_j so that $\log(2N) \geq |T| \geq |C|/d$ and so

$$|C| \leq d \log(2N)$$

completing the proof of the lemma. □

Let us call the First Pass *successful* if there is no $1,2$-tree of size greater than $d \log(2N)$ all of whose vertices lie in B. By the last lemma the probability the First Pass is successful is at least $1/2$. In case it is not, we simply repeat the entire procedure. In expected *linear* time the First Pass is successful.

We can now fix the coloring by recoloring, in the Second Pass, the vertices of H that belong to the bad edges. Let us call an edge *dangerous* if it contains at least αn vertices that belong to bad edges. (Thus, in particular, bad edges are also dangerous). Observe that if an edge is not dangerous then it will not become monochromatic after the recoloring. This is because less than αn of its points will change color, and it has at least αn points of each color before the recoloring. Thus we only have to worry about the dangerous edges. However, if we recolor all the vertices in bad edges randomly and independently than we recolor at least αn vertices in each dangerous edge, and hence the probability it becomes monochromatic does not exceed $2^{-\alpha n}$. Since each dangerous edge intersects at most d others it follows from (2) and from Theorem 1.2 (or directly from the Lovász Local Lemma) that there exists a recoloring in which no edge is monochromatic.

The crucial point is that the recoloring of the points in the edges of each maximal $1,2$-tree C of bad edges can be done separately. This is because there is no dangerous edge that intersects edges from two distinct such maximal $1,2$-trees, and hence it suffices to recolor the points in the edges of each such C in a way that only makes sure that no dangerous edge intersecting an edge in C becomes monochromatic. Since each of the $1,2$-trees C as above has only $O(\log N)$ vertices that have to be recolored, we can find the required recoloring by *exhaustive enumeration!* Examining all possible two-colorings in each such C only takes time $O(2^{O(\log N)}) = N^{O(1)}$ and hence doing it for all the above C-s can be done in polynomial time.

This completes the description of the randomized algorithm with expected polynomial running time. In case $d < 2^{cn}$ for a smaller c we can make another pass similar to the first one in each $1,2$-tree separately, get new $1,2$-trees of size $O(\log logN)$ and complete as before obtaining an expected running time which is nearly linear- $O(N(\log N)^{O(1)})$. We omit the detailed computation.

The randomized algorithm above is trivially parallelizable and can be implemented on a standard $EREW$-PRAM in time $O(\log N)$ using $N^{O(1)}$ parallel processors. (See [16] for the basic definitions of an $EREW$-PRAM and the complexity classes NC and NC^1.) Moreover, the algorithm can be derandomized maintaining the running time (with some increase in the number of processors), showing that the problem can be solved in NC^1. To see this observe that the recoloring step is deterministic even in the version described above, so the only problem is the derandomization of the First Pass. This can be done by applying the techniques from [19] or [6]. (It is also possible to apply the methods of [18] and [13], but this will not supply NC^1-algorithms, although it would yield algorithms with a smaller number of processors). The basic idea is that for every constant c there is a constant $b = b(c)$ such that for every m there are explicit sample spaces of size at most m^b in which one can embed m random variables taking the values $0,1$ in which every set of $c \log m$ of the variables is nearly independent. The details of these constructions appear in the above mentioned papers. Let us only mention here that the First Pass described above can be performed deterministically as follows.

Let $q \geq N^b$ be a prime, where b is a constant depending only on n. Let χ be the quadratic character defined on the elements of the finite field $GF(q)$, i.e., $\chi(x) = 1$ if x is a quadratic residue modulo q and $\chi(x) = -1$ otherwise. Define a family of q two-colorings of the set of vertices of H as follows. For each $i \in GF(q)$, the color of the j-th point in the i-th coloring is blue if $\chi(i-j) = 1$ and is red if $\chi(i-j) = -1$. Using the results in [6] (based on the ideas in [19] (see also [3])), it can be shown that at least one of the q colorings defined above will produce a successful First Pass. All these (deterministically defined) colorings can be checked in parallel, completing the proof of Theorem 1.3. $\Box$

It is worth noting that one can give a different variant of the first pass in the basic algorithm, which is also parallelizable, though it does not yield NC^1 algorithms. This variant is more similar to Beck's original algorithm, but since it supplies somewhat slower parallel algorithms we do not describe it here.

3 More on hypergraph coloring

The argument in the proof of Theorem 1.3 can be easily extended to the case where n and d are not fixed, provided d is somewhat smaller as a function of n, say, $d \leq 2^{n/500}$. Moreover, we can easily obtain a coloring in which each edge contains *many* points of each of the two colors. Here is a description of the randomized algorithm for this more general case. Its parallelization and derandomization are similar to these of the algorithm for the case of fixed n and d. Since the algorithms for fixed n, d have already been described we assume here, whenever it is needed, that d is sufficiently large.

For the First Pass, choose, say, $\alpha = 1/8$, and immitate the First Pass described in the previous section. That is, color all the points randomly and independently red or blue, call an edge bad if it contains at most $n/8$ red points or at most $n/8$ blue points, and call an edge dangerous if at least, say, $n/9$ of its points belong to bad edges. We will later recolor only the points in bad edges, and hence every non-dangerous edge will have at least $n/72$ points of each color. Since d is here smaller as a function of n than it is in the previous section, one can repeat the proof of Lemma 2.1 and show that the probability of getting a $2,3$-tree of size u in the dependency graph G, all of whose vertices correspond to bad edges of H is at most $2^{-(cnu)}$ for some absolute constant c. Moreover, for this we only need that each set of nu points receive their colors independently (or almost independently). Taking $u = c_1 \log N/n$ for an appropriately chosen positive constant c_1 we conclude that with probability at least $1/2$ we get no $2,3$-tree of size u whose vertices correspond to bad edges, and hence no $1,2$-tree of size $c_1 d \log N/n$ with such edges. Now we can consider each such maximal $1,2$-tree and the set of all dangerous edges intersecting it separately. We replace each edge here by a subset of $n/9$ of its points which are to be recolored and observe that it suffices to obtain a 2- coloring of each of these new hypergraphs so that each edge will have many points of each of the two colors. Since each bad edge intersects at most d dangerous edges, the number of edges in each new hypergraph is $O(d^2 \log N/n)$ and the number of vertices in it is $O(d^2 \log N)$.

We can now make another coloring pass in each of these hypergraphs (when we now color only the points in the edges which were bad in the first coloring) and repeat the same argument. An edge (which is now of size $n/9$) will be bad if it has at most $n/72$ red or at most $n/72$ blue points. An edge will be dangerous if it has at least $n/81$ points in bad edges. We thus get now $1,2$-trees of bad

edges of size at most $O(\log\log N + \log d)d/n$, and each of these together with the dangerous edges it intersects has at most $O(\log\log N + \log d) \cdot d^2$ edges and vertices. Moreover, in each dangerous edge, at least $n/81$ points will get new colors.

Now each such piece can be recolored separately. If $d > \log\log N$ then a random coloring will fix each piece with high probability. This is because the probability that an edge will have less than, say, $n/1000$ points of each of the two colors is at most

$$\frac{2 \cdot \sum_{i=0}^{n/1000} \binom{n/81}{i}}{2^{n/81}} < 1/d^4$$

and we have much less than d^4 edges in a piece. Otherwise, we can check all possible colorings of each piece separately in time $O(\log N)^{O(1)}$ per piece. Since the Local Lemma gives the existence of a coloring in which each edge has at least $n/1000$ points of each color we can find one and complete the coloring.

We note that in all the above we never have to use more than $O(\log N)$-wise independence of our randomly chosen colors. Hence, the above procedure can be derandomized and parallelized as before. We have thus proved:

Theorem 3.1 *For any $n \geq 2, d$ satisfying $d \leq 2^{n/500}$ there is an NC^1 algorithm whose input is a hypergraph $H = (V, E)$ in which each edge has at least n vertices and each edge intersects at most d other edges, and whose output is a vertex two-coloring of H such that each edge has at least $n/1000$ points of each color. The algorithm runs in time $O(\log N)$ on an EREW-PRAM with $N^{O(1)}$ parallel processors, where N is the number of edges of H.*

We note that the constants 500 and 1000 can be easily improved, and we make no attempt to optimize them here. It is well known that the Local Lemma implies that for every n-uniform hypergraph H in which each edge intersects at most d others there is a two coloring of the vertices of H such that the number of vertices of each color in each edge is at least $\frac{n}{2} - O(\sqrt{n})\sqrt{\log d}$. For relatively small values of d this is a better estimate than the $n/1000$ given above, but for our purposes here the $n/1000$ estimate suffices.

In various applications it is required to color the vertices of a hypergraph by more than 2 colors so that each color will appear in each edge. The Local Lemma yields the existence of such a coloring in certain cases, and one can immitate the proof above to get a parallel algorithmic version of this result. It seems simpler to derive this result here from Theorem 3.1.

Corollary 3.2 *For every fixed k there is an $\epsilon = \epsilon(k) > 0$ such that the following holds. There is an NC^1 algorithm whose input is a hypergraph $H = (V, E)$ in which each edge has at least $n \geq 2$ vertices and each edge intersects at most d others, where $d \leq 2^{\epsilon n}$, and whose output is a k-coloring of the vertices of H so that each color occures in each edge. The algorithm runs in time $O(\log N)$ on an EREW-PRAM with $N^{O(1)}$ parallel processors, where N is the number of edges of H.*

Proof We first apply Theorem 3.1 and obtain a 2-coloring of H as in the theorem. For $i \in \{1, 2\}$ let H_i be the hypergraph whose set of vertices V_i is the set of all vertices of H colored i and whose edges are all the intersections of the edges of H with V_i. Since each edge of H_i has at least $n/1000$ vertices and intersects at most d other edges of H_i, we can 2-color each H_i using Theorem 3.1 again. After a constant number of steps of this type we obtain the desired result. □

4 Additional algorithmic applications

In this section we describe several additional algorithmic problems for which the technique above yields efficient parallel algorithms. Whenever we can we prefer to apply the results on hypergraph coloring proved in the previous sections instead of applying similar arguments directly, although this costs in a little loss in the estimates.

4.1 Even cycles in directed graphs

The problem of deciding whether a given directed graph contains a (simple) even (directed) cycle arises in various contexts, including the study of the problem of sign-solvability of a system of linear equations, (see [21]). There is no known polynomial time algorithm for answering this decision problem, but it is also not known to be NP-complete. (It is known that the similar decison problem of deciding whether a given directed graph contains an even cycle through a given edge is NP-complete). There are several results that supply a sufficient condition for a digraph to contain such a cycle. Friedland ([15]) showed that for every $r \geq 7$, any r-regular digraph contains an even cycle. His proof supplies no polynomial time agorithm for finding an even cycle in such a digraph. In [9] it is shown that one can assume much less than exact regularity. In fact, any digraph in which the maximum indegree is not much bigger than the minimum outdegree contains an even cycle. The proof applies the Local Lemma. The results here enable us to convert this proof into

an efficient NC^1 algorithm, as shown in the next theorem. (Here and from now on we make no attempt to optimize the constants.)

Theorem 4.1 *There is an NC^1 algorithm whose input is a digraph $D = (V, E)$ in which each outdegree is at least $\delta > 0$ and each indegree is at most Δ, where*

$$\Delta \leq \frac{2^{(\delta+1)/500}}{\delta+1},$$

and whose output is an even directed cycle in D. The algorithm runs in time $O(\log N)$ on an EREW-PRAM with $N^{O(1)}$ parallel processors, where N is the number of edges of D.

Proof We may assume, by deleting edges if necessary, that every outdegree is precisely δ. For each vertex v of D, let $N^+(v)$ denote the set of all vertices u of D such that there is a directed edge from v to u. Let H be the hypergraph whose vertex set $V(H)$ is the set V of all vertices of D, and whose edges are all the sets $v \cup N^+(v)$, where $v \in V$. Every edge of H has precisely $\delta + 1$ vertices. We claim that every edge intersects at most $2^{(\delta+1)/500}$ other edges. To see this observe that for any vertex v of D and for any $w \in N^+(v)$ there are at most $\Delta - 1$ vertices u other then v such that $w \in N^+(u)$. There are at most Δ vertices u such that $v \in N^+(u)$ and there are at most δ additional vertices u with $u \in N^+(v)$. Altogether, there are at most $(\Delta - 1)\delta + \Delta + \delta = \Delta(\delta + 1)$ vertices u with $(u \cup N^+(u)) \cap (v \cup N^+(v)) \neq \emptyset$, and this number is, by assumption, at most $2^{(\delta+1)/500}$, as claimed.

It thus follows from Theorem 3.1 that we can find a proper two coloring $f : V \mapsto \{0, 1\}$ of H in time $O(\log N)$ using $N^{O(1)}$ parallel processors. For each vertex $v \in V$, let $u = u(v)$ be a vertex so that $f(u) \neq f(v)$ and (v, u) is a directed edge of D. (Such a u exists and can be found quickly for all v since f is not a constant on $v \cup N^+(v)$). Let G denote the subgraph of D consisting of all the (directed) edges $(v, u(v))$. Observe that each directed cycle in G is even, since the values of f on the vertices along this cycle alternate. Moreover, every outdegree in G is precisely 1. It is easy to see that one can find a directed cycle in G in time $O(\log N)$ using $N^{O(1)}$ processors, completing the proof. □

The assertion of the last theorem can be generalized by applying Corollary 3.2 rather then Theorem 3.1 for deriving an algorithmic version of the general result in [9], dealing with cycles of length 0 modulo k. This gives the following result, whose detailed proof (which is analogous to that of the previous theorem) is omitted.

Theorem 4.2 *For every fixed k there is an $\epsilon = \epsilon(k) > 0$ such that the following holds. There is an NC^1 algorithm whose input is a digraph $D = (V, E)$ in which each outdegree is at least $\delta > 0$ and each indegree is at most Δ, where*

$$\Delta \leq 2^{\epsilon\delta},$$

and whose output is a (simple) directed cycle in D whose length is divisible by k. The algorithm runs in time $O(\log N)$ on an EREW-PRAM with $N^{O(1)}$ parallel processors, where N is the number of edges of D.

4.2 Star arboricity

A *star forest* is a forest whose connected components are stars. The *star arboricity* of a graph G, denoted by $st(G)$, is the minimum number of star forests whose union covers all edges of G. For an integer $\Delta \geq 1$ define $st(\Delta) = Max\{st(G)\}$, where the maximum is taken over all simple graphs with maximum degree Δ.

The star arboricity of graphs was introduced by Akiyama and Kano [1] and has been studied in various papers. The asymptotic behaviour of $st(\Delta)$ is determined in [11], improving a previous estimate from [2]. In particular it is known that for every $\gamma > 0$ there is a $\Delta = \Delta_0(\gamma)$ such that for every $\Delta \geq \Delta_0$, $st(\Delta) \leq (\frac{1}{2} + \gamma)\Delta$.

Given a small fixed $\gamma > 0$ and a graph G with maximum degree Δ, where $\Delta > \Delta_0(\gamma)$ is fixed, can we find efficiently a family of at most $(\frac{1}{2} + \gamma)\Delta$ star-forests that cover all edges of G? As shown in the next subsection this question is naturally suggested by the study of a certain communication model. The next result shows that the desired star-forests can be found efficiently and in parallel.

Theorem 4.3 *For every fixed $\gamma > 0$ there is a $\Delta_0 = \Delta_0(\gamma)$ such that the following holds. There exists an algorithm whose input is a (simple) graph G with a fixed maximum degree $\Delta > \Delta_0$, and whose output is a family of at most $(\frac{1}{2} + \gamma)\Delta$ star-forests whose union covers all edges of G. If N is the number of vertices of the graph, then the algorithm runs on an EREW-PRAM in time $O(\log N)$ using $N^{O(1)}$ processors.*

Proof We start with some technical details. We may assume that our graph G is Δ-regular, and that Δ is even, since we can always complete G to a $2\lceil\Delta/2\rceil$- regular graph, by adding (a polynomial number of) vertices, as well as edges, if necessary. By orienting G along an Euler cycle in each

of its connected components we obtain a digraph $D = (V, E)$ in which every outdegree and every indegree is precisely $\Delta/2$. It is well known that all this can be done in NC^1.

Let H be the hypergraph whose set of vertices is V and whose edges are all the sets of the form $v \cup N^-(v)$, where $N^-(v) = \{u \in V : (u, v) \in E\}$. Observe that H is $\Delta/2$-uniform, and each edge of H intersects at most $O(\Delta^2)$ others. Let k be a fixed integer satisfying $1/(k-1) < \gamma$. By Coroallry 3.2, if Δ is sufficiently large we can k-color the vertices of H so that each color will appear in each edge. This gives a coloring of the set of vertices of D in which for each vertex v and for each color $1 \leq j \leq k$, there is a vertex u in that color such that (u, v) is an edge of D. We now define k star-forests $S_1, \ldots, S_k$ in D whose union is a spanning subgraph of D in which each indegree is precisely $k-1$. To do so, we let the centers of the stars of S_j be all the vertices colored j. For each vertex v whose color is not j, we choose a vertex u whose color is j such that $(u, v) \in E$ and let the edge (u, v) belong to S_j. Deleting these resulting k star-forests from G we obtain a graph in which every indegree is $\Delta/2-(k-1)$, and in which each outdegree is (of course) at most $\Delta/2$. This process can be continued as long as the remaining indegrees are sufficiently large to apply Corollary 3.2. If we start with a sufficiently large Δ this can be repeated until every indegree is, say, $\sqrt{\Delta}$. It is easy to see that the set of all remaining edges can be covered by, e.g., $2\sqrt{\Delta}+2$ star-forests (since they can be covered by $\sqrt{\Delta}+1$ forests, by the well known theorem of Nash-Williams on the arboricity of graphs, and each forest can be covered by two star-forests). Therefore, if we start with a sufficiently large Δ so that $2\sqrt{\Delta}+2 < \frac{1}{2}\gamma\Delta$ we can bound the total number of star-forests by $\frac{k}{k-1}\Delta/2 + 2\sqrt{\Delta}+2 < (\frac{1}{2}+\frac{1}{2k-2})\Delta + \frac{1}{2}\gamma\Delta < (\frac{1}{2}+\gamma)\Delta$, completing the proof. □

4.3 Radio networks

The study of star arboricity is naturally suggested by the analysis of certain communication networks. A *radio network* is a synchronous network of processors that communicate by transmitting messages to their neighbors. A processor P can receive at most one message in one step. Let us mention here two possible models.

Type I : P receives a message from its neighbor Q in a given step if P is silent, Q transmits and P chooses to receive from Q in this step.

Type II: P receives a message from its neighbor Q if P is silent, and Q is the *only* neighbor of P that transmits in this step.

Suppose, now, that the model is the Type I model and the network is represented by an undirected graph $G = (V, E)$ whose vertices are the processors and two are adjacent if they can transmit to each other. Suppose, further, that we need to transmit once along every edge (in one of the two possible directions), say, in order to check that there is indeed a connection between each adjacent pair. It is easy to see that the minimum number of steps in which we can finish all the required transmissions is precisely $st(G)$, since the set of edges corresponding to the transmissions performed in a single step forms a star forest. Theorem 4.3 thus supplies an upper bound for the minimum number of required steps. Observe that if G is a Δ-regular graph on N vertices then clearly at least $\Delta/2$ steps are necessary, simply because each star-forest has at most $N - 1$ edges. Therefore, the $(\frac{1}{2} + \gamma)\Delta$ estimate is almost tight in many cases, and the required communication pattern can be found in this case efficiently in parallel.

What about the Type II networks? This model is much more popular, and has been considered in many papers (see, e.g., [4] and its many references). A basic parameter of a network represented by a directed graph D in this model is its *hitting number* $h(D)$ defined as follows. Given a digraph $D = (V, E)$ and a subset A of V, we say that A *hits* a directed edge $(x, y) \in E$ if $x \in A$, $y \notin A$ and the only vertex in $N^-(y) \cap A$ is x. A family of subsets of V *hits* D if for every directed edge e of D there is a set in the family that hits e. Let $h(D)$ denote the minimum cardinality of a family that hits D. One can easily see that this is precisely the minimum number of steps in which it is possible to complete a transmission along every edge of D. This parameter is of central interest in the study of the task referred to as a Single Round Simulation in [5].

For simplicity let us restrict our attention to symmetric graphs, i.e., (x, y) is an edge of D iff (y, x) is an edge. In [4], [5] it is shown that there exist two positive constants c_1 and c_2 such that
(i) For every (symmetric) digraph D with maximum degree Δ, $h(G) \leq c_1 \Delta \log \Delta$.
(ii) For every Δ there is a symmetric digraph with maximum degree Δ such that $h(D) > c_2 \Delta \log \Delta$.

The proof of the upper bound stated in (i) applies the Local Lemma. Can we find, given a symmetric digraph D with maximum degree Δ, a family of cardinality $O(\Delta \log \Delta)$ efficiently and in parallel? This problem is suggested naturally by the discussion in [4], [5]. The technique described here can be used to answer this problem.

Theorem 4.4 *There exists a constant $c > 0$ such that for every fixed Δ there exists an algorithm whose input is a symmetric digraph $D = (V, E)$ with maximum degree Δ, and whose output is a*

family F of subsets of V that hits D, where $|F| \leq c\Delta \log \Delta$. Such a family can be computed in time $O(\log N)$ on an EREW-PRAM with $N^{O(1)}$ processors, where N is the number of vertices of D.

The proof in this case does not follow from the results on hypergraph coloring, but it resembles the proofs of these results. Let $D = (V, E)$ be a (symmetric) digraph with maximum degree Δ. Described briefly, the algorithm (in its randomized version) first chooses some $s = c_1 \Delta \log \Delta$ subsets $A_1, \ldots, A_s$ of V as the first members of the family of sets F, by choosing each element $v \in V$, randomly and independently, to be a member of A_i with probability $1/\Delta$. It can be shown that with high probability these sets suffice to hit almost all edges of D. The non-hit edges can be partitioned into classes, so that the distance in D between each two classes is at least 3, a fact that enables us to fix each class separately. Moreover, with high probability each such class contains only $O(\log N)$ edges. The Local Lemma can be used to show that there is an additional set of $O(\Delta \log \Delta)$ sets that hits all these remaining edges, and these sets can be found for each class separately by exhaustive enumeration. This algorithm can be derandomized and parallelized by the method described in Section 2. We omit the details.

4.4 Acyclic edge-coloring

We have seen several algorithmic problems that can be solved efficiently in parallel using the technique described in the first sections. In his original paper, Beck [12] describes several other results obtained by the Local Lemma, for which he obtains efficient (sequential) algorithms. There are several additional examples, including the main results in [7] and in [8] that can be converted into efficient algorithms by similar methods. It is not yet clear if this technique suffices for converting all the known applications of the Local Lemma into efficient algorithms. An example which looks difficult is finding an algorithmic version for the following theorem proved in [10]:

Theorem 4.5 *The edges of any graph with maximum degree Δ can be colored by 64Δ colors so that there are no two adjacent edges having the same color and there is no 2-colored cycle.*

Acknowledgement I would like to thank József Beck for sending me an early draft of his paper and Joel Spencer for helpful comments.

References

[1] J. Akiyama and M. Kano, *Path factors of a graph*, in: Graph Theory and its Applications (Wiley and Sons, New York, 1984).

[2] I. Algor and N. Alon, *The star arboricity of graphs*, Discrete Mathematics 75 (1989), 11-22.

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

[4] N. Alon, A. Bar-Noy, N. Linial and D. Peleg, *On the complexity of radio communication*, Proc. 21^{th} ACM Symp. on the Theory of Computing, Seattle, Washington, ACM Press (1989), 274-285.

[5] N. Alon, A. Bar-Noy, N. Linial and D. Peleg, *Single round simulation on radio networks*, to appear.

[6] N. Alon, O. Goldreich, J. Hastad and R. Peralta , Simple constructions of almost k-wise independent random variables, Proc. 31^{st} FOCS, St. Louis, Missouri, IEEE (1990), 544-553.

[7] N. Alon, *The linear arboricity of graphs*, Israel J. Math. 62(1988), 311-325.

[8] N. Alon, *The strong chromatic number of a graph*, to appear.

[9] N. Alon and N. Linial, *Cycles of length* 0 *modulo* k *in directed graphs*, J. Combinatorial Theory, Ser. B 47(1989), 114-119.

[10] N. Alon, C. McDiarmid and B. Reed, *Acyclic coloring of graphs*, to appear.

[11] N. Alon, C. McDiarmid and B. Reed, *Star Arboricity*, to appear.

[12] J. Beck, *An algorithmic approach to the Lovász Local Lemma*, to appear.

[13] B. Berger and J. Rompel, *Simulating* $log^c n$*-wise independence in NC*, Proc. 30^{th} IEEE FOCS (1989), 2-7.

[14] P. Erdös and L. Lovász, *Problems and results on* 3*-chromatic hypergraphs and some related questions*, in:"Infinite and Finite Sets" (A. Hajnal et. al. eds), Colloq. Math. Soc. J. Bolyai 11, North Holland, Amsterdam, 1975, pp. 609-627.

[15] S. Friedland, *Any 7-regular digraph contains an even cycle*, J. Combinatorial Theory Ser. B, 46 (1989), 249-252.

[16] R. M. Karp and V. Ramachandran, *A survey of parallel algorithms for shared memory machines*, in: Handbook of Theoretical Computer Science (J. Van Leeuwen Ed.), Chapter 17, MIT Press (1990), 871-941.

[17] D. Knuth, *The Art of Computer Programming*, Vol. I, Addison Wesley, London, 1969, p. 396 (Exercise 11).

[18] R. Motwani, J. Naor and M. Naor, *The probabilistic method yields deterministic parallel algorithms*, Proc. 30^{th} IEEE FOCS (1989), 8-13.

[19] J. Naor and M. Naor, *Small-bias probability spaces: efficient constructions and applications*, Proc. 22^{nd} annual ACM STOC, ACM Press (1990), 213-223.

[20] J. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.

[21] C. Thomassen, *Sign-nonsingular matrices and even cycles in directed graphs*, Linear Algebra Appl. 41 (1986), 27-42.