The Probabilistic Method Yields Deterministic Parallel Algorithms

Rajeev Motwani [*] Joseph (Seffi) Naor [†] Moni Naor [‡]

Abstract

We present a technique for converting RNC algorithms into NC algorithms. Our approach is based on a parallel implementation of the method of conditional probabilities. This method was used to convert probabilistic proofs of existence of combinatorial structures into polynomial time deterministic algorithms. It has the apparent drawback of being extremely sequential in nature. We show certain general conditions under which it is possible to use this technique for devising deterministic parallel algorithms.

We use our technique to devise an NC algorithm for the set balancing problem. This problem turns out to be a useful tool for parallel algorithms. Using our de-randomization method and the set balancing algorithm, we provide an NC algorithm for the lattice approximation problem. We also use the lattice approximation problem to bootstrap the set balancing algorithm, and the result is a more processor efficient algorithm. The set balancing algorithm also yields an NC algorithm for near-optimal edge coloring of simple graphs. Our methods also extend to the parallelization of various algorithms in computational geometry that rely upon the random sampling technique of Clarkson. Finally, our methods apply to constructing certain combinatorial structures, e.g., Ramsey graphs and independent sets and covers of hypergraphs.

Key words. de-randomization, parallel algorithms, probabilistic method, set balancing, lattice approximation, edge coloring, random sampling, computational geometry

AMS(MOS) subject classifications. 68C05. 68C25, 68E10

---

[*] Dept. of Computer Science, Stanford University, Stanford CA 94305, USA. Supported by NSF Grant CCR-9010517, NSF Young Investigator Award CCR-9357849, and grants from Mitsubishi Corporation and OTL.

[†] Department of Computer Science, Technion, Haifa 32000, Israel. Most of this work was done while the author was at the Department of Computer Science, Stanford University, and supported by contract ONR N00014-88-K-0166.

[‡] Department of Applied Math and Computer Science, Weizmann Institute, Rehovot 76100, Israel. Part of this work was done when the author was at the Computer Science Division, University of California, Berkeley, and supported by NSF grant CCR 88-13632. Part of this work was done when the author was with the IBM Almaden Research Center.

1 Introduction

A randomized algorithm $\mathcal{A}$ has a probability space (Ω, P) associated with it, where Ω is the sample space and P is some probability measure. We call a point $w \in \Omega$ a **good** point for some input instance I, if $\mathcal{A}(I, w)$ computes a correct solution. In most cases, it is guaranteed that, for each I, if w is chosen uniformly at random then the probability that w is good for I is sufficiently high. An important question in complexity theory is whether randomized algorithms have greater computational power than deterministic algorithms. For example, it is not known whether primality testing can be done in polynomial time without using randomization, or similarly, whether it is possible to find a maximum matching in a graph in NC. Since good random number generators are hard to come by, it is also a practical issue to remove the use of randomness in algorithms. This paper is concerned with de-randomizing parallel (RNC) algorithms.

A natural approach towards de-randomizing algorithms is to find a method for searching the associated sample space Ω for a good point w with respect to a given input instance I. Given such a point w, the algorithm $\mathcal{A}(I, w)$ is now a deterministic algorithm and it is guaranteed to find a correct solution. The problem faced in searching the sample space is that it is generally exponential in size. The result of Adleman [Ad] showing that $\mathsf{RP} \subseteq \mathsf{P/poly}$ implies that the sample space Ω associated with a randomized algorithm always contains a polynomial-sized subspace which has a good point for each possible input instance. However, this result is highly non-constructive and it appears that it cannot be used to actually de-randomize algorithms. Two methods for searching the sample space have emerged in recent years and are briefly described below.

The first technique is an exhaustive search of the sample space [KW, Lu1, ABI]. Since there is a good point in Ω for each possible input, trying every $w \in \Omega$ is guaranteed to yield a good point. The problem with this approach is that generally the size of the sample space is exponential in the size of the input. This difficulty was overcome in certain cases (the above mentioned papers) by constructing a different polynomial-sized sample space. The new sample space is constructed by showing that the probabilistic choices of the randomized algorithm are only required to be k-wise independent. Then, a sample space of size $O(n^k)$ suffices. An example in which this approach is used is computing in NC a maximal independent set in a graph. This approach seems limited since the degree of independence k must be bounded by some constant, as opposed to the natural assumption of complete independence. An advantage of this technique is that it is especially suited for parallel algorithms because each point in Ω can be checked independently of the other points.

The other approach is that of **the method of conditional probabilities**, a folklore technique (see for example Erdös and Selfridge [ES]) popularized by Spencer [Sp] and Raghavan [Ra]. This method was first formalized by Spencer [Sp] with the aim of converting probabilistic proofs of existence of combinatorial structures into efficient deterministic algorithms for actually constructing these structures. The idea is to perform a **binary search** of the sample space Ω for a good point. (In

Spencer's case, points in Ω correspond to different combinatorial structures and a good point corresponds to a structure which satisfies some desired properties.) At each step of the binary search, the current sample space is split into two equal halves and the conditional probability of obtaining a good point is computed for each half. The search is then restricted to the half where the conditional probability is higher. The search terminates when only one sample point, which must be a good point, remains. This method is applicable to large sample spaces since it requires only $\log|\Omega|$ steps. However, it has the drawback of being extremely sequential in nature.

Our work was motivated in part by the problem posed by Spencer and Raghavan of finding a parallel implementation of their technique. We show how to apply the method of conditional probabilities to randomized parallel (RNC) algorithms. We identify a general class of RNC algorithms which may be de-randomized within our framework. The crucial difference between our work and the techniques described above is the use of higher moment inequalities to bound the tail of the binomial distribution (see Lemma 6.1). This is in contrast to the use of pairwise independence in the work of Luby [Lu1, Lu2] and the use of Chernoff bounds in the work of Spencer [Sp] and Raghavan [Ra]. The fact that we use higher moment inequalities allows us to address more complex algorithms than those that can be handled by pairwise independence; furthermore, unlike in the case of Chernoff bounds, these inequalities apply even in the case where the random bits are chosen with limited (logarithmic) independence, and this allows us to perform efficient parallel binary search on the resulting sample spaces which are of size $O(n^{\mathrm{polylog} n})$.

In general, the conditional probabilities are difficult to compute, and Raghavan [Ra] had introduced the notion of a pessimistic estimator as an efficiently computable substitute for the exact probabilities. Another crucial ingredient of our work is the parallelization of the computation of pessimistic estimators of the conditional probabilities used in the binary search of the sample space (see Section 4).

The main application and motivation of our techniques is the set balancing problem [Sp, OS] which is: given a set system, partition the base set into two parts so that each set is balanced in the sense that it has roughly the same number of elements in each partition class. It will be shown that the RNC algorithm for this problem can be converted into a deterministic one. A generalization of this problem is the lattice approximation problem [Ra] where a point p in an n-dimensional space is given, and we wish to round it to a lattice point q which closely approximates it in terms of the value of their product with a given matrix A. We exhibit a tight relationship between the two problems. To solve the lattice approximation problem, we employ a technique which we call bit-by-bit rounding in which each step is an instance of the vector balancing problem, this in turn is solved by reduction to set balancing. Lattice approximation is also important in the other direction. The precision of the solution to the set balancing problem depends on a parameter ϵ which determines the number of processors required. We also give a bootstrapping method for obtaining an algorithm such that the number of processors is independent of ϵ, but now the running time of the algorithm depends on it. This is achieved by combining a "rough" set balancing algorithm with the (sequential) lattice approximation algorithm for a long and narrow matrix.

An important application of the set balancing algorithm is to the problem of computing **edge colorings** of graphs. We de-randomize an algorithm due to Karloff & Shmoys [KS] which colors the edges of a graph with at most $\Delta + \Delta^{0.5+\epsilon}$ colors, where Δ is the maximum degree in the graph, and ϵ is an arbitrarily small positive constant. This improves upon the previous best-known NC algorithm which used $\Delta + \frac{\Delta}{O(\log^{O(1)} n)}$ colors. (This result follows by combining the work of Arjomandi [Ar] and Karloff & Shmoys [KS]).

A wide class of problems in computational geometry can be efficiently solved by the **random sampling** technique due to Clarkson [Cl1, Cl2, Cl3, CS, HW, RS]. Chazelle & Friedman [CF] reduced the problem of de-randomizing the above mentioned algorithms to computing certain set covers in hypergraphs. We show that this random sampling can be de-randomized in NC using our methods and again the set balancing problem plays an important role.

Finally, we give two more problems to which our techniques can be applied to, namely finding large independent sets in hypergraphs [ABI] and constructing Ramsey graphs.

The rest of this paper is organized as follows. In Section 2 we describe the set balancing and lattice approximation problems and randomized algorithms for them. In Section 3 we present a brief overview of the method of conditional probabilities by describing its application to the sequential set balancing problem. Sections 4 and 5 present the basic idea behind the parallelization of the method of conditional probabilities. In Section 6 we devise an NC algorithm for the set balancing problem using the ideas from the earlier sections and also present the bootstrapping method. In Sections 7 and 8 we show how to solve the vector balancing problem and the lattice approximation problem. In Section 9 we show how our methods give better edge coloring bounds in NC. In Section 10 we show how set balancing can be applied to computational geometry and finally in Section 11, we give two more applications.

A preliminary version of this paper was presented in [MNN]. The idea of de-randomizing by combining limited independence and higher moments with the method of conditional probabilities was discovered independently by Berger & Rompel [BR].

2 Preliminaries

In this section we define and give the background of the two main problems solved in this paper: set balancing and lattice approximation.

2.1 The set balancing problem

Let us first define the problem:

Input: A collection of subsets $\mathcal{S} = \{S_1, S_2, \cdots S_m\}$ of a base set $B = \{b_1, b_2, \cdots b_n\}$ such that each $S_i \subseteq B$.

Output: A partition of B into B^0 and B^1 such that for some small function $f(n,m)$ and for all S_i,

$$\Big| |B^0 \cap S_i| - |B^1 \cap S_i| \Big| \leq f(n,m)$$

The aim is to minimize the value of $f(n,m)$ or, equivalently, find a partition of B such that each subset $S_i \in \mathcal{S}$ has roughly the same number of elements in B^0 and B^1. Initially, we will only concern ourselves with the case where $m = n$ and each set $S_i \in \mathcal{S}$ has cardinality δ. The bounds extend immediately to the case where the cardinality of $\mathcal{S}$ and each $S_i \in \mathcal{S}$ are arbitrary.

Definition 2.1 *A* 2-coloring *of B is an n-bit vector $\vec{C} = (C_1, C_2, \cdots C_n)$ such that $C_i \in \{0,1\}$ is the color of $b_i \in B$. It follows that*

$$B^0 = \{b_i \in B \,:\, C_i = 0\}$$

$$B^1 = \{b_j \in B \,:\, C_j = 1\}$$

Definition 2.2 *The partition of a set $S_i \in \mathcal{S}$ induced by a 2-coloring is given by*

$$S_i^0 = S_i \cap B^0$$

$$S_i^1 = S_i \cap B^1$$

The following notions of discrepancy will help to measure the quality of a solution to the set balancing problem. The discrepancy for a set S_i is essentially the difference in the sizes of the two sets S_i^0 and S_i^1.

Definition 2.3 *The* discrepancy *of a set $S_i \in \mathcal{S}$ of cardinality δ with respect to a 2-coloring $\vec{C}$ is given by*

$$\Delta(S_i, \vec{C}) = \left| |S_i^1| - \frac{\delta}{2} \right| = \left| \sum_{b_j \in S_i} (C_j - \frac{1}{2}) \right|$$

Definition 2.4 *The* discrepancy *of $\mathcal{S}$ with respect to a 2-coloring $\vec{C}$ is given by*

$$\Delta(\mathcal{S}, \vec{C}) = \max_{S_i \in \mathcal{S}} \Delta(S_i, \vec{C})$$

Spencer [Sp1] has shown that for each family $\mathcal{S}$ there exists a 2-coloring $\vec{C}$ such that $\Delta(\mathcal{S}, \vec{C}) \leq 6\sqrt{n}$. This result is the best possible up to constant factors but it has the drawback of being non-constructive, i.e. does not even imply a probabilistic algorithm. Using the method of conditional probabilities Spencer devised a polynomial-time deterministic algorithm which guarantees a 2-coloring $\vec{C}$ such that $\Delta(\mathcal{S}, \vec{C}) = \mathrm{O}(\sqrt{n \log n})$. Beck & Fiala [BF] and, later, Raghavan [Ra] improved this result to finally obtain a discrepancy bound of $\mathrm{O}(\sqrt{\delta \log n})$.

We are interested in devising NC algorithms to compute a 2-coloring $\vec{C}$ such that the discrepancy is small. For parallel algorithms we cannot guarantee as small a discrepancy as in the sequential case. However, we can come arbitrarily close to the sequential bounds by computing an ϵ-good coloring.

Definition 2.5 *Let $\mathcal{S}$ be a set family consisting of exactly $m = n$ subsets with $|S_i| = \delta$, for each $1 \leq i \leq n$. A 2-coloring $\vec{C}$ is ϵ-good for $\mathcal{S}$ if, for $0 < \epsilon < \frac{1}{2}$,*

$$\Delta(\mathcal{S}, \vec{C}) \;=\; \delta^{0.5+\epsilon}\sqrt{\log n}$$

There is a randomized algorithm to find an ϵ-good coloring for any $\mathcal{S}$: pick a random $\vec{C}$ and check that the resulting 2-coloring is ϵ-good for $\mathcal{S}$. The random selection is repeated until an ϵ-good coloring is actually obtained. The task of verifying that a 2-coloring is ϵ-good can be easily performed in NC and, therefore, we also have an RNC algorithm for the set balancing problem. It only remains to demonstrate a probability space for choosing $\vec{C}$ which ensures that the coloring is ϵ-good with high probability.

The random coloring $\vec{C}$ is chosen such that the colors C_i are mutually independent and uniform over $\{0, 1\}$. This corresponds to choosing $\Omega = \{0, 1\}^n$ with the uniform probability measure. Under this assumption we define the following random variables.

Definition 2.6 *Let X_i^0 and X_i^1 denote the number of elements of color 0 and 1, respectively, in the set S_i under the coloring $\vec{C}$. In other words,*

$$X_i^0 = |S_i^0| = \sum_{b_j \in S_i} (1 - C_j)$$

$$X_i^1 = |S_i^1| = \sum_{b_j \in S_i} C_j$$

Each of the $2n$ random variables X_i^c is binomially distributed as $B(\delta, \frac{1}{2})$, The coloring is ϵ-good if, for each i and c, we have that $X_i^c \leq \frac{\delta}{2} + \delta^{0.5+\epsilon}\sqrt{\log n}$. The probability of any these random variables exceeding the stated bound can be estimated by means of the Chernoff bound [Ch].

Theorem 2.1 *Let X be any random variable, $\alpha \geq \mathrm{E}[X]$ and $z > 0$. Then,*

$$\mathrm{Prob}[X \geq \alpha] \leq \frac{\mathrm{E}[e^{zX}]}{e^{z\alpha}}$$

For the particular case of the binomial distribution, Raghavan & Spencer [Ra] have obtained the following from Chernoff's bound.

Theorem 2.2 *Let X be distributed as $B(m, p)$. Then, for any $\lambda > 0$,*

$$\mathrm{Prob}[X \geq (1+\lambda)mp] \leq \left[\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right]^{mp}$$

Applying this bound on the tail of the binomial distribution to the random variable X_i^c, we obtain the following theorem for sufficiently large δ.

Theorem 2.3 *Let $0 < \epsilon < \frac{1}{2}$. Then, for all i and c,*

$$\mathrm{Prob}[X_i^c \geq \frac{\delta}{2} + \delta^{0.5+\epsilon}\sqrt{\log n}] = \mathrm{O}(n^{-2})$$

The discrepancy $\Delta(\mathcal{S}, \vec{C})$ has the same value as that of the largest random variable X_i^c. Summing the probability of exceeding the allowed discrepancy over all X_i^c, we obtain the following corollary. This completes the proof of the randomized algorithm for the set balancing problem.

Corollary 2.1 *If $\vec{C}$ is chosen uniformly at random then,*

$$\mathrm{Prob}[\vec{C} \text{ is } \epsilon\text{-good}] = 1 - \mathrm{O}(n^{-1})$$

In Section 3 we will show how to de-randomize the above to obtain a sequential deterministic algorithm for this problem. Later we will de-randomize the RNC version of the above algorithm to obtain an NC algorithm.

It should be noted that the definition of an ϵ-good coloring for a set system is crucial for the parallel case when using our techniques. As mentioned earlier, in the sequential case, a discrepancy of $O(\sqrt{\delta \log n})$ can be achieved.

2.2 Lattice approximation

The class of problems called the lattice approximation problem were considered by Beck & Fiala [BF] and Raghavan [Ra]. Informally, we are given a point p in n-dimensional space and wish to find a lattice point q, i.e. with integer coordinates, such that q is a "good approximation" to p.

The notion of a good approximation is that the vector $p - q$ has a small inner product with every one of a collection of vectors, say the rows of some matrix A. It is not very hard to see that this problem is related to that of solving a linear programming relaxation of an integer program (see Lovász [Lo]), and then "rounding" that solution to obtain a near-optimum solution to the original integer program. Raghavan [Ra] studied this problem in the context of integer programs for packing problems and showed that it can be used to obtain approximate solutions to a wide class of problems, e.g. packing problems, VLSI routing problems and multicommodity flow problems.

The Lattice Approximation Problem:

Input: An $n \times m$ matrix A, with each entry $A_{ij} \in [0,1]$; a vector $\vec{p} \in [0,1]^m$ and $\vec{c} \in \Re^n$ such that,

$$A \cdot \vec{p} = \vec{c}$$

Output: An m-bit vector $\vec{q} \in \{0,1\}^m$ such that it minimizes the value of the error in the approximation which is given by $\| \, |A \cdot \vec{p} \, - \, A \cdot \vec{q}| \, \|_\infty$.

The set balancing problem described in the previous section is a special case of the lattice approximation problem. The notion of discrepancy extends to the new problem as follows. Let $\vec{\Delta} \in \Re^n$ be the vector of the absolute errors in the approximation, i.e. $\vec{\Delta} = |A \cdot (\vec{p} - \vec{q})|$ and, for each $i \in [1 \cdots n]$, $\Delta_i = |\sum_{j=1}^{n} a_{ij}(p_j - q_j)|$. The discrepancy of a solution $\vec{q}$ is given by $\Delta(A, \vec{p}, \vec{q}) = \|\vec{\Delta}\|_\infty$. Spencer [Sp1] has shown that there always exists a lattice point $\vec{q}$, for each instance of A and $\vec{p}$, such that the discrepancy $\Delta(A, \vec{p}, \vec{q}) \leq 6\sqrt{n}$.

Raghavan [Ra] applied probabilistic methods to the solution of the lattice approximation problem. His randomized rounding algorithm is to choose each q_i to have the value 1 with probability p_i and the value 0 with probability $1 - p_i$. This procedure can be de-randomized using the method of conditional probabilities to set the bits of $\vec{q}$ and the same bounds on the discrepancy that are obtained probabilistically, can also be achieved deterministically. His method constructs a vector $\vec{q}$ such that the i^{th} discrepancy is $\Delta_i = O(\sqrt{c_i \log n})$. This technique can be easily parallelized to give an RNC algorithm for the lattice approximation problem. Unfortunately, it turns out that we cannot de-randomize this particular algorithm. We will later describe a different RNC algorithm, the *bit-by-bit randomized rounding algorithm*, to which our techniques can be extended.

3 The method of conditional probabilities

In this section we first provide a brief description of the sequential version of the method of conditional probabilities. We then provide some evidence why the most general form of this method is not amenable to parallelization. We will illustrate the basic idea by applying it to the de-randomization of the randomized set balancing algorithm. Suppose the input family of sets $\mathcal{S}$ is fixed. We have established that a random coloring $\vec{C}$ is ϵ-good for $\mathcal{S}$ with a non-zero probability. The probability space used for selecting $\vec{C}$ consists of the sample space $\Omega = \{0,1\}^n$ with the uniform probability measure. We now wish to perform a binary search in Ω for an ϵ-good coloring.

Let P denote the probability that $\vec{C}$ is **not** ϵ-good for $\mathcal{S}$; we know that $P < 1$. Further, let P^0 and P^1 denote the **conditional** probability that $\vec{C}$ is not ϵ-good for $\mathcal{S}$ given that $C_1 = 0$ and $C_1 = 1$, respectively. These probabilities are related as follows.

$$P = \frac{1}{2}P^0 + \frac{1}{2}P^1$$

Since P is a convex combination of P^0 and P^1, it follows that $\min\{P^0, P^1\} \leq P < 1$. This means there is a way of choosing the value of the first color C_1 such that if the remaining colors are chosen at random then the probability of obtaining an ϵ-good coloring will still be non-zero. If we could compute the conditional probabilities P^0 and P^1 then we could choose the appropriate value for C_1. This process can be repeated for the remaining colors. At a general stage of this process we would have fixed the first k colors already and would be trying to determine the value of C_{k+1}. This could be done by computing the conditional probabilities for both choices of C_{k+1} under the assumption that the first k colors are fixed as before and the last $n-k-1$ colors are chosen uniformly at random. At each stage of the process it would be ensured that the conditional probability of **not** obtaining an ϵ-good coloring does not increase.

The process terminates when all the colors have been fixed. At this point we are guaranteed that the probability of not obtaining an ϵ-good coloring is less than 1 and, therefore, we have such a coloring. Thus, we have an n-stage deterministic algorithm which always finds an ϵ-good coloring. Of course, we need to efficiently compute the appropriate conditional probabilities and we refer the reader to [Sp, Ra] for more details on how this can be done.

Note that the above process corresponds to a binary search of the sample space since fixing the colors one-by-one is equivalent to halving the sample space at each point. In general, such an algorithm would require $\log |\Omega|$ stages of computing conditional probabilities. Unfortunately, there does not seem to be any obvious way of parallelizing the search process. Another problem with this approach is that the conditional probabilities are hard to compute even sequentially, so it is not clear how any conditional probability can be computed efficiently in parallel. In the next section we show how these obstacles can be overcome by our techniques.

4 Parallelizing the method of conditional probabilities

In general, the method of conditional probabilities is not applicable to every randomized parallel algorithms. However, we will specify certain conditions under which we can obtain a parallel implementation of this method for a special class of problems.

Suppose we have a randomized algorithm with the associated sample space Ω such that $|\Omega| = 2^h$.

We assume that there is a one–to-one correspondence f between the points in Ω and the h-bit vectors in $\{0,1\}^h$. Given $\vec{R} = (r_1, r_2, \ldots, r_h) \in \{0,1\}^h$, the corresponding sample point $w = f(\vec{R})$ is assumed to be NC–computable. The sample space Ω can be sampled uniformly at random by choosing a random vector $\vec{R}$ from $\{0,1\}^h$ and computing $f(\vec{R})$.

For $1 \leq m \leq h$, let $P_m(d_1, d_2 \ldots d_m)$ denote the conditional probability that $f(\vec{R}) \in \Omega$ is not a good point given that the first m bits of the vector $\vec{R}$ are equal to $d_1, d_2 \ldots d_m$ and the remaining $h - m$ bits of $\vec{R}$ are chosen uniformly at random. We use P_m to denote the value of the conditional probability when the choice of the parameters d_1, d_2, ..., d_m is clear from the context.

We present below a set of conditions which must be satisfied by the sample space Ω so that we can implement the method of conditional probabilities in NC.

1. The sample space Ω contains at least one good point for each possible input instance. In other words, for any input I, if $w \in \Omega$ is chosen uniformly at random then $P_0 = \text{Prob}[w \,\text{is not good for}\, I] < 1$
2. The conditional probabilities, $P_m(d_1, d_2 \ldots d_m)$, are NC-computable.
3. $h = \log|\Omega| = \text{polylog}(n)$, where n is the input size.

For such a sample space, the method of conditional probabilities can be used to obtain an NC algorithm. The idea is to fix the bits of $\vec{R}$ one-by-one and at each step, to compute the conditional probabilities, P_m, to ensure that the probability of obtaining a good solution using $\vec{R}$ is non-decreasing. The number of steps in this process is h, which is poly-logarithmic, and each step can be implemented in NC because of condition 2.

In the next section we will demonstrate how to construct such sample spaces and present some general techniques for computing conditional probabilities over such a sample space. We will need the following notion of an approximation of the conditional probability.

A major stumbling block in applying the method of conditional probabilities is always the computation of the conditional probabilities. It turns out that in most cases it suffices to compute an estimator of the conditional probabilities rather than the exact value. Following Raghavan [Ra], we call $\hat{P_m}$ a **pessimistic estimator** of the conditional probability P_m if it satisfies the following conditions. (In Raghavan's case condition 4 required that the estimator be computable in polynomial time, unlike our condition that it be computable in NC.)

1. $\hat{P_0} < 1$.
2. For any partial setting of the first m bits of $\vec{R}$, $P_m \leq \hat{P_m}$.

3. For any $d_1, d_2, \ldots d_m$,

$$\min\{\hat{P_m}(d_1, d_2, \ldots d_m, 0), \hat{P_m}(d_1, d_2, \ldots d_m, 1)\} \leq \hat{P_m}(d_1, d_2 \ldots d_m)$$

4. The pessimistic estimators can be computed in NC.

It is not very hard to see that such a pessimistic estimator can equally well be used in the method of conditional probabilities instead of the exact conditional probabilities which are hard to compute in general.

4.1 Limitation on parallelization

To see that the general form described in Section 3 is not amenable to parallelization, we consider the following concrete complexity type scenario. We are given a discrete space Ω in which the points are each labeled either *good* or *bad*. For any subset $\Omega' \subset \Omega$, let $P(\Omega')$ denote the number of good points in that subset. The problem is to identify any one good point in Ω under the following computation model. We have q processors, each processor can compute $P(\Omega')$ in one time-unit for any $\Omega' \subset \Omega$. After each round of computation, the processors can share all the information they have gained so far and decide on the queries of the next round. However, we assume that the only access the processors have to Ω is through a "black-box" computation of P, and the complexity of deciding which queries to ask is completely ignored. P is also assumed to have the properties listed above. The sequential complexity (one processor) of finding a good point under such conditions is $\log|\Omega|$. The next theorem tells us that having q processors hardly helps.

Theorem 4.1 *For all q, the number of rounds required to find a good point in Ω using q processors is at least $\frac{\log|\Omega|}{\log(q+1)}$.*

The proof of the theorem follows immediately from an adversary argument similar to the one given in Theorem 1 of [KUW2]. If the function $P(\Omega')$ is better than the one promised above, and tells us the exact number of good points in Ω', then the adversarial argument in the proof does not work. However we conjecture that no significant speed up is possible in this case as well. As with all concrete complexity results, this one says nothing about the possibility of computing a good point outside the black-box model.

5 Designing and searching small sample spaces

In this section we present a construction for sample spaces which satisfy the conditions required for parallelizing the method of conditional probabilities. We also show that a pessimistic estimator for the conditional probabilities can be found for such sample spaces.

We will assume that the random choices made by an algorithm $\mathcal{A}$ can be represented by an n-bit vector $\vec{C} = (C_1, C_2, \ldots, C_n) \in \{0,1\}^n$. Thus, the associated probability space can be represented by the space $\{0,1\}^n$ with the uniform probability measure. This is equivalent to saying that $\mathcal{A}$ chooses n random bits uniformly and independently. For example, in the case of the sequential set balancing algorithm, the random variable C_i represents the color chosen for the element b_i.

We describe below a new sample space $\Omega \subseteq \{0,1\}^n$ which will have the desired properties. The sample space will also have the property that if the vector $\vec{C}$ is chosen uniformly at random from Ω then the following conditions will hold.

1. Each C_i is uniformly 0 or 1.
2. Any k random variables in $\vec{C}$ are mutually independent, where k is poly-logarithmic in n.
3. For any input $\mathcal{I}$, $\mathrm{Prob}[\vec{C}\,\text{is good for}\,I] > 0$.

In Section 5.1 we present an explicit construction of such a sample space. It is clear that this sample space can be searched in NC using the method of conditional probabilities. In Section 5.2 we show how the special structure of this sample space can be used to provide a pessimistic estimator for the relevant conditional probabilities.

5.1 Constructing a k-wise independent sample space

We wish to construct a small sample space such that a random n-bit vector from the space has the k-wise independence property. We will make use of a convenient construction which was suggested by Alon, Babai & Itai [ABI]. We will have n binary vectors, $\vec{H}_1, \vec{H}_2, \ldots \vec{H}_n$ so that any k of these are linearly independent over GF(2). To meet this end, each $\vec{H}_i$ would be a column in the matrix H defined as follows. Assume $n = 2^t - 1$ and let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be the the nonzero elements of the field $GF(2^t)$ where each element is treated as a binary column vector of length t.

$$H = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \ldots & \alpha_n \\ \alpha_1^3 & \alpha_2^3 & \alpha_3^3 & \ldots & \alpha_n^3 \\ \alpha_1^5 & \alpha_2^5 & \alpha_3^5 & \ldots & \alpha_n^5 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \alpha_3^{k-1} & \ldots & \alpha_n^{k-1} \end{pmatrix}$$

This is the parity check matrix of BCH codes. The length of each column is $h = t \cdot k/2$. The next proposition is proved in [MS].

Proposition 5.1 *The collection of vectors $(\vec{H}_1, \vec{H}_2, \ldots, \vec{H}_n)$ is k-wise linearly independent over $GF(2)$.*

Let $\vec{R} = (r_1, r_2, \ldots, r_h)$ be a vector whose components are mutually independent uniform $\{0,1\}$ random variables. The random variable C_i is defined to be $\vec{H}_i \cdot \vec{R}$, namely the inner product of the two vectors. The next proposition follows from basic linear algebra.

Proposition 5.2 *The variables $(C_1, C_2, \ldots, C_n)$ are k-wise stochastically independent Bernoulli random variables which are uniform over $\{0,1\}$.*

5.2 Evaluating estimators in the sample space

We now show that estimators of a certain form can be evaluated in sample spaces like the one constructed in the previous section. It is often the case that an estimator is the expected value of some function of $C_1, C_2, \ldots C_n$. In order to implement the method of conditional probabilities in such a case, we need to be able to compute the conditional expectation of this function, given that the first m bits of $\vec{R}$ are chosen to be $(d_1, d_2, \ldots d_m)$.

We will first show how to compute the conditional expectation of a function of the form $T = C_1 \cdot C_2 \cdots C_t$ where $1 \le t \le n$. T can be a function of any t of the random variables but, without loss of generality, we demonstrate our method for the variables with the t smallest indices. This implies that we can compute functions of the form $\sum_i z_i T_i$, where each z_i is some constant.

Lemma 5.1 *For all $0 \le m \le h$, the conditional expectation of T,*

$$E[T = C_1 C_2 \cdots C_t | (r_1, r_2, \ldots r_m) = (d_1, d_2, \ldots d_m)]$$

can be computed in NC.

Proof: For all i, separate $\vec{H}_i$ into two parts, $\vec{H}_i^d$ and $\vec{H}_i^r$, where $\vec{H}_i^d$ consists of the first m bits of $\vec{H}_i$, and $\vec{H}_i^r$ are the last $h - m$ bits of $\vec{H}_i$. The vector $\vec{R}$ is separated similarly into $\vec{R}^d$ and $\vec{R}^r$; also $C_i^r = \vec{H}_i^r \cdot \vec{R}^r$ and $C_i^d = \vec{H}_i^d \cdot \vec{R}^d$. Hence, $C_i = C_i^d \oplus C_i^r$, where $\oplus$ denotes the addition operation in GF(2).

Obviously, to compute the expected value of T, we just need to compute the conditional probability $Pr[T = 1 | (r_1, r_2, \ldots r_m) = (d_1, d_2, \ldots d_m)]$. For all vectors $\vec{R}$ such that $(r_1, r_2, \ldots r_m) = (d_1, d_2, \ldots, d_m)$, $T = 1$ at $f(\vec{R})$ if and only if, for all j, $1 \le j \le t$, $C_j^d \oplus C_j^r = 1$.

To compute the probability of this happening, find a basis over GF(2) for $\vec{H}_1^r, \vec{H}_2^r, \ldots \vec{H}_t^r$. Let the cardinality of the basis be ℓ and assume for convenience that the basis vectors are $\vec{H}_1^r, \vec{H}_2^r, \ldots, \vec{H}_\ell^r$. For all j such that $\vec{H}_j^r$ is a non basis vector, C_j^r is determined by $C_1^r, C_2^r, \ldots, C_\ell^r$ in the following way. Let $\vec{H}_j^r = \sum_{q=1}^{\ell} a_q \vec{H}_q^r$ where $a_q \in \{0,1\}$. We consider only the case where $C_1 = 1, C_2 = 1, \ldots, C_\ell = 1$. Then, for $\ell < j \le t$,

$$C_j = C_j^d \oplus C_j^r = C_j^d \oplus (\sum_{q=1}^{\ell} a_q \cdot \vec{H}_j^r \cdot \vec{R}^r) = C_j^d \oplus \sum_{q=1}^{\ell} a_q \cdot C_q^r = C_j^d \oplus \sum_{q=1}^{\ell} a_q \cdot (1 \oplus C_q^d).$$

Notice that the value of C_j can now be determined by the bits of $\vec{R}$ that were already set. There are two cases to consider:

- If, for some $\ell < j \leq t$, $C_j^d \oplus \sum_{q=1}^{\ell} a_q \cdot (1 \oplus C_q^d) \neq 1$, then whichever way the rest of $\vec{R}$ is set, $T = 0$, and hence
$$Pr[T = 1|(r_1, r_2, \ldots r_m) = (d_1, d_2, \ldots d_m)] = 0$$
- If, for all $\ell < j \leq t$, $C_j^d \oplus \sum_{q=1}^{\ell} a_q \cdot (1 \oplus C_q^d) = 1$, then for each choice of $\vec{R}$ such that $C_1 = C_2 = \ldots = C_\ell = 1$, all the C_j's are 1 and $T = 1$. Since $\vec{H}_1^r, \vec{H}_2^r, \ldots, \vec{H}_\ell^r$ are independent, $Pr[C_1 = C_2 = \ldots = C_\ell = 1] = 2^{-\ell}$ and hence,
$$Pr[T = 1|(r_1, r_2, \ldots r_m) = (d_1, d_2, \ldots d_m)] = 2^{-\ell}$$

There are NC algorithms for finding a basis and the rank of a set of vectors (see the results of Mulmuley [Mu]). Hence we can compute $E[T = 1|(r_1, r_2, \ldots r_m) = (d_1, d_2, \ldots d_m)]$ in NC. ▮

Suppose the pessimistic estimator is the expected value of a function which is the sum of a polynomial number of such T_i's. It follows from the last lemma that this estimator can be computed in NC. To apply the techniques outlined in Section 4 for a given problem, one has to give an algorithm where limited independence suffices, and then provide an estimator for that algorithm.

6 An NC algorithm for set balancing

We now apply the ideas developed in the previous sections to the construction of an NC algorithm for the set balancing problem. The new sample space $\Omega \subset \{0,1\}^n$ for set balancing will consist of $|\Omega| = 2^h$ coloring vectors where the exact value of h will be specified later. It will be guaranteed that if a 2-coloring $\vec{C}$ is chosen uniformly at random from Ω then it has the following properties.

1. Each C_i is uniformly 0 or 1.
2. Any k colors in $\vec{C}$ are mutually independent, where k is poly-logarithmic in n.
3. For any $\mathcal{S}$, $\text{Prob}[\vec{C}\,\text{is}\,\epsilon\text{−good for}\,\mathcal{S}] > 0$.

We need to prove two results to show that the parallel implementation of the method of conditional probabilities leads to an NC algorithm. First, we need to show that for any input $\mathcal{S}$ there exists at least one good point in the new sample space. Then, we need to demonstrate that the conditional probabilities of obtaining good points can be estimated by the technique outlined in the previous section.

6.1 Limited independence suffices

In this section we show that a polylog bound on k, the amount of independence, suffices to assure that a random coloring is ϵ-good with positive probability. Let C_i denote the color of element b_i. The color C_i is a random variable defined as follows:

$$C_i = \begin{cases} 0 & \text{with probability } 0.5 \\ 1 & \text{with probability } 0.5 \end{cases}$$

For each subset S_i we also define a random variable X_i as follows: $X_i = \sum_{b_j \in S_i} C_j$. The expected value of X_i is $E[X_i] = \delta/2$; define the normalized random variable $Y_i = X_i - \delta/2$.

The discrepancy for which we are aiming is $disc = \delta^{0.5+\epsilon}\sqrt{\log n}$. The following lemma shows that this is possible.

Lemma 6.1 *Let $k = a\log n/\log\delta$ be even, where $a > \frac{1}{\epsilon}$, and let the random variables C_i (for all $b_i \in B$) be k-wise independent. Then, for any input set family $\mathcal{S}$, the probability that $\vec{C}$ is ϵ-good is non-zero.*

Proof: First, assume that $\delta > \log n > k$. Otherwise $\delta^{\frac{1}{2}+\epsilon}\sqrt{\log n} > \delta$, implying that *any* coloring vector will be ϵ-good (since the discrepancy can never exceed δ) and that the lemma is trivially true.

We will show that

$$\text{Prob}[|X_i - E[X_i]| > disc] < \frac{1}{n}$$

This implies that

$$\sum_{S_i \in \mathcal{S}} \text{Prob}[|X_i - E[X_i]| > disc] < 1$$

which in turn implies that there exists a coloring which is ϵ-good. The k^{th} moment inequality [Fe] states that

$$\text{Prob}(|X_i - E[X_i]| \geq \lambda) \leq \frac{E[|X_i - E[X_i]|^k]}{\lambda^k}$$

We will assume that k is even and hence,

$$|X_i - E[X_i]|^k = (X_i - E[X_i])^k$$

The expression to be evaluated is $E[Y_i^k]$. For $1 \leq j \leq n$ let $\hat{C}_j = C_j - \frac{1}{2}$ and let the elements of S_i be $b_1, b_2, \ldots, b_\delta$, without loss of generality.

$$Y_i^k = (\hat{C}_1 + \hat{C}_2 \ldots + \hat{C}_\delta)^k = \sum_{i_1 i_2 \ldots i_k} \hat{C}_{i_1}\hat{C}_{i_2}\ldots\hat{C}_{i_k}$$

and hence, by the linearity of expectations,

$$E[Y_i^k] \;=\; E\left[\sum_{i_1 i_2 \ldots i_k} \hat{C}_{i_1}\hat{C}_{i_2}\ldots\hat{C}_{i_k}\right] \;=\; \sum_{i_1 i_2 \ldots i_k} E\left[\hat{C}_{i_1}\hat{C}_{i_2}\ldots\hat{C}_{i_k}\right]$$

Each term in the above summation is of the form $\hat{C}_{i_1}^{p_1}\hat{C}_{i_2}^{p_2}\ldots\hat{C}_{i_r}^{p_r}$, such that $\sum_{j=1}^{r} p_{i_j} \;=\; k$, and $p_{i_j} > 0$. Since the random variables $\hat{C}_1, \hat{C}_2, \ldots, \hat{C}_n$ are k-wise independent we obtain that

$$E[Y_i^k] \;=\; \sum_{i_1 \neq i_2 \neq \ldots \neq i_r} E[\hat{C}_{i_1}^{p_1}]E[\hat{C}_{i_2}^{p_2}]\ldots E[\hat{C}_{i_r}^{p_r}]$$

It now follows from the definition of $\hat{C}_i$ that

$$E[\hat{C}_i^{p_i}] = \begin{cases} 0 & \text{if } p_i \text{ is odd} \\ \frac{1}{2^{p_i}} & \text{if } p_i \text{ is even} \end{cases}$$

Thus, only the terms where all the exponents are even contribute to the sum. Also, recall that we are only concerned with the case where $\delta > k$. It is easy to see that the number of such terms is dominated by the number of terms where all exponents are exactly 2. Therefore, for some constant $\alpha > 1$, there are at most $\alpha \cdot \binom{\delta}{\frac{k}{2}} \cdot \frac{k!}{2^{\frac{k}{2}}}$ terms in which the exponents are even. Each term has a value of $\frac{1}{2^k}$ and, therefore,

$$\begin{aligned} E[Y_i^k] \;&\leq\; \alpha \cdot \frac{1}{2^k} \cdot \binom{\delta}{\frac{k}{2}} \cdot \frac{k!}{2^{\frac{k}{2}}} \\ &<\; (\delta k)^{k/2} \end{aligned}$$

Substituting $\lambda = disc = \delta^{0.5+\epsilon}\sqrt{\log n}$ in the k^{th} moment inequality, we get that

$$\text{Prob}[|X_i - E[X_i]| > disc] \;<\; \left(\frac{k}{\delta^{2\epsilon}\log n}\right)^{\frac{k}{2}}$$

We have that $k = \frac{a \log n}{\log \delta}$ where $a > \frac{1}{\epsilon}$. Thus, we get the desired result that,

$$\text{Prob}[|X_i - E[X_i]| > disc] \;<\; \frac{1}{\delta^{\epsilon k}} \;<\; \frac{1}{n}$$

∎

We remark that, unlike the sequential case, we are unable to obtain a discrepancy of $\delta^{0.5}$polylog$n$. This is because, as shown in the next section, the number of processors required for computing the conditional probability is δ^k and we must ensure that is polynomial in n. On the other hand, in the previous inequality, we require that the error probability be at most $1/n$. The only way to reconcile these two conflicting requirements is to allow an extra factor of δ^ϵ in the discrepancy bound.

6.2 Finding an ϵ-good coloring

Recall that for each ground set element $b_i \in B$, its color is $C_i = \vec{H}_i \cdot \vec{R}$, namely the inner product of the two vectors.

Let the family $\mathcal{F}$ be the different colorings defined by the 2^h possible settings of $\vec{R}$. It follows from Lemma 6.1 that for each input set family $\mathcal{S}$, $\mathcal{F}$ contains an ϵ-good coloring.

An ϵ-good coloring in $\mathcal{F}$ is computed by setting the bits of the vector $\vec{R}$, one bit at a time. We know that initially

$$\frac{\sum_{S_i \in \mathcal{S}} E[Y_i^k]}{(disc)^k} < 1$$

Suppose that the first m bits of $\vec{R}$ have already been set such that $r_1 = d_1, r_2 = d_2, \ldots, r_m = d_m$. For shorthand notation let

$$E_m[Y_i^k] = E[Y_i^k | (r_1, r_2, \ldots, r_m) = (d_1, d_2, \ldots, d_m)]$$

We assume inductively that the following holds:

$$\frac{\sum_{S_i \in \mathcal{S}} E_m[Y_i^k]}{(disc)^k} < 1$$

For each subset $S_i \in \mathcal{S}$, the definition of expectation implies that,

$$E_m[Y_i^k] = \frac{1}{2} E_m[Y_i^k | r_{m+1} = 0] + \frac{1}{2} E_m[Y_i^k | r_{m+1} = 1]$$

It follows from the linearity of expectation that there exists a $d_{m+1} \in \{0, 1\}$ such that

$$\frac{\sum_{S_i \in \mathcal{S}} E_{m+1}[Y_i^k]}{(disc)^k} < 1$$

This can be summarized by the following algorithm:

for $m := 0$ **to** $h - 1$ **do**

  `compute` $t_0 = \sum_{S_i \in \mathcal{S}} E_m[Y_i^k | r_{m+1} = 0]$

  `compute` $t_1 = \sum_{S_i \in \mathcal{S}} E_m[Y_i^k | r_{m+1} = 1]$

  if $t_0 < t_1$ **then** `set` $d_{m+1} = 0$

  else `set` $d_{m+1} = 1$

end

We now explain how to compute t_0 and t_1 in the above algorithm.

$$E_m[Y_i^k] = \sum_{i_1 i_2 \ldots i_k} E\left[\hat{C}_{i_1}\hat{C}_{i_2}\ldots\hat{C}_{i_k} | (r_1, r_2, \ldots, r_m) = (d_1, d_2, \ldots, d_m)\right]$$

This sum contains exactly δ^k terms and recall that

$$\delta^k = \delta^{\frac{a \log n}{\log \delta}} = n^a$$

If the value of one term can be computed in NC then we can allocate enough processors to compute all terms. This is exactly the kind of function for which the method given in Section 4.2 is applicable. However, we can be more efficient in this case.

Let us focus on computing one conditional expectation term of the form $E[T|(r_1, r_2, \ldots, r_m) = (d_1, d_2, \ldots, d_m)]$ where, without loss of generality, $T = \hat{C}_1\hat{C}_2\ldots\hat{C}_k$. Unlike the initial case where the random variables are assured to be k-wise independent, once we start setting the bits in the vector $\vec{R}$, this property is lost. We go about computing this term in a different way.

Recall that $\hat{C}_i = \vec{H}_i \cdot \vec{R} - \frac{1}{2}$ and hence equal to $\frac{1}{2} \cdot (-1)^{\vec{H}_i \cdot \vec{R}}$. Using the notation of Section 5.2, we have that $\vec{H}_i = (\vec{H}_i^d, \vec{H}_i^r)$, $\vec{R} = (\vec{R}^d, \vec{R}^r)$ and $C_i = \vec{H}_i^d \cdot \vec{R}^d \oplus \vec{H}_i^r \cdot \vec{R}^r = C_i^d \oplus C_i^r$. Thus,

$$\hat{C}_i = \frac{1}{2}(-1)^{(\vec{R}^d, \vec{R}^r)\cdot(\vec{H}_i^d, \vec{H}_i^r)} = \frac{1}{2}(-1)^{(\vec{R}^d \cdot \vec{H}_i^d)} \cdot (-1)^{(\vec{R}^r \cdot \vec{H}_i^r)}$$

But the value of $(-1)^{(\vec{R}^d \cdot \vec{H}_i^d)}$ is fixed and known beforehand.

$$E[\hat{C}_1\hat{C}_2\ldots\hat{C}_k|(r_1, r_2, \ldots, r_m) = (d_1, d_2, \ldots, d_m)] = \frac{1}{2^k} E[\prod_{j=1}^{k} (-1)^{(\vec{R}^d, \vec{R}^r)\cdot(\vec{H}_i^d, \vec{H}_i^r)}]$$

The right hand side of the above equation can be written as

$$\left(\frac{1}{2}\right)^k (-1)^{\sum_{j=1}^k (\vec{R}^d \cdot \vec{H}_i^d)} E[(-1)^{\sum_{j=1}^k (\vec{R}^r \cdot \vec{H}_i^r)}]$$

The following proposition can easily be verified.

Proposition 6.1

$$E[(-1)^{\sum_{i=1}^k (\vec{R}^r \cdot \vec{H}_i^r)}] = \begin{cases} 0 & \textit{if } \vec{H}_1^r \oplus \vec{H}_2^r \oplus \ldots \oplus \vec{H}_k^r \neq (0, 0, \ldots, 0) \\ 1 & \textit{otherwise} \end{cases}$$

Thus the conditional expectations can be evaluated by computing $\vec{H}_1^r \oplus \vec{H}_2^r \oplus \ldots \oplus \vec{H}_k^r$ for each term.

Finally, note that in Lemma 6.1 we had assumed that $\delta > k$, since otherwise any coloring vector would be ϵ-good. Therefore, we do not need to consider the case $\delta < k$ while computing the pessimistic estimator for the conditional probability.

Theorem 6.1 *An ϵ-good coloring can be found for the set balancing problem in* $O(\log^3 n)$ *time using* n^{a+1} *processors, for any* $a > 1/\epsilon$.

A similar result was obtained independently by [BR].

6.3 Generalizations of the set balancing algorithm

In the next theorem we show that the bound on the discrepancy can be strengthened in the case where we are guaranteed that the value of δ is sufficiently large.

Theorem 6.2 *Let* $\delta = \Omega(\log^{\frac{1}{\epsilon}} n)$ *and* $DISC = \delta^{\frac{1}{2}+\epsilon}$. *Then, there is an NC algorithm for set balancing which computes a coloring vector* $\vec{C}$, *for any set family* $\mathcal{S}$, *such that* $\Delta(\mathcal{S}, \vec{C}) \leq DISC$.

Proof: This result can be obtained by replacing *disc* by *DISC* in the construction of the pessimistic estimator described above. The proof of Lemma 6.1 also applies to this case as follows. We need to show that

$$\text{Prob}[|X_i - E[X_i]| > DISC] \ < \ \frac{1}{n}$$

As in Lemma 6.1, we have that

$$\text{Prob}[|X_i - E[X_i]| > DISC] \ < \ \left(\frac{k}{\delta^{2\epsilon}}\right)^{\frac{k}{2}} \ \leq \ \left(\frac{k}{\log^2 n}\right)^{\frac{k}{2}}$$

The latter term is bounded by $1/n$. ∎

The results of the set balancing problem were presented only for the case where the set family had $m = n$ subsets and each subset was of cardinality δ. We first extend the definition of an ϵ-good coloring to the more general case and then show that even the more general problem can be solved in NC.

Definition 6.1 *Let* $\mathcal{S}$ *be a set family consisting of* m *subsets with* $|S_i| = \delta_i$, *for each* $1 \leq i \leq m$. *A 2-coloring* $\vec{C}$ *is* ϵ*-good for* $\mathcal{S}$ *if, for* $0 < \epsilon < \frac{1}{2}$ *and* $1 \leq i \leq m$,

$$\Delta(S_i, \vec{C}) \ = \ disc_i \ = \ \delta_i^{0.5+\epsilon}\sqrt{\log m}$$

Theorem 6.1 can now be generalized as follows. This generalization is also applicable to the stronger guarantee on the discrepancy as presented in Theorem 6.2 above.

Theorem 6.3 *Let* $\mathcal{S}$ *be a set family consisting of exactly* m *subsets with* $|S_i| = \delta_i$, *for each* $1 \leq i \leq m$. *An* ϵ*-good coloring can be found for the set balancing problem involving* $\mathcal{S}$ *in NC.*

Proof: Consider first the case where $m \neq n$. This only affects the proof of Lemma 6.1 to the extent that we now need to show that

$$\text{Prob}[|X_i - E[X_i]| > disc] < \frac{1}{m}$$

since we will be summing this probability over m subsets instead of n as before. This is easily taken care of by modifying the value of k to be $\frac{a \log m}{\log \delta}$. In the case where $m > n$, these changes will also affect the running time and processor count for the algorithm. The running time will be polylogarithmic in m and the processor count will become polynomial in m. However, this is still an NC algorithm since the input size also depends linearly on m.

Consider now the case where the subsets are not all of the same size. For $1 \leq i \leq m$, define $k_i = \frac{a \log m}{\log \delta_i}$, and choose $k = \max_{i=1}^{m} k_i$. The idea is to choose the random coloring with k-wise independence as before. Recall that the pessimistic estimator of the conditional probability was earlier defined as the sum of the k^{th}-moment estimators for each subset S_i. In the current setting, the pessimistic estimator for the i^{th} subset will use only k_i-wise independence and will be defined using the k_i-moment inequality. This means that in the proof of Lemma 6.1 we have that

$$\text{Prob}[|X_i - E[X_i]| > disc_i] < \frac{1}{{\delta_i}^{\epsilon k_i}} < \frac{1}{m}$$

Moreover, each pessimistic estimator has ${\delta_i}^{k_i} = n^{\mathrm{O}(1)}$ terms and can be computed in NC. Note that the k_i-moment is not well-defined in the case where $\delta_i < k_i$. However, as we remarked in the proof of Lemma 6.1, any such set S_i will be colored well by any coloring vector and can be dropped from the problem instance altogether. ▮

6.4 Bootstrapping

In this section we present an algorithm for the set balancing problem in which the number of processors does not depend on ϵ. The price we pay is that our time complexity is now $\mathrm{O}(\log^{1/2\epsilon} n)$. However, the total work, i.e the number of processors multiplied by the time, can now be bounded by a polynomial independent of ϵ.

The general idea is to partition first the ground set into ℓ parts using a roughly-good coloring, i.e. a γ-good coloring, where γ is a fixed constant independent of ϵ. This is done via the algorithm of the previous section. Then, an ϵ-good solution for the set balancing problem is computed recursively in each of the ℓ parts. Finally, we combine the solutions obtained for the different parts, and take advantage of the freedom to interpret the colors in each part, so as to minimize the global discrepancy. This is done using Raghavan's sequential lattice approximation algorithm [Ra]. It turns out that the matrix associated with this instance of the lattice approximation problem is of dimension $n \times l$, and Raghavan's algorithm requires $\mathrm{O}(\ell)$ steps. Our total running time remains polylogarithmic, as $\ell = \lceil 2 \log^{\frac{1}{2\epsilon}} n \rceil$. (We assume that ℓ is a power of 2).

Assume that the size of each subset is at most d, and for simplicity also assume that the discrepancy we are aiming for is $d_0 \cdot d^{1/2+\epsilon}$ for $d_0 = \log^6 n$. (This can be justified by taking a smaller ϵ to absorb d_0). The general scheme to achieve that is the following:

Algorithm Bootstrap$(\mathcal{S}, B)$
$\{\mathcal{S}$ is the set system and B is the ground set $\}$.

1. Partition the ground set B into ℓ parts $D_1, D_2, \ldots D_\ell$ so that for each subset S_i and each part D_j, $|S_i \cap D_j| < 2d/\ell$.

2. For each part D_j, solve recursively the problem induced by that part: compute D_j^0 and D_j^1, a 2-coloring of D_j, such that the discrepancy of the 2-coloring is bounded by $d_0 \cdot d^{1/2+\epsilon}$.

3. Merge the solutions obtained in step 2: find a vector $\vec{q} \in \{0,1\}^\ell$ such that $B^0 = \cup_{j=1}^{\ell} D_j^{q_j}$ and $B^1 = \cup_{j=1}^{\ell} D_j^{1-q_j}$. The discrepancy of the 2-coloring should be at most $d_0 \cdot \left(\frac{2d}{\ell}\right)^{1/2+\epsilon}$.

Let us now elaborate on how each step is implemented. It is worth noting that at each of the three steps of the algorithm, a set balancing procedure is applied.

6.4.1 Step 1

This step is implemented by $\log \ell$ recursive calls to the set balancing algorithm (note that a similar idea was used by Anderson [An]). Fix γ to be $\frac{1}{4}$. Each call computes a γ-good coloring. Step 1 is implemented by calling Algorithm Partition$(\mathcal{S}, B, \log \ell)$.

Algorithm Partition$(\mathcal{S}, B, j)$
$\{\mathcal{S}$ is the set system, B is the ground set, and j is the recursion depth.$\}$.

1. Partition the ground set B into 2 parts B^0 and B^1 by computing a γ-good coloring. Let $\mathcal{S}^i$ denote the set system induced by B^i for $i = 0, 1$.

2. If $j > 1$, then for $i = 0, 1$: Partition$(\mathcal{S}^i, B^i, j-1)$.

We denote the parts obtained at the bottom of the recursion by $D_1, D_2, \ldots, D_\ell$. Let S_{ij} denote $S_i \cap D_j$.

Proposition 6.2 *For all $1 \le i \le n$ and $1 \le j \le \ell$, $|S_{ij}| < \max\{\frac{2d}{\ell}, d_0\}$.*

Proof: Let T be a subset obtained in some recursive call to Algorithm Partition, and let T^0 and T^1 be the partitioning of T generated by the γ-good coloring. The discrepancy of the γ-good coloring is bounded by $|T|^{0.5+\gamma}\sqrt{\log n}$. Assume $|T| > d_0 = \log^6 n$. Then, for $i = 0, 1$:

$$|T^i| \leq \frac{|T|}{2} + |T|^{0.5+\gamma}\sqrt{\log n} \leq |T| \cdot \left(\frac{1}{2} + \frac{1}{\log n}\right)$$

Therefore,

$$|S_{ij}| \leq \max\left\{d_0, d \cdot \left(\frac{1}{2} + \frac{1}{\log n}\right)^{\log \ell}\right\} \leq \max\left\{d_0, d \cdot \left(\frac{1}{2}\right)^{\log \ell} \cdot \left(1 + \frac{2}{\log n}\right)^{\log \ell}\right\} \leq \max\left\{d_0, \frac{2d}{\ell}\right\}$$

Where the last inequality holds if $\epsilon \geq 2 \log\log n/(\ln 2 \log n)$ which can be assumed without loss of generality. ∎

The time complexity of this step is $O(\log \ell \log^3 n)$ and the number of processors required is $n^{1+1/\gamma}$.

6.4.2 Step 2

For each part D_j, in parallel: find an ϵ-good coloring in the set system restricted to the ground set D_j according to the following.

- If $\max_{1\leq i\leq n} |S_i \cap D_j| > d_0$, find the solution by applying Algorithm Bootstrap on ground set D_j and sets $S_{ij} = S_i \cap D_j$ for all $1 \leq i \leq n$
- Otherwise, decide (arbitrarily) that $D_j^0 = D_j$.

Let S_{ij}^k denote $S_{ij} \cap D_j^k$. In both cases,

$$\max_{1\leq i\leq n, 1\leq j\leq \ell} \left|\frac{|S_{ij}|}{2} - |S_{ij}^0|\right| \leq d_0 \cdot (2d/\ell)^{1/2+\epsilon}$$

The discrepancy of the solution is bounded by $d_0(2d/\ell)^{1/2+\epsilon}$.

6.4.3 Step 3

In this step, we merge the color classes computed for each part, so as to obtain a good coloring of the ground set. If the color classes that were obtained in the different parts are merged carelessly, the discrepancies might add up. Instead, we would like them to cancel each other. To do that, we can express the merging problem as a lattice approximation problem, and use Raghavan's [Ra] sequential algorithm.

Definition 6.2 *Let the* $(n+1)\times \ell$ *matrix* C *be defined as follows: for* $1\le i\le n$ *and* $1\le j\le \ell$

$$C_{i,j}=\frac{\frac{|S_{ij}|}{2}-|S^0_{ij}|}{2d_0\cdot(2d/\ell)^{1/2+\epsilon}}+\frac{1}{2}$$

and for $i=n+1, 1\le j\le \ell,\quad C_{n+1,j}=1.$

Each of the first n rows is the normalized discrepancy of the set S_i in all the parts D_j $(1\le j\le l)$. The role of the last row will become clear later.

For $1\le i\le n+1$, let $m_i=\sum_{j=1}^{\ell}C_{i,j}$. Since we know that $0\le C_{i,j}\le 1$, it follows that $0\le m_i\le \ell$.

A $(0,1)$ vector $\vec{q}=(q_1,q_2\ldots q_\ell)$ induces the merge: $B^0=\cup_{j=1}^{\ell}D_j^{q_j}$ and $B^1=\cup_{j=1}^{\ell}D_j^{1-q_j}$.

Suppose that we have a vector $\vec{q}$ and a fixed value y such that

$$\max_{1\le i\le n+1}|\sum_{j=1}^{\ell}q_i\cdot C_{i,j}-\frac{m_i}{2}|\le y$$

We will show that if y is small enough then B^0 and B^1 have discrepancy at most $d_0(2d/\ell)^{0.5+\epsilon}$.

For $1\le i\le n$,

$$\sum_{j=1}^{\ell}(2\cdot q_j-1)\cdot C_{i,j}=2\cdot\sum_{j=1}^{\ell}q_j\cdot C_{i,j}-m_i.$$

Therefore,

$$|\sum_{j=1}^{\ell}(2\cdot q_i-1)\cdot C_{i,j}|\le 2y$$

Note that

$$(2q_j-1)\frac{\frac{|S_{ij}|}{2}-|S^0_{ij}|}{2d_0\cdot(2d/\ell)^{1/2+\epsilon}}=\frac{\frac{|S_{ij}|}{2}-|S^{1-q_j}_{ij}|}{2d_0\cdot(2d/\ell)^{1/2+\epsilon}}$$

It follows that,

$$\begin{aligned}2y\ge\left|\sum_{j=1}^{\ell}(2q_j-1)\cdot C_{i,j}\right|&=\left|\sum_{j=1}^{\ell}(2q_j-1)\cdot\frac{\frac{|S_{ij}|}{2}-|S^0_{ij}|}{2d_0\cdot(2d/\ell)^{1/2+\epsilon}}+\frac{1}{2}\sum_{j=1}^{\ell}(2q_j-1)\right|\\&=\left|\sum_{j=1}^{\ell}\frac{\frac{|S_{ij}|}{2}-|S^{1-q_j}_{ij}|}{2d_0\cdot(2d/\ell)^{1/2+\epsilon}}+\sum_{j=1}^{\ell}q_j-\frac{\ell}{2}\right|=\left|\frac{\frac{|S_i|}{2}-|B^0\cap S_i|}{2d_0\cdot(2d/\ell)^{1/2+\epsilon}}+\sum_{j=1}^{\ell}q_i-\frac{\ell}{2}\right|.\end{aligned}$$

The $(n+1)^{th}$ row ensures that $|\sum_{j=1}^{\ell}q_i-\frac{\ell}{2}|\le y$. Therefore, we can conclude that

$$\left|\frac{|S_i|}{2}-|B^0\cap S_i|\right|\le 6yd_0\cdot(2d/\ell)^{1/2+\epsilon}$$

If $y\le 1/6(\ell/2)^{1/2+\epsilon}$, then the discrepancy of the above merge is at most $d_0\cdot d^{1/2+\epsilon}$.

Lemma 6.2 *If $\ell > \log^{1/2\epsilon} n$, there exists a vector $\vec{q} \in \{0,1\}^\ell$ such that*

$$\max_{1 \le i \le n+1} \left| \sum_{j=1}^{\ell} q_i \cdot C_{i,j} - \frac{m_i}{2} \right| \le y = \frac{1}{6}(\ell/2)^{1/2+\epsilon}$$

Furthermore, it can be found in $\ell \log n$ time using $n\ell$ processors.

Proof: Let $F_i = \sum_{j=1}^{\ell} q_j \cdot C_{i,j}$. If $\vec{q}$ is selected by choosing its entries uniformly at random, then from the generalized Chernoff bound in [Ra, Thm. 1], we can deduce that

$$\mathrm{Prob}[F_i > \frac{m_i}{2} + y] \le \left(\frac{e^{2y/m_i}}{(1+2y/m_i)^{(1+2y/m_i)}} \right)^{m_i/2} \le e^{-2y^2/m_i} \le e^{-2y^2/\ell} \le e^{-(\ell/2)^{2\epsilon}}.$$

Similarly,

$$\mathrm{Prob}[F_i < \frac{m_i}{2} - y] \le e^{-(\ell/2)^{2\epsilon}}.$$

We need these expressions to be less that $\frac{1}{2(n+1)}$. Hence, taking ℓ to be $\lceil 2 \log^{1/2\epsilon}(2n+1) \rceil$ suffices.

Raghavan [Ra] showed how to find a deterministic solution that matches the randomized one using the method of conditional probabilities. His solution requires ℓ steps, where in each step an entry in the vector $\vec{q}$ is computed. Each step involves the computation of an estimator function which can be evaluated by $\mathrm{O}(n\ell)$ processors in $\mathrm{O}(\log n)$ time. ∎

Theorem 6.4 *For any $\epsilon > 0$, an ϵ-good coloring can be found in $\mathrm{O}(\log^2 n \log^{1/2\epsilon} n)$ time using $\mathrm{O}(n^6)$ processors.*

Proof: The recursion depth is $\log n / \log \ell$. Step 3 is the most time consuming step, and hence the total time is $\ell \log^2 n / \log \ell \le \log^{2+1/2\epsilon} n$. There are at most n simultaneous calls to the algorithm during its execution where each requires $n^{1+1/\gamma}$ processors (Step 1). Therefore the total processor requirement does not exceed $n^{2+1/\gamma} = n^6$. ∎

7 The vector balancing problem

In this section we consider the following generalization of the set balancing problem. It turns out that this problem is useful for the lattice approximation problem.

The Vector Balancing Problem

Input: An $n \times m$ matrix A with each entry $A_{ij} \in [0,1]$. Let $\vec{h}$ be an m-vector with each $h_j = 1/2$ and $\vec{c} = A \cdot \vec{h}$.
Output: An m-bit vector $\vec{q} \in \{0,1\}^m$ such that it minimizes the discrepancy $\|\vec{\Delta}\|_\infty$, where $\vec{\Delta} = |A \cdot \vec{q} - \vec{c}| = |A \cdot (\vec{q} - \vec{h})|$.

This problem requires a partition (or, 2-coloring) of the columns of the matrix A into two new matrices A_0 and A_1 such that the corresponding row sums of the new matrices are close to each other. Notice that the set balancing problem is the special case where the matrix A is the incidence matrix of the input set family, with the rows corresponding to the subsets and the columns to the elements of the base set. In the **weighted set balancing problem**, we associate a weight $w_j \in [0,1]$ with each base element $b_j \in B$; the problem is to find a 2-coloring which partitions each set into two near-equal **weight** subsets, rather than just near-equal **cardinality** subsets. The weighted set balancing problem is the special case of the vector balancing problem where all the non-zero entries in a column of A are identical.

The sequential version of this problem was studied earlier by Spencer [Sp, Sp2]. It serves as a bridge between the set balancing algorithm and the lattice approximation problem. In this section we will present an NC algorithm for this problem by showing that it is reducible to the set balancing problem. Later, we will solve the lattice approximation problem by providing a reduction to the vector balancing.

7.1 Reduction to set balancing

In reducing the vector balancing problem to the set balancing problem, we need to assume that each entry in the matrix A is a real number that is represented to $L = \mathrm{O}(\log n)$ bits of precision only. We may assume this without any loss of generality due to the following observation. Suppose the entries in A were arbitrary real numbers. Let $m \leq n^\beta$, for some constant β, and let $L = \beta \log n$. Further, let A'_{ij} denote the entry A_{ij} truncated to L bits of precision and set $\vec{d} = A' \cdot \vec{h}$.

Proposition 7.1 *Let $\vec{q} \in [0,1]^m$ be any vector. Then, the discrepancy between $A \cdot \vec{q}$ and $A' \cdot \vec{q}$, given by $\vec{\delta} = |(A - A') \cdot \vec{q}|$, satisfies*

$$||\vec{\delta}||_\infty = \mathrm{O}(1)$$

This proposition may be verified by noting that

$$|\delta_i| = |\sum_{j=1}^{m} (A_{ij} - A'_{ij}) \cdot q_j| \leq m \cdot \frac{1}{m} \leq 1$$

It implies that any good balancing solution for the matrix A' is also a good balancing solution for the matrix A. Thus, we will assume here onwards that each entry in the matrix is specified to exactly L bits of precision.

Definition 7.1 *For $1 \leq k \leq L$, let $A_{ij}^{(k)}$ denote the k^{th} most significant bit in the binary representation of each A_{ij}. Since each A_{ij} lies in the interval $[0,1]$, we have that*

$$A_{ij} = \sum_{k=1}^{L} 2^{-k} \cdot A_{ij}^{(k)}$$

In order to reduce the vector balancing problem to the set balancing problem, we construct a new matrix B with Ln rows and m columns such that each $B_{ij} \in \{0,1\}$. The new matrix is obtained by representing each A_{ij} as a column vector with L bits, these L bits are the binary representation of A_{ij}. Substituting each A_{ij} by such a column vector gives us the matrix B which represents an instance of the (unweighted) set balancing problem.

Definition 7.2 *For any instance A of the vector balancing problem, let B be a 0-1 matrix with Ln rows and m columns such that, for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq L$ and $r = (i-1)L + k$,*

$$B_{rj} = A_{ij}^{(k)}$$

Further, define the (Ln)-vector $\vec{d} = B \cdot \vec{h}$ with $\vec{h}$ representing the all-halves vector as before.

The following proposition relates the vector $\vec{c}$ with $\vec{d}$ and will be useful in the later analysis

Proposition 7.2 *For each $1 \leq i \leq n$, $c_i = \sum_{k=1}^{L} 2^{-k} \cdot d_{(i-1)L+k}$.*

Proof: It follows from the above definitions that

$$\begin{aligned} c_i &= \sum_{j=1}^{m} A_{ij} \cdot \frac{1}{2} = \sum_{j=1}^{m} \sum_{k=1}^{L} 2^{-k} \cdot A_{ij}^{(k)} \cdot \frac{1}{2} = \sum_{k=1}^{L} 2^{-k} \sum_{j=1}^{m} A_{ij}^{(k)} \cdot \frac{1}{2} = \\ &= \sum_{k=1}^{L} 2^{-k} \sum_{j=1}^{m} B_{\{(i-1)L+k\}j} \cdot \frac{1}{2} = \sum_{k=1}^{L} 2^{-k} \cdot d_{(i-1)L+k} \end{aligned}$$

∎

In the next section we will show that, for any $0 < \epsilon < \frac{1}{2}$, an ϵ-good balancing vector (or, 2-coloring) for the matrix B is also a good balancing vector for the matrix A. Thus, to solve the vector balancing problem all we need to do is to construct the matrix B, which can easily be done in NC, and then invoke the NC algorithm for the set balancing problem (see Section 6.3). Note that we have increased the number of rows from n to Ln in going from A to B but this presents no problems since $L = \mathrm{O}(\log n)$.

7.2 Analysis of the reduction process

Suppose we obtain an ϵ-good solution vector $\vec{q} \in \{0,1\}^m$ for the set balancing problem involving the matrix B. Let the vector $\vec{\delta} = B \cdot \vec{q} - \vec{d}$ denote the discrepancy vector for this solution. Also, let $\vec{\Delta} = A \cdot \vec{q} - \vec{c}$ denote the discrepancy vector when we use $\vec{q}$ as the balancing solution for the original matrix A.

Definition 7.3 *A balancing vector $\vec{q}$ is called an ϵ-good solution for the vector balancing problem with the matrix A if, for each $1 \leq i \leq n$*

$$|\Delta_i| = \mathrm{O}(c_i^{\frac{1}{2}+\epsilon}\sqrt{\log n})$$

We conclude the analysis by showing that if $\vec{q}$ is an ϵ-good solution to the set balancing problem B, then it is also an ϵ-good solution to the vector balancing problem A.

Theorem 7.1 *The vector $\vec{q}$ is an ϵ-good solution for the vector balancing problem A.*

Proof: We will show that, for each $1 \leq i \leq n$, the value of $|\Delta_i|$ is suitably bounded. Consider any row i in the matrix A.

$$|\Delta_i| = |c_i - \sum_{j=1}^{m} A_{ij} \cdot q_j|$$

Using Definition 7.1, this may be rewritten as,

$$|c_i - \sum_{j=1}^{m}\sum_{k=1}^{L} 2^{-k} \cdot A_{ij}^{(k)} \cdot q_j|$$

Proposition 7.2 and Definition 7.2 together give us that this is equal to,

$$|\sum_{k=1}^{L} 2^{-k} d_{(i-1)L+k} - \sum_{k=1}^{L} 2^{-k} \sum_{j=1}^{m} B_{\{(i-1)L+k\}j} \cdot q_j|$$

which, using the definition of the vector $\vec{\delta}$, may be rewritten as,

$$|\sum_{k=1}^{L} 2^{-k} \cdot \delta_{(i-1)L+k}|.$$

The absolute value of a sum is bounded by the sum of the absolute values of its summands,

$$|\Delta_i| \leq \sum_{k=1}^{L} 2^{-k} \cdot |\delta_{(i-1)L+k}|$$

Theorem 6.3 now gives us the bound,

$$\mathrm{O}\left(\sum_{k=1}^{L} 2^{-k} d_{(i-1)L+k}^{\frac{1}{2}+\epsilon}\sqrt{\log n}\right)$$

Using the fact that a decreasing geometric sum is bounded, this may be rewritten as,

$$\mathrm{O}\left(\sqrt{\log n} \cdot \sum_{k=1}^{L} (2^{\frac{1}{2}-\epsilon})^{-k} \cdot (2^{-k} \cdot d_{(i-1)L+k})^{\frac{1}{2}+\epsilon}\right)$$

Proposition 7.2 bounds each term of the summation by c_i and so, $2^{-k} \cdot d_{(i-1)L+k} \leq c_i$. Hence, the bound

$$\mathrm{O}\left(\sqrt{\log n} \cdot \sum_{k=1}^{L} (2^{\frac{1}{2}-\epsilon})^{-k} \cdot c_i^{\frac{1}{2}+\epsilon}\right)$$

Again, the sum of a decreasing geometric series is bounded and we have the desired result.

$$|\Delta_i| = \mathrm{O}\left(c_i^{\frac{1}{2}+\epsilon}\sqrt{\log n}\right)$$

∎

8 Lattice approximation and randomized rounding

In this section we show how to reduce the lattice approximation problem to the vector balancing problem. To meet this goal a rounding technique, called bit-by-bit rounding is used. This rounding technique can be efficiently implemented in RNC and can also be de-randomized using the parallelized version of the method of conditional probabilities. This is essentially the same as the method used by Beck & Fiala [BF] (see also [Be, Sp]), although our bounds are better, i.e., they depend on c_i and not on n. It is also similar to various scaling methods.

Raghavan uses randomized rounding to provide a randomized algorithm that solves the lattice approximation problem. This algorithm was then de-randomized using the method of conditional probabilities. His rounding is done by setting the value of each q_i to 1 with probability p_i and to 0 with probability $1 - p_i$. The method of conditional probabilities can then be applied to setting the bits in the vector q one by one.

Unfortunately the analysis of Section 6.1 does not seem to carry over when the probabilities are not all halves. Therefore, we present the bit-by-bit randomized rounding which yields the same bounds as Raghavan's randomized algorithm. Its advantage is that it can be de-randomized by a reduction to the vector balancing problem and give a discrepancy which is slightly larger than that guaranteed by Raghavan's algorithm.

8.1 Bit-by-bit randomized rounding

As in the previous section, we first make the assumption that each p_i is represented by L bits, where $L = \mathrm{O}(\log n)$. Since each $p_i \in [0, 1]$, this implies that the k^{th} bit of p_i contributes 2^{-k} to its value. This assumption is not at all restrictive since if we truncate each p_i to L bits of precision, we introduce only a small error in the product with A. More precisely, let the p_i's be specified with arbitrary precision and let P_i's denote their truncation to exactly L bits. The next proposition is very similar to Proposition 7.1.

Proposition 8.1 *Let $\vec{d} = A \cdot \vec{P}$ and let $m \leq n^\beta$ for some fixed β. If $L \geq \beta \log n$ then the discrepancy between $A \cdot \vec{p}$ and $A \cdot \vec{P}$ is bounded as follows.*

$$|||\vec{d} - \vec{c}|||_\infty \;=\; |||A \cdot (\vec{p} - \vec{P})|||_\infty \;\leq\; 1$$

In the sequel we will assume that the vector $\vec{p}$ is specified with L bits of precision only. Since each $p_j \in [0,1]$, we will further assume that p_j is represented by the bits $p_j^{(0)}$, $p_j^{(1)}$, $p_j^{(2)}$, $\cdots$, $p_j^{(L)}$. Here $p_j^{(L)}$ represents the least significant bit and $p_j^{(0)}$ is non-zero if and only if $p_j = 1$. Thus,

$$p_j \;=\; \sum_{k=0}^{L} 2^{-k} p_j^{(k)}$$

The bit-by-bit randomized rounding works in L stages. At each stage it "rounds" the least significant bit of each p_j, giving them new values which can be represented with one less bit of precision. It is important to note that the rounding process removes the least significant bit of each p_j while possibly modifying all the other bits in its representation. After L stages of rounding, we will be left with only the bits $p_1^{(0)}$, $p_2^{(0)}$, $p_3^{(0)}$, $\cdots$, $p_m^{(0)}$ and these can only represent values which are either 0 or 1. We then choose $q_j = p_j^{(0)}$, for each j, and obtain a lattice point which, as we will show, gives a small discrepancy with a non-zero probability.

It only remains to specify how the least significant bits are rounded at each stage of the above process. Assume that we have completed $L - k$ stages and that we now have each p_j represented with exactly k bits of precision. At the current stage we are required to "round" the least significant bits which are $p_1^{(k)}$, $p_2^{(k)}$, $p_3^{(k)}$, $\cdots$, $p_m^{(k)}$. The randomized rounding of these bits works as follows: if $p_j^{(k)} = 0$ then we simply omit this bit without creating any discrepancy; on the other hand, if $p_j^{(k)} = 1$, we will **round up** or **round down** this bit with equal probability. Rounding down is equivalent to setting the bit $p_j^{(k)}$ to 0, and then we can drop this bit. It is clear that rounding down is also equivalent to subtracting 2^{-k} from p_j. Rounding up is equivalent to adding 2^{-k} to the value of p_j; this will cause $p_j^{(k)}$ to become 0 and then we can omit the bit altogether. It is not very hard to see that the bit-by-bit randomized rounding will always maintain the property that each $p_j \in [0,1]$ and that at the end of the process we will have that each $p_j \in \{0,1\}$. The following lemma shows that the bounds on the discrepancy obtained by the randomized bit-by-bit rounding technique are exactly equal to those obtained by Raghavan's algorithm.

Lemma 8.1 *Let the vector $\vec{p}$ be specified with L bits of precision. For $1 \leq j \leq m$, the probability of rounding the j^{th} entry in $\vec{p}$ to 1 is equal to p_j.*

Proof: The proof is by induction on L, the number of precision bits. It is obviously true for $L = 1$ and assume it is true for $L \leq k - 1$.

If the bit $p_j^{(k)}$ was rounded up, then the new value of p_j is equal to

$$p_j^{\text{up}} = p_j + 2^{-k} \cdot p_j^{(k)}$$

On the other hand, if it is rounded down, then the new value of p_j is equal to

$$p_j^{\text{down}} = p_j - 2^{-k} \cdot p_j^{(k)}$$

Both p_j^{up} and p_j^{down} are specified by $L-1$ bits. Hence by the induction hypothesis,

$$\text{Prob}[p_j^{\text{up}} \text{ is rounded to } 1] = p_j^{\text{up}}$$

and

$$\text{Prob}[p_j^{\text{down}} \text{ is rounded to } 1] = p_j^{\text{down}}$$

Therefore,

$$\begin{aligned}\text{Prob}[p_j \text{ is rounded to } 1] &= \frac{1}{2}\text{Prob}[p_j^{\text{up}} \text{ is rounded to } 1] \\ &+ \frac{1}{2}\text{Prob}[p_j^{\text{down}} \text{ is rounded to } 1] = p_j\end{aligned}$$

∎

8.2 Deterministic rounding

To de-randomize the above algorithm, we view each rounding step as choosing a random solution to a vector balancing problem. Using the results of Section 7, we can obtain a deterministic solution to this vector balancing problem. Yet, we must establish that the sum of the discrepancies introduced at each stage of rounding is sufficiently small. We first present the deterministic lattice approximation algorithm and then bound the discrepancy introduced by it. Let $\vec{p}^{(l)}$ denote the vector $(p_1^{(l)}, p_2^{(l)}, \ldots, p_m^{(l)})$. The precision of the algorithm denotes the number of bits with which each entry in $\vec{p}$ is represented. Note that in this algorithm we will invoke the vector balancing algorithm for a matrix A and a vector $\vec{p}$ whose entries are either 0 or $\frac{1}{2}$. Such instances can also be solved by the algorithm from the previous section (see Remark 7.1).

The Lattice Approximation Algorithm
Input: matrix A and vector $\vec{p}$ with precision l.
Output: rounded vector $\vec{q}$.

1. If $l = 1$, then $\vec{q}$ is a solution to the vector balancing problem with the matrix A and the vector $\vec{p}$.

2. Else,

   (a) Solve the vector balancing problem with the matrix A and the vector $\frac{1}{2}\vec{p}^{(l)}$. Let the solution be $\vec{q}^{(l)}$.

(b) The output vector $\vec{q}$ is the solution to the following lattice approximation problem: matrix A and vector $\vec{P} = \vec{p} + 2^{-(l-1)} \cdot (\vec{q}^{(l)} - \frac{1}{2}\vec{p}^{(l)})$ with precision $l-1$. This solution can be computed by a recursive call to this algorithm.

It only remains to establish a bound on the discrepancy of the solution obtained by this bit-by-bit deterministic rounding procedure.

Definition 8.1 *For $0 \leq k \leq L$ and for $1 \leq i \leq m$, let $P_i^{\{k\}}$ denote the rounded value of p_i at the end of the k^{th} stage of rounding. Also, let $\vec{C}^{\{k\}} = A \cdot \vec{P}^{\{k\}}$. Considering the cases $k = 0$ and $k = L$, following special cases.*

1. *The initial conditions are: $\vec{P}^{\{0\}} = \vec{p}$ and $\vec{C}^{\{0\}} = A \cdot \vec{P}^{\{0\}} = \vec{c}$.*
2. *The final output of the rounding procedure is: $\vec{q} = \vec{P}^{\{L\}}$ with $A \cdot \vec{q} = \vec{C}^{\{L\}}$.*

This means that each $P_i^{\{k\}}$ is represented with $L-k$ bits of precision. Further, $\vec{C}^{\{k\}}$ denotes the approximation to the vector $\vec{c} = A \cdot \vec{p}$ at the end of the k^{th} stage of rounding. We now define the k^{th} stage discrepancy as the extra discrepancy introduced at the k^{th} stage of rounding.

Definition 8.2 *The k^{th} stage discrepancy vector is defined as $\vec{D}^{\{k\}} = |\vec{C}^{\{k\}} - \vec{C}^{\{k-1\}}|$.*

In going from $\vec{P}^{\{k\}}$ to $\vec{P}^{\{k+1\}}$ we are solving a vector balancing problem with respect to the least significant bits, the $(L-k)^{th}$ bits, of the vector $\vec{P}^{\{k\}}$. It then follows that the L_∞-norm of the stage discrepancy vector $\vec{D}^{\{k\}}$ must be small. Applying Theorem 7.1 from the previous section to the k^{th} stage discrepancy, we obtain the following lemma.

Lemma 8.2 *For $1 \leq i \leq n$ and $1 \leq k \leq L$, we have that*

$$D_i^{\{k\}} = |C_i^{\{k\}} - C_i^{\{k-1\}}| = \beta \cdot \frac{\left(C_i^{\{k-1\}}\right)^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}}{(2^{\frac{1}{2}-\epsilon})^{L-k}}$$

where β is some positive constant.

Proof: Let $l = L - k + 1$ and let $\vec{p}^{(l)}$ be the vector of the least significant bits of the elements of the vector $\vec{P}^{\{k-1\}}$. The k^{th} stage of the lattice approximation algorithm computes a vector $\vec{q}^{(l)}$ which is an ϵ-good solution to the vector balancing problem with the matrix A and the vector $\frac{1}{2} \cdot \vec{p}^{(l)}$. This implies that,

$$\vec{P}^{\{k\}} = \vec{P}^{\{k-1\}} + 2^{-(l-1)} \cdot (\vec{q}^{(l)} - \frac{1}{2} \cdot \vec{p}^{(l)})$$

From this conclude that,

$$\vec{D}^{\{k\}} = 2^{-(l-1)} \cdot |A \cdot (\vec{q}^{(l)} - \frac{1}{2} \cdot \vec{p}^{(l)})|$$

Noting that the i^{th} entry in the vector $A \cdot \vec{p}^{(l)}$ cannot exceed $2^l \cdot C_i^{\{k-1\}}$, we obtain the following bound from Theorem 7.1.

$$D_i^{\{k\}} = \mathrm{O}\left(\frac{\left(2^l C_i^{\{k-1\}}\right)^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}}{2^{(l-1)}}\right)$$

Upon rearrangement, this gives the desired bound for some suitable constant β. ∎

The above lemma establishes that the amount of discrepancy introduced at each stage is suitably bounded. We now establish that the net discrepancy produced by the solution vector $\vec{q}$ is also extremely small.

Lemma 8.3 *The bit-by-bit rounding procedure produces a solution vector $\vec{q}$ with the discrepancy vector $\vec{\Delta} = |A \cdot (\vec{p} - \vec{q})|$ such that, for $1 \le i \le n$,*

$$\Delta_i = \mathrm{O}\left(c_i^{\frac{1}{2}+\epsilon}\sqrt{\log n} + \log^{\frac{1}{1-2\epsilon}} n\right)$$

Proof: Assume, for the moment, that we are considering a row i such that $c_i = \Omega\left(\log^{\frac{1}{1-2\epsilon}} n\right)$. Consider any stage k such that before this stage $C_i^{\{k-1\}}$ satisfies the following inequality (1), for some positive constant α.

$$C_i^{\{k-1\}} \le c_i + \alpha \cdot c_i^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n} \tag{1}$$

Using Lemma 8.2, we then obtain the following bound on the discrepancy $D_i^{\{k\}}$ introduced at this stage.

$$\begin{aligned}
D_i^{\{k\}} &= |C_i^{\{k\}} - C_i^{\{k-1\}}| \\
&\le \beta \cdot \frac{\left(C_i^{\{k-1\}}\right)^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}}{(2^{\frac{1}{2}-\epsilon})^{L-k}} \\
&\le \beta \cdot \frac{\left(c_i + \alpha \cdot c_i^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}\right)^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}}{(2^{\frac{1}{2}-\epsilon})^{L-k}} \\
&\le \frac{2\beta \cdot c_i^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}}{(2^{\frac{1}{2}-\epsilon})^{L-k}}
\end{aligned}$$

The last inequality follows from the assumption that $c_i = \Omega\left(\log^{\frac{1}{1-2\epsilon}} n\right)$.

We now have that, when $C_i^{\{k-1\}}$ is bounded as in (1), the discrepancy introduced at stage k is bounded as above in terms of $C_i^{\{0\}} = c_i$. Let r be such that at each stage $k \le r$ the value $C_i^{\{k\}}$

satisfies the inequality (1). It then follows from the definition of the stage discrepancy that,

$$\begin{aligned} C_i^{\{r\}} &\leq c_i + \sum_{k=1}^{r} D_i^{\{k\}} \\ &\leq c_i + \sum_{k=1}^{r} \frac{2\beta \cdot c_i^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}}{(2^{\frac{1}{2}-\epsilon})^{L-k}} \\ &= c_i + c_i^{\frac{1}{2}+\epsilon}\sqrt{\log n} \cdot \sum_{k=1}^{r} \frac{2\beta}{(2^{\frac{1}{2}-\epsilon})^{L-k}} \end{aligned}$$

A straightforward induction on the value of r now shows that (1) is satisfied for all $k \leq L$ provided we choose α such that

$$\begin{aligned} \alpha &\geq \sum_{j=1}^{L} \frac{2\beta}{(2^{\frac{1}{2}-\epsilon})^{L-j}} \\ &= 2\beta \cdot \left(\frac{2^{\frac{1}{2}-\epsilon}}{2^{\frac{1}{2}-\epsilon} - 1} \right) + \mathrm{o}(1) \end{aligned}$$

Such a choice of α gives us the desired result that

$$\Delta_i = |C_i^{\{L\}} - C_i^{\{0\}}| \leq \alpha \cdot c_i^{\frac{1}{2}+\epsilon} \cdot \sqrt{\log n}$$

Finally, we have to complete the analysis for the case where $c_i = \mathrm{o}\left(\log^{\frac{1}{1-2\epsilon}} n\right)$. In the initial stages of the rounding process, the discrepancy can be extremely high (with respect to c_i) for such rows i. However, once the value of $C_i^{\{k\}}$, for some k, becomes $\Omega\left(\log^{\frac{1}{1-2\epsilon}} n\right)$, the preceding analysis applies and we have that the discrepancy introduced in the subsequent stages cannot exceed $\mathrm{O}\left(\log^{\frac{1}{1-2\epsilon}} n\right)$. Thus, even when c_i is small, the net discrepancy is bounded as stated in the lemma. ▮

These results can now be summarized as follows.

Theorem 8.1 *The bit-by-bit rounding procedure gives an NC algorithm for the lattice approximation problem which produces a solution $\vec{q}$ with the discrepancy vector $\vec{\Delta} = |A \cdot (\vec{p} - \vec{q})|$ such that, for $1 \leq i \leq n$,*

$$\Delta_i = \mathrm{O}\left(c_i^{\frac{1}{2}+\epsilon}\sqrt{\log n} + \log^{\frac{1}{1-2\epsilon}} n \right)$$

Proof: In Lemma 8.3 we have already shown that the discrepancy of the solution is suitably bounded. The lattice approximation algorithm is essentially L invocations of the NC algorithm for vector balancing. Since $L = \mathrm{O}(\log n)$, it follows that the entire lattice approximation algorithm also runs in NC. ▮

9 Edge coloring in NC

Let $G = (V, E)$ be an undirected graph whose maximal degree is Δ. A **legal edge coloring** is an assignment of colors to the edges such that two adjacent edges cannot have the same color. Vizing's Theorem [Vi] states that every graph can be edge colored by at most $\Delta + 1$ colors and hence χ', the chromatic index of G, is either Δ or $\Delta + 1$. Determining the exact value of χ' was shown to be NP-complete in cubic graphs by Holyer [Ho] and extended later by Galil & Leven [GL] to any Δ-regular graph. Algorithmically, Vizing's proof also implies a polynomial time algorithm for obtaining a $\Delta + 1$ coloring.

Obtaining a $\Delta + 1$ edge coloring in a general graph has long been an open problem in parallel computation. Partial results were achieved by Karloff & Shmoys [KS]. They gave an NC algorithm for $\Delta+1$ edge coloring when $\Delta \le O(\log^{O(1)} n)$. For the general case, they gave an NC algorithm that colors the edges with at most 1.5Δ colors. This latter result can be improved to $\Delta + \frac{\Delta}{O(\log^{O(1)} n)}$ by combining the methods of Arjomandi [Ar] with those of [KS]. It should also be noted that bipartite graphs can always be edge colored optimally in NC using Δ colors, as described by Lev, Pippenger & Valiant [LPV].

Karloff & Shmoys [KS] have also exhibited an RNC algorithm which edge colors a graph with $\Delta + O(\Delta^{0.5+\epsilon})$ colors. This algorithm uses a divide-and-conquer approach where the divide step is essentially the solution to a set balancing problem. Since this is the only segment of the algorithm which is randomized, the entire algorithm can be de-randomized by our results. Assume the vertices of the graph are labeled by the two labels, 0 and 1, and let G_0 and G_1 be the induced graphs of the vertices labeled 0 and 1 respectively. Let G_{01} denote the bipartite graph such that the vertices in each part have the same label. The randomized algorithm is as follows:

1. Label the vertices uniformly at random by 0 and 1.
2. (a) Recursively, color the edges of G_0 and G_1 using the same colors.
   (b) Color the edges of the bipartite graph G_{01} optimally [LPV].

The algorithm achieves its best bounds when the maximum degree of G_0, G_1 and G_{01} is minimized. It is not hard to see that minimizing the maximum degree in the above three graphs is an instance of the set balancing problem. Hence, the algorithm of Karloff & Shmoys can be de-randomized and we get the following theorem. Notice that if $\Delta < O(\log^{\frac{1}{\epsilon}})$, then the graph can be colored optimally by $\Delta + 1$ colors [KS]. Thus, we apply the set balancing algorithm described in Theorem 6.2 to partition the graph, recursively, until the maximum degree becomes too small. Since the discrepancy at each stage of the recursion is bounded as in Theorem 6.2, we obtain the following result.

Theorem 9.1 *There is an NC algorithm which colors the edges of a simple graph with $\Delta + O(\Delta^{0.5+\epsilon})$ colors for $\epsilon > 0$. The number of processors is independent of ϵ.*

10 Applications to computational geometry

Random sampling is a common method used in randomized algorithms in computational geometry. This method was first introduced by Clarkson [Cl1] and has since found applications in numerous papers, for example [Cl2, Cl3, CS, HW, RS]. Chazelle & Friedman [CF] have shown that in many instances, the sample needed for this method can be abstracted in terms of computing a certain vertex cover in a hypergraph. Moreover, they showed how to find this vertex cover deterministically and as a result, many algorithms that use random sampling have now become deterministic. In this section we show how the results of [CF] can be obtained in NC as well. The reader is warned that this section is not self contained and is referred to the paper of [CF] as our description relies heavily on it. Anderson [An] was the first to notice the connection between set balancing and the hypergraph covering problems. Consequent to our results, [BRS] provided more efficient NC algorithms for these problems.

We follow the notation of [CF]. A hypergraph $H = (V, E)$ contains a finite set V of vertices of cardinality n and an edge set E of nonempty subsets of V. A subset $T \subset V$ of cardinality r is called an r-**sample**. A **frame** $\mathcal{F}$ is a pair $(H; \varphi)$ where H is a hypergraph and φ is a homomorphism from the semilattice $(2^V, \cap)$ to $(2^E, \cap)$ in which $\varphi(V) = E$. The frame is said to be of dimension d, if d is a positive real constant and, for each $W \subseteq V$, the size of $\{W \cap e | e \in \varphi(W)\}$ is at most $c|W|^d$ for some constant c. The ratio $\min\{|e|/|V| : e \in E\}$ is called the **threshold** of the frame. An r-sample R is an r-**cover** for the frame if it has a nonempty intersection with every edge of $\varphi(R)$. The main theorem of [CF] on frames is the following:

Theorem 10.1 *Let $\mathcal{F}$ be a frame of dimension d with n vertices and let $r \leq n$ be any integer than some constant. If the threshold of $\mathcal{F}$ is at least $c(\log r)/r$, for some appropriate constant c, then it is possible to find an r-cover R for the frame in $O(rn^{d+1})$ time.*

A multi-hypergraph $H = (V, E)$ has **bounded vertex dependency** if each edge is associated with a nonempty subset $\sigma(e) \in V$ called its **signature**. Furthermore, for each edge e and any $W \subseteq V$ we have $|\sigma(e)| \leq d$ and $|\sigma^{-1}(W)| \leq c$ for some constants c and d. The domain of a sample R, denoted $H(R)$, is the multiset of edges whose signatures are subsets of R. An edge e is called t-**deficient** if: (i) $e \in H(R)$, (ii) $R \cap e = \emptyset$, and (iii) $|e| \geq tn/r$. The following theorem of [CF] is a special case of Theorem 10.1.

Theorem 10.2 *Given an n-vertex multi-hypergraph H of vertex dependency d and any integer $r \leq n$ larger than some constant, it is possible to compute an r-sample that leaves no edge of H*

$(c \log r)$-deficient, for some constant c. The running time is $O(rn^{d+1})$.

The **Vapnik-Chervonenkis dimension (VC)** of a hypergraph H is defined as the maximum size of any $W \subseteq V$ such that $\{W \cap e | e \in E\} = 2^W$. Given an r-sample $R \subseteq V$, an edge is t-**deficient** if: (i) $R \cap e = \emptyset$, and (ii) $|e| \geq tn/r$. The next theorem of [CF] is also a special case of Theorem 10.1.

Theorem 10.3 *Given an n-vertex hypergraph of VC-dimension d and any integer $r \leq n$ larger than some constant, it is possible to compute an r-sample that leaves no edge of the hypergraph $(c \log r)$-deficient, for some constant c. The running time is $O(rn^{d+1})$.*

We start by showing how the algorithm presented in [CF, Section 3] can be implemented in NC when only limited independence is available. This will provide us with an NC algorithm for Theorem 10.1. Let us now consider H as an arbitrary hypergraph with $O(n^d)$ edges without any reference to a frame. Let α be a real (not necessarily a constant) such that $0 \leq \alpha \leq 1$, and assume that each edge of H contains at least $m = \alpha n$ vertices.

Our first goal is to find an r-sample that intersects each edge e in at least $\Omega(|e|r/n)$ vertices. Lovàsz [Lo] showed that a greedy algorithm can compute a sample of size $c(\log n)/\alpha$ that intersects each edge at least once for some constant $c > 0$. Notice that a random sample of the same size for a large enough constant c would work just as well as Lovàsz's greedy method. Moreover, by a proper setting of c, an intersection of size $|e|r/n$ can be obtained. Anderson [An] showed that another way of computing an r-cover of size $c(\log n)/\alpha$ is by repeatedly invoking a set balancing subroutine. (In a similar way to the procedure outlined in Section 6.3.1). Combining this with the results of the previous section yields the following.

Lemma 10.1 *Let $c > 0$ be a suitable constant. An r-cover can be computed in NC.*

The last lemma is in fact equivalent to the Theorem 3.3 of [CF]. To obtain Theorem 10.1 we follow the same algorithm as [CF]. Three samples $R_1 \supseteq R_2 \supseteq R_3$ are computed and R is derived from them. The reader is referred to [CF, Section 3] for the precise details which can be easily implemented in NC.

11 Further Applications

In this sections we briefly outline some of the applications of our work to problems in combinatorics. The following sections briefly outline how we can take probabilistic proofs of existence of combinatorial objects and turn them into NC algorithms.

11.1 Constructing Ramsey graphs

One of the first applications of the probabilistic method was to show the existence of a graph with no large clique and no independent set. It has been an outstanding open problem to give explicit construction for such graphs. Our techniques yield a polylogarithmic time algorithm to construct such a graph using $n^{\log n}$ processors. Consider the analysis given in [Sp, ch. 1]. In a random graph where edges are selected with uniform probability to be in the graph, for each set S of k nodes we can define A_S to be the event that S is a clique or an independent set. If k is large enough ($= 2\log n$), then $\sum_S Pr[A_S] < 1$. The same analysis works if the choices of the edges are made only k^2-wise independently. Furthermore, $\sum_S Pr[A_S]$ can be used as a pessimistic estimator, and using the techniques of Section 4.2 it can be evaluated in polylogarithmic time, if we are provided with $n^{\log n}$ processors. Recently, a different construction of Ramsey graphs that uses limited independence was suggested in [Na].

11.2 Independent sets in sparse hypergraphs

Alon, Babai & Itai [ABI] provided an algorithm for finding a large independent set in a d-uniform hypergraph $H = (V, E)$. (In a d-uniform hypergraph, each contains exactly d vertices). A set $W \subseteq V$ is independent if W contains no edge. Their construction [ABI, Prop. 7.1] holds when d is bounded by a constant. We can handle the case where the value of d is polylogarithmic in n, the number of vertices.

Theorem 11.1 *Let $k = c(n^d/|E|)^{1/(d-1)}$ for a suitable constant c. Then finding in the hypergraph H an independent set W of cardinality at least k where d is polylogarithmic in n is in NC.*

Proof: In the proof of [ABI, Prop. 7.1], the vertices are chosen to W at random (d-wise independent) with probability $p = (3k)/n$. The proof proceeds by showing that the expected number of vertices chosen to W is at least $2k$; the expected number of edges completely contained in W is no more than k. Hence, by removing at most k vertices from W, we obtain an independent set of cardinality k. As the probabilistic scheme is based on computing expectations, it can be very easily de-randomized by our methods.

Let C_i be the indicator random variable of the event $v_i \in W$ where $v_i \in V$, and let $e = v_1, v_2, \ldots, v_d$ be a typical edge in H. Two conditional probabilities have to be estimated when the random bits are set one-by-one: (i) the event $e \in W$, i.e. $C_1 = 1, C_2 = 1, \ldots, C_d = 1$; (ii) the cardinality of the set W is at least $2k$. Both can be estimated as described in Section 5.2.

To conclude the proof, we explain how to generate $\{0,1\}$ random variables which are not uniformly distributed. Let $\ell = \lceil \log \frac{n}{3k} \rceil$. We associate a $\{0,1\}$ vector $\vec{x} = (x_1, \ldots x_\ell)$ with each random variable. Each entry in the vector x is uniformly distributed; a vector H_i and a random vector r_i are associated with it. The value of $\vec{x}$ determines the value of each random variable C_i. Assume that the random variable C_i is set to 1 if all the entries in $\vec{x}$ are equal to 1. The error incurred by rounding p to the nearest power of 2 affects only constant factors. In the de-randomization, we set the entries of $\vec{x}$ one-by-one in an arbitrary order. A detailed example of de-randomizing biased $\{0,1\}$ random variables is shown in [Lu2]. ∎

12 Discussion and Open Problems

A natural open problem that this paper suggests is obtaining the same discrepancy in the parallel case as in the sequential case, i.e., getting rid of the dependence on ϵ. A step in this direction is made in Section 6.4 by the Bootstrapping Algorithm where, instead of the number of processors, it is the running time that depends on ϵ. Is it possible to achieve a constructive version of Spencer's result that a discrepancy of $O(\sqrt{n})$ can always be guaranteed for the basic set balancing problem? Recently, [NN] suggested a different approach to the set balancing problem using small bias probability spaces. They show that Lemma 6.1 holds not only in the case where the partitioning is k-wise independent, but also when it is *almost* k-wise independent.

An outstanding open problem in the area of de-randomizing parallel algorithms is that of devising an NC algorithm for the problem of finding a maximum matching in a graph. This problem is known to have an RNC algorithm [KUW1, MVV]. Recently, Karger & Motwani [KM] de-randomized an RNC algorithm for the problem of finding a minimum cut in a (weighted) graph. This is closely related to the problem of finding a minimum s-t cut, which in turn can be used to solve the problem of finding a maximum matching. It remains an interesting open problem to show that maximum matching is in NC.

Acknowledgement

We would like to thank the anonymous referees for carefully reading the paper and making useful suggestions.

References

[Ad] L. Adleman, "Two theorems on random polynomial time," *19th Annual Symposium on Foundations of Computer Science*, 1978, pp. 75-83.

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

[An] R. Anderson, "Set intersection problems," *Unpublished manuscript* (1985).

[Ar] E. Arjomandi, "An efficient algorithm for coloring the edges of a graph with $\Delta+1$ colors", *Canadian Journal of Operational Research and Information Processing (INFOR)*, vol. 20 (1982), pp. 82-101.

[Be] J. Beck, "Balanced two-colorings of finite sets in the cube", *Discrete Math*, vol. 73, (1988/89), pp. 13-25.

[BF] J. Beck and T. Fiala, "Integer-making theorems," *Discrete Applied Mathematics*, vol. 3 (1981) pp. 1-8.

[BR] B. Berger and J. Rompel, "Simulating ($\log^c n$)-wise independence in NC," *Journal of the ACM*, vol. 38 (1991), pp. 1026-1046. Also: *30th Annual Symposium on the Foundations of Computer Science*, 1989.

[BRS] B. Berger, J. Rompel and P. Shor, "Efficient NC algorithms for set cover with applications to learning and geometry", *30th Annual Symposium on the Foundations of Computer Science*, 1989, pp. 2-7.

[Ch] H. Chernoff, "A measure of asymptotic efficiency for tests based on the sum of observations," *Annals of Mathematical Statistics*, vol. 23 (1952), pp. 492-509.

[CF] B. Chazelle and J. Friedman, "A deterministic view of random sampling and its use in geometry," *29th Annual Symposium on the Foundations of Computer Science*, 1988, pp. 539-549.

[Cl1] K. L. Clarkson, "A randomized algorithm for closest point queries", *Siam J. on Computing*, vol. 17 (1988), pp. 830-847.

[Cl2] K. L. Clarkson, "New applications of random sampling in computational geometry", *Disc. Comp. Geom.*, vol.2 (1987), pp. 195-222.

[Cl3] K. L. Clarkson, "Applications of random sampling in computational geometry", *Proc. 4th Annual ACM Symposium on Computational Geometry* (1988), pp. 1-11.

[CS] K. L. Clarkson and P. W. Shor, "Algorithms for diametrical pairs and convex hulls that are optimal , randomized, and incremental", *Proc. 4th Annual ACM Symposium on Computational Geometry* (1988), pp. 12-17.

[ES] P. Erdös and J. Selfridge, "On a combinatorial game", *J. Combinatorial Theory*, ser. B, vol. 14 (1973), pp. 298-301.

[Fe] W. Feller, *An introduction to the theory of probability and its applications*, John Wiley and Sons, 1967.

[GL] Z. Galil and D. Leven, "NP completeness of finding the chromatic index of regular graphs", *Journal of Algorithms*, vol. 4 (1983), pp. 35-44.

[Ho] I. Holyer, "The NP-completeness of edge-coloring", *Siam J. on Computing*, vol. 10 (1981), pp. 718-720.

[HW] D. Haussler and E. Welzl, "Epsilon-nets and simplex range queries," *Disc. Comp. Geom.*, vol.2 (1987), pp. 127-151.

[KM] D. Karger and R. Motwani, "Derandomization through Approximation: An $\mathcal{NC}$ Algorithm for Minimum Cuts," Report No. STAN-CS-93-1471, Department of Computer Science, Stanford University, 1993.

[KS] H. J. Karloff and D. B. Shmoys, "Efficient parallel algorithms for edge coloring problems," *J. of Algorithms,* vol. 8 (1987), pp. 39-52.

[KUW1] R. M. Karp, E. Upfal and A. Wigderson, "Constructing a perfect matching is in random NC," *Combinatorica*, vol. 6 (1986), pp 35-48.

[KUW2] R. M. Karp, E. Upfal and A. Wigderson, "The complexity of parallel search", *JCSS*, vol. 36 (1985), pp. 225-253.

[KW] R. M. Karp and A. Wigderson, "A fast parallel algorithm for the maximal independent set problem," *Journal of the ACM*, vol. 32 (1985), pp. 762-773.

[LPV] G. F. Lev, N. Pippenger and L. G. Valiant, "A fast parallel algorithm for routing in permutation networks," *IEEE Trans. Comput.*, vol. 30 (1981), pp. 93-100.

[Lo] L. Lovàsz, "On the ratio of optimal integral and fractional covers", *Discrete Math*, vol. 13 (1975), pp. 383-390.

[Lu1] M. Luby, "A simple parallel algorithm for the maximal independent set," *Siam J. on Computing*, vol. 15 (1986), pp. 1036-1053.

[Lu2] M. Luby, "Removing randomness in parallel computation without a processor penalty," *29th Annual Symposium on Foundations of Computer Science*, 1988, pp. 162-173.

[MNN] R. Motwani, J. Naor and M. Naor, "The probabilistic method yields deterministic parallel algorithms," *30th Annual Symposium on the Foundations of Computer Science*, 1989, pp. 8-13.

[MS] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*, North Holland, Amsterdam, 1977.

[Mu] K. Mulmuley, "A fast parallel algorithm to compute the rank of a matrix over an arbitrary field," *Combinatorica*, vol. 7, (1987), pp. 101-104.

[MVV] K. Mulmuley, U. V. Vazirani and V. V. Vazirani, "Matching is as easy as matrix inversion," *19th Annual ACM Symposium on the Theory of Computing*, 1987, pp. 355-364.

[Na] M. Naor, "Constructing Ramsey graphs from small probability spaces", IBM Research Report RJ 8810, 1992.

[NN] J. Naor and M. Naor, "Small bias probability spaces: efficient constructions and applications", Siam Journal on Computing, Vol. 22, (1993), pp. 838-856.

[OS] J. E. Olson and J. Spencer, "Balancing families of sets," *J. Combinatorial Theory (Series A)*, vol. 25 (1978), pp. 29-37.

[Ra] P. Raghavan, "Probabilistic construction of deterministic algorithms: approximating packing integer programs," *JCSS*, vol. 37 (1988), pp. 130-143.

[RS] J. H. Reif and S. Sen, "Optimal randomized parallel algorithms for computational geometry", *Proceedings 16th International Conference on Parallel Processing*, St. Charles, IL, 1987. Full version: Duke University Tech. Report, CS-88-01, (1988).

[Sp] J. Spencer, *Ten lectures on the probabilistic method.* SIAM (Philadelphia), 1987.

[Sp1] J. Spencer, "Six standard deviations suffice," *Trans. American Mathematical Society*, vol. 289 (1985), pp 679-706.

[Sp2] J. Spencer, "Balancing vectors in the max norm", *Combinatorica*, vol. 6 (1986), pp. 55-65.

[Vi] V. G. Vizing, "On an estimate of the chromatic class of a p-graph" (in Russian), *Diskret. Anal.*, vol. 3 (1964), pp. 25-30.