Simulating $(\log^c n)$-Wise Independence in *NC*

BONNIE BERGER AND JOHN ROMPEL

Massachusetts Institute of Technology, Cambridge, Massachusetts

Abstract. A general framework for removing randomness from randomized *NC* algorithms whose analysis uses only polylogarithmic independence is developed. Previously, no techniques were known to remove the randomness from those randomized *NC* algorithms depending on more than constant independence. One application of our techniques is an *NC* algorithm for the set discrepancy problem, which can be used to obtain many other *NC* algorithms, including a better *NC* edge coloring algorithm. As another application of the techniques in this paper, an *NC* algorithm for the hypergraph coloring problem is provided.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism*; *probabilistic computation*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Discrepancy, removing randomness

1. *Introduction*

A fundamental issue for theoretical computer science is the degree to which randomness helps in computation. In many cases, the most natural algorithm to solve a problem involves randomness. Often, however, it is possible to convert a randomized algorithm into a deterministic one.

For many applications [1, 13, 15, 20, 22], the problem of removing randomness from an algorithm can be solved by finding an $X = \langle X_1, \ldots, X_n \rangle$ such that $F(X) \geq E[F(X)]$, for some function F on some sample space $\mathcal{S}$ over which the expectation is to be computed. The problem is then how best to find a good sample point (e.g., an X such that $F(X) \geq E[F(X)]$) in $\mathcal{S}$. If

An earlier version of this paper appeared in the *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science* (Oct. 1989).

B. Berger was supported in part by a Graduate Fellowship from ARO Grant DAAL03-86-K-0171, by DARPA Contract N00014-89-J-1988, and by National Science Foundation PYI Grant CCR 86-57688 with matching support from UPS, IBM, and Sun Microsystems.

J. Rompel was supported in part by a National Science Foundation graduate fellowship, DARPA Contracts N00014-87-K-825 and N00014-89-J-1988, and Air Force Grant AFSOR-86-0078.

Authors' address: Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

the space of sample points is small (e.g., polynomial), then this can be accomplished by brute force; namely, we could try all points until we get a good one, for one must exist. However, typical sample spaces are larger than polynomial, and brute force is too expensive. In such situations, the only general method available is the *method of conditional probabilities*, developed through the papers of Erdös and Selfridge [9a], Spencer [21, 22], and Raghavan [20]. This method works by setting the X_i's one by one in such a way as to not decrease the conditional expectation (e.g., setting X_{i+1} so that $E[F(X) \mid X_1, \ldots, X_{i+1}] \geq E[F(X) \mid X_1, \ldots, X_i]$). The main difficulty with this approach is in computing the conditional expectations—the ability to do so determines when the method can and cannot be used.

Unfortunately, the method of conditional probabilities is inherently sequential; hence, the running time of the Spencer and Raghavan algorithms is at least n. Since the best time one could hope for is logarithmic in the size of the sample space, for large sample spaces this is probably as good as one can get; yet, for smaller spaces, n is far from optimal. The only improvement to this approach was by Luby [16] who showed how to search in time logarithmic in the size of the sample space for the special case of pairwise independence, thereby improving the processor efficiency of several *NC* algorithms ($\Delta + 1$ vertex coloring, MIS, and maximal matching) from $n^2(n + m)$ to $n + m$.

In this paper, we show how to search in time logarithmic in the size of the sample space for a wide range of functions F and arbitrarily large sample spaces. As a result, we can prove substantially stronger results than is possible with the Luby method. In particular, we are able to derive *NC* algorithms for several problems that were not previously know to be in *NC*, and we can search ($\log^c n$)-wise independent $n^{\log^c n}$-size sample spaces in *NC*.

In Section 2, we demonstrate how our techniques apply to the problem of set discrepancy. In this problem, we are given a set of n points and n subsets of at most Δ of these points, and we want to color the points 0 and 1 so that the discrepancy, or maximum difference between the number of 0's and the number of 1's in any subset, is small. The best known randomized (parallel) algorithm achieves a discrepancy of $O(\sqrt{\Delta \log n})$; Spencer [21] applied the method of conditional probabilities to this algorithm to obtain a deterministic sequential algorithm with the same bound. We show that the randomized algorithm, using only ($\log n/\epsilon \log \Delta$)-independence, gives a discrepancy bound of $\Delta^{1/2+\epsilon}\sqrt{\log n}$ for any fixed $\epsilon > 0$; then we apply our techniques to convert this to an *NC* algorithm that attains the same bound (using $n^{1/\epsilon}$ processors). We find, in fact, that at least $\Omega(\log n/\log \Delta)$-wise independence is required to guarantee this upper bound on discrepancy; thus, our methods to handle more than constant independence are in fact required to get an *NC* algorithm for this particular problem. Our set discrepancy algorithm has many applications. As an example, we give a deterministic version of the Karloff–Shmoys parallel edge coloring algorithm [12], obtaining the same $\Delta + \Delta^{1/2+\epsilon}$ bound on the number of colors used as their randomized algorithm and beating the known deterministic bound of $2\Delta - 1$ [16]. The results of Section 2 first appeared in Berger [5]. Results similar to those in Section 2 were subsequently discovered by Motwani et al. [17], and further work along these lines appears in [18].

In Section 3, we show how to apply our techniques to a large class of problems that depend on ($\log^c n$)-wise independence. In particular, we describe an *NC* algorithm for obtaining the expected value of any function that is the sum of a polynomial number of terms, each depending on $O(\log n)$ binary random variables; for example, a function of the form

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1}, \ldots, X_{i,b \log n}).$$

Alternatively, we can allow the terms to be simple functions of $\log^c n$ random variables; for example, characteristic functions of affine subspaces, which, by a reduction, can be used to build any function that is nonzero for only a polynomial number of points.

In Section 4, we give several methods for extending our technique to multivalued random variables. As an illustration, we consider the hypergraph coloring problem: Given a d-uniform hypergraph $(V, \mathscr{E})$, color the vertices with d colors so that at least $d! \,|\mathscr{E}|/d^d$ edges have one vertex of each color. If d is a constant, this problem can be solved by trying all sample points in a d-wise independent distribution [1]. Using our techniques for handling $(\log n)$-wise independent multivalued random variables, we give a deterministic *NC* algorithm that solves this problem for all d. The particular technique used to handle the multivalued random variables in this problem involves generating and solving a series of problems with binary random variables, highlighting the importance of being able to solve a large class of these.

Finally, in Section 5, we provide improved algorithms and bounds for set discrepancy. Among the results in this section is an *NC* algorithm for weighted discrepancy, which has applications to other problems such as lattice approximation. Also included is an *NC* algorithm that achieves an $O(\sqrt{\Delta \log n})$ discrepancy bound for the case when Δ is polylogarithmic.

2. *An Example of the Method—Set Discrepancy*

2.1. Definition of Problem. Spencer [22, p. 30] defines the *set discrepancy* problem as follows. Let $\mathscr{A} \subseteq 2^{\Gamma}$, $|\mathscr{A}| = |\Gamma| = n$, be a family of finite sets. Let $\chi: \Gamma \to \{-1, +1\}$ be a 2-coloring of the underlying points. Define

$$\chi(A) = \sum_{i \in A} \chi(i);$$
$$disc(\chi) = \max_{A \in \mathscr{A}} |\chi(A)|.$$

We want to fine a χ such that $disc(\chi)$ is small.

How small can we make $disc(\chi)$? Spencer [22, pp. 73–77] shows that there exists an χ with $disc(\chi) = O(\sqrt{n})$. He also shows that this is the best possible; that is, that there exists an $\mathscr{A}$ such that all χ have $disc(\chi) = \Omega(\sqrt{n})$.

It is interesting to bound discrepancy in terms of maximum degree $\Delta = \max_{A \in \mathscr{A}} |A|$. Spencer's lower bound can be easily modified to give, for any Δ, an $\mathscr{A}$ with cardinality n and maximum degree Δ such that all χ have $disc(\chi) = \Omega(\sqrt{\Delta})$. It is easily shown in Section 2.2 that if we pick χ at random, with high probability $disc(\chi)$ is at most $2\sqrt{\Delta \log n}$. This immediately gives an *RNC* algorithm achieving that bound. Spencer [21] shows how to convert this into a deterministic polynomial algorithm. In the sections that follow, we develop an *NC* algorithm that outputs a χ with $disc(\chi) \leq \Delta^{1/2+\epsilon}\sqrt{\log n}$.

An interesting special case of the set discrepancy problem is the *graph discrepancy* problem. Given a graph $G = (V, E)$, we want to find a 2-coloring of the vertices $\chi: V \to \{-1, +1\}$ such that $\max_{v \in V} |\sum_{u \in N(v)} \chi(u)|$ is small, where $N(v) = \{u \mid (v, u) \in E\}$. We can reduce this problem to set discrepancy by letting $\Gamma = V$ and $\mathscr{A} = \{N(v) \mid v \in V\}$. Plugging in our *NC* algorithm for discrepancy, we get a χ with $\max_{v \in V} |\sum_{u \in N(v)} \chi(u)| \le \Delta^{1/2+\epsilon} \sqrt{\log n}$, where Δ is the maximum degree of G.

A variation on the set discrepancy problem is the *weighted discrepancy* problem, where each element of Γ is assigned a real weight between -1 and 1. Then, $\chi(A)$ becomes a weighted sum of the $\chi(i)$'s. In Section 5.2, we give an *NC* algorithm for this problem.

2.2. An *RNC* Algorithm.

Consider the following algorithm for set discrepancy: randomly pick χ until one is found such that $disc(\chi) \le 2\sqrt{\Delta \ln n}$. One iteration of this is clearly in *RNC*. We show that the expected number of iterations is less than two. The following will prove useful:

Proposition 2.1 [22, p. 29]. *Let $X_1, \ldots, X_\Delta$ be independent and identically distributed with $Pr[X_i = +1] = Pr[X_i = -1] = 1/2$. Let $S = \sum_i X_i$. Then, $Pr[S > \lambda] < e^{-\lambda^2/2\Delta}$.*

Lemma 2.2. *$Pr\,[disc(\chi) > 2\sqrt{\Delta \ln n}] < 2/n$.*

Proof. For each $A \in \mathscr{A}$, Proposition 2.1 shows

$$\begin{aligned}\Pr\left[\,|\chi(A)| > 2\sqrt{\Delta \ln n}\,\right] &\le 2\Pr\left[\chi(A) > 2\sqrt{|A| \ln n}\,\right]\\ &< 2e^{-(2\sqrt{|A|\ln n})^2/2|A|}\\ &= \frac{2}{n^2}.\end{aligned}$$

Thus,

$$\begin{aligned}\Pr\left[disc(\chi) > 2\sqrt{\Delta \ln n}\,\right] &\le \sum_{A \in \mathscr{A}} \Pr\left[\,|\chi(A)| > 2\sqrt{\Delta \ln n}\,\right]\\ &\le \frac{2}{n}.\end{aligned}$$ □

Thus, the expected number of iterations is at most $1/(1 - (2/n)) < 2$. So the above is clearly an *RNC* algorithm. Also, one can easily show using Lemma 2.2 that $E[disc(\chi)] \le 2\sqrt{\Delta \ln n}$.

2.3. The Overall Approach.

Lemma 2.2 shows that the probability of $disc(\chi)$ being larger than $2\sqrt{\Delta \ln n}$ is small. This implies that there exists a χ with $disc(\chi) \le 2\sqrt{\Delta \ln n}$. We wish to find such a χ deterministically. Unfortunately, the random construction in Section 2.2 assumed a fully independent distribution, which must have 2^n sample points. Clearly, we cannot search this sample space exhaustively. However, Spencer [22] developed a method to perform a binary search on the sample space. While he achieves a polynomial time algorithm, it requires n decisions to be made. Since each decision depends on previous ones, it seems very unlikely that these decisions could be made in parallel. To get an efficient parallel algorithm, we must work with a smaller sample space. A natural choice would be to choose the vector $\langle \chi(1), \ldots, \chi(n) \rangle$ from a k-wise independent distribution, where k is small.

Definition 2.3. The random variables $X_1, \ldots, X_n$ are *k-wise independent* if for any k-subset of $X_1, \ldots, X_n$ and for any $r_1, \ldots, r_k$,

$$\begin{aligned}&\Pr[X_{i_1} = r_1 \wedge X_{i_2} = r_2 \wedge \cdots \wedge X_{ik} = r_k]\\ &\quad = \Pr[X_{i_1} = r_1]\Pr[X_{i_2} = r_2] \cdots \Pr[X_{i_k} = r_k].\end{aligned}$$

Ideally, our goal is to find a χ with small discrepancy by finding a χ for which $disc(\chi) \le E[disc(\chi)]$, where the expectation is taken over a k-wise independent distribution for some small k. The choice of k is influenced by two factors:

(1) if k is too small, then $E[disc(\chi)]$ might be too large, and
(2) if k is too large, then finding a χ that achieves the expectation takes too long.

As a compromise, we eventually choose $k = \log n/\epsilon \log \Delta$, $\epsilon > 0$.

There are other problems with this approach, however. Most important, to find a good χ, we need to compute expectations of $disc(\chi)$ conditioned on some knowledge of the distribution, in NC. This is hopelessly complicated by the *max* and absolute value in $disc(\chi)$. To get around these problems, we use higher moments, using $\sum_{A\in\mathscr{A}} |\chi(A)|^k$ as an upper bound on $disc^k(\chi)$. If k is even, this gets rid of both the *max* and the absolute value. In other words, we

(1) show that $E[\sum_{A\in\mathscr{A}} |\chi(A)|^k]$ is small for suitable k where the expectation is taken over a k-wise independent distribution, and then
(2) find a χ such that $\sum_{A\in\mathscr{A}} |\chi(A)|^k \le E[\sum_{A\in\mathscr{A}} |\chi(A)|^k]$.

As a consequence, we have found a χ for which $disc^k(\chi)$ is small, and thus for which $disc(\chi)$ is small. By choosing $k = \log n/\epsilon \log \Delta$, we produce a χ for which $disc(\chi) \le \Delta^{1/2+\epsilon}\sqrt{\log n}$. This is not quite the $\sqrt{\Delta \log n}$ bound we got with the RNC algorithm, but it is close.

2.4. Bounding the Independence Needed. Our first task is to bound $E[\sum_{A\in\mathscr{A}} \chi^k(A)]$. This is accomplished in the following lemmas:

Lemma 2.4. *Any function that is the sum of functions depending on at most k random variables each has the same expected value taken over any distribution with k or greater independence.*

Proof. Follows directly from the definition of k-wise independence and linearity of expectation. □

Lemma 2.5. *Let k be a positive even integer. For $\langle\chi(1), \ldots, \chi(n)\rangle$ chosen from a k-wise independent distribution, for all $A \in \mathscr{A}$,*

$$E[\chi^k(A)] \le O\left(\sqrt{k}\left(\frac{k\Delta}{e}\right)^{k/2}\right).$$

Proof. We first observe that $E[\chi^k(A)]$ for any k-wise independent distribution is the same as for the uniform distribution. This follows from Lemma 2.4 and the fact that

$$\chi^k(A) = \left(\sum_{i\in A} \chi(i)\right)^k = \sum_{i_1\in A}\sum_{i_2\in A} \cdots \sum_{i_k\in A} \chi(i_1) \cdots \chi(i_k).$$

Hence, we assume the $\chi(i)$'s are independent and unbiased in the analysis below.

Since k is even, and hence $\chi^k(A)$ nonnegative, we can write

$$\begin{aligned} E[\chi^k(A)] &= \int_0^\infty \Pr[\chi^k(A) > x]\, dx \\ &= \int_0^\infty \Pr[\,|\chi(A)| > x^{1/k}]\, dx \\ &\le \int_0^\infty 2e^{-x^{2/k}/2\Delta}\, dx, \end{aligned}$$

(where the last inequality is from Proposition 2.1). With the change of variables $y = x^{2/k}/2\Delta$, we have

$$\begin{aligned} E[\chi^k(A)] &\le 2\frac{k}{2}(2\Delta)^{k/2}\int_0^\infty y^{k/2-1}e^{-y}\, dy \\ &= 2\frac{k}{2}(2\Delta)^{k/2}\left[-\left(\frac{k}{2}-1\right)!e^{-x}\sum_{i=0}^{k/2-1}\frac{x^i}{i!}\right]_0^\infty \\ &= 2\frac{k}{2}(2\Delta)^{k/2}\left(\frac{k}{2}-1\right)! \\ &= 2\left(\frac{k}{2}\right)!(2\Delta)^{k/2} \\ &\le O\left(\sqrt{k}\left(\frac{k}{2e}\right)^{k/2}(2\Delta)^{k/2}\right) \\ &= O\left(\sqrt{k}\left(\frac{k\Delta}{e}\right)^{k/2}\right). \end{aligned}$$

□

COROLLARY 2.6. *For any even k-wise independent distribution*,

$$E\left[\sum_{A\in\mathcal{A}}\chi^k(A)\right] \le nO\left((k\Delta)^{k/2}\right).$$

We can now give a lower bound on the value for k. We want

$$E\left[\sum_{A\in\mathcal{A}}\chi^k(A)\right]^{1/k} \le \Delta^{1/2+\epsilon}\sqrt{\log n}\,;$$

this is roughly captured by having $n^{1/k} \le \Delta^\epsilon$. This implies we need

$$k = 2\left\lceil\frac{\log n}{2\epsilon\log\Delta}\right\rceil.$$

If k is thus, and we are able to find a χ such that $\sum_{A\in\mathcal{A}}\chi^k(A)$ is at most its expectation, this implies that

$$disc^k(\chi) \le \sum_{A\in\mathcal{A}}\chi^k(A) \le E\left[\sum_{A\in\mathcal{A}}\chi^k(A)\right] \le O\left(n(k\Delta)^{k/2}\right).$$

So,

$$\begin{aligned} disc(\chi) &\le O(n^{1/k}\sqrt{k\Delta}) \\ &\le O(\Delta^\epsilon\sqrt{k}\,\sqrt{\Delta}) \\ &\le \Delta^{1/2+\epsilon}\sqrt{\log n}\,. \end{aligned}$$

It is worth pointing out that the preceding analysis is in some sense tight; that is, that to get $disc(\chi) \leq \Delta^{1/2+\epsilon}$, any method based on independence alone requires at least $((2 \log n)/\log \Delta)$-wise independence. This notion is captured by the following theorem:

THEOREM 2.7. *For any n and Δ, we can construct a $((2 \log n)/\log \Delta)$-wise independent distribution and a set system $\mathscr{A}$ of size n and maximum degree Δ such that $E[disc(\chi)] = \Omega(\Delta)$.*

PROOF. Our distribution is as follows: first pick $\chi(1), \ldots, \chi(\Delta)$ from a $((2 \log n)/\log \Delta)$-wise independent distribution with at most n sample points (see [1] for construction). Then, to pad out the distribution to n variables, pick $\chi(\Delta + 1), \ldots, \chi(n)$, independently of $\chi(1), \ldots, \chi(\Delta)$, from an arbitrary $((2 \log n)/\log \Delta)$-wise independent distribution. Now we construct $\mathscr{A}$ such that for every possible sample point in the distribution for the first Δ variables, there is some set with large discrepancy. This implies that the expected discrepancy is large. $\mathscr{A}$ is constructed as follows: for each sample point in the distribution for $\chi(1), \ldots, \chi(\Delta)$, we include in $\mathscr{A}$ the larger of sets $\{i \mid 1 \leq i \leq \Delta, \chi(i) = -1\}$ and $\{i \mid 1 \leq i \leq \Delta, \chi(i) = +1\}$. This ensures that for each sample point, we have a corresponding set $A \in \mathscr{A}$ (with $\Delta/2 \leq |A| \leq \Delta$) whose elements are assigned either all $+1$'s or all -1's. Thus, $disc(\chi) \geq \Delta/2$, which implies $E[disc(\chi)] \geq \Delta/2$. □

The next three sections will be devoted to finding a χ such that $\sum_{A \in \mathscr{A}} \chi^k(A)$ is at most its expectation. Since it is more convenient to work with 0/1 variables, we let $\chi(i) = (-1)^{X_i}$, where $X_i \in \{0, 1\}$. Let

$$F(X) = -\sum_{A \in \mathscr{A}} \chi^k(A) = -\sum_{A \in \mathscr{A}} \sum_{i_1 \in A} \cdots \sum_{i_k \in A} (-1)^{X_{i_1} + \cdots + X_{i_k}}.$$

Then finding a χ such that $\sum_{A \in \mathscr{A}} \chi^k(A)$ is at most its expectation is the same as finding an X such that $F(X) \geq E[F(X)]$.

2.5. GENERATING k-WISE INDEPENDENT VARIABLES.

It still remains to demonstrate a k-wise independent distribution on which we can perform a binary search efficiently in parallel.

Luby [16] gave the following such distribution for the case $k = 2$. Let $l = \lceil \log(n + 1) \rceil + 1$ and $\omega = \langle \omega_1, \ldots, \omega_l \rangle$ be picked uniformly from Z_2^l. Define random variables $X_1, \ldots, X_n$ such that

$$X_i = \left(\sum_{j=1}^{l-1} (i_j \omega_j) + \omega_l \right) \bmod 2,$$

where $\langle i_1, \ldots, i_{l-1} \rangle$ is the binary expansion of i. Observe that Luby's distribution is not 4-wise independent: In particular, X_4, X_5, X_6, and X_7 are dependent since $X_7 = X_4 + X_5 + X_6$.

We extend Luby's distribution to be k-wise independent for all k as follows: We assign a label $a_i \in Z_2^l$ to each point i, where l is bounded by some polylogarithmic function of n. We pick $\omega \in Z_2^l$ uniformly at random, and let

$$X_i = a_i \cdot \omega.$$

Note that we can express Luby's distribution in this framework by letting $a_i = \langle i_1, \ldots, i_{l-1}, 1\rangle$.

The main benefit of our distribution is that we can now give a necessary and sufficient condition for the X_i's to be independent and unbiased. (By unbiased, we mean each X_i has equal probability of being 0 or 1.) The following result is similar to others used in the literature [1]. For completeness, we provide a proof in what follows.

THEOREM 2.8. *$X_{i_1}, \ldots, X_{i_k}$ are independent and unbiased if and only if $a_{i_1}, \ldots, a_{i_k}$ are linearly independent as vectors over Z_2.*

PROOF. First, we prove the "only if" direction. Suppose we have k a's that are not linear independent; that is, we have $\sum_{j=1}^{k} \alpha_j a_{i_j} = 0$, where some $\alpha_{j^*} \neq 0$. Therefore,

$$\begin{aligned}\sum_{j=1}^{k} \alpha_j X_{i_j} &= \sum_{j=1}^{k} \alpha_j (a_{i_j} \cdot \omega) \\ &= \left(\sum_{j=1}^{k} \alpha_j a_{i_j} \right) \cdot \omega \\ &= 0 \cdot \omega \\ &= 0.\end{aligned}$$

But consider the event, E, that $X_{i_{j^*}} = 1$ and $X_{i_j} = 0$ for all $j \neq j^*$. Then

$$\sum_{j=1}^{k} \alpha_j X_{i_j} = \alpha_{j^*} \neq 0.$$

So E is not possible; that is, $\Pr[E] = 0$. But this implies that $X_{i_1}, \ldots, X_{i_k}$ are not independent, as any distribution where they were would have $\Pr[E] = 2^{-k}$.

Now for the proof of the "if" direction. Assume $a_{i_1}, \ldots, a_{i_k}$ are linearly independent. Let A be the matrix containing $a_{i_1}, \ldots, a_{i_k}$ as row vectors. Then, we have

$$A\omega = \begin{pmatrix} a_{i_1} \\ a_{i_2} \\ \vdots \\ a_{i_k} \end{pmatrix} \begin{bmatrix} \omega \end{bmatrix} = \begin{bmatrix} X_{i_1} \\ X_{i_2} \\ \vdots \\ X_{i_k} \end{bmatrix}.$$

Since $a_{i_1}, \ldots, a_{i_k}$ are linearly independent, we can create a new nonsingular matrix $\hat{A}$ by augmenting A with $l - k$ new rows. Then $\hat{A} \cdot \omega = \hat{x}$, where the first k entries of $\hat{x}$ are $X_{i_1}, \ldots, X_{i_k}$. $\hat{A}$ is invertible, so $\omega = \hat{A}^{-1} \cdot \hat{x}$; hence, there is a 1-1 correspondence between $\hat{x}$'s and ω's implying that the $\hat{x}$'s are uniformly distributed. But there are exactly $2^{l-k}\hat{x}$'s for every assignment to $X_{i_1}, \ldots, X_{i_k}$. So every assignment to $X_{i_1}, \ldots, X_{i_k}$ has probability $2^{l-k}/2^l = 2^{-k}$ of occurring. Therefore, $X_{i_1}, \ldots, X_{i_k}$ are independent and unbiased. □

To get $X_1, \ldots, X_n$, which are k-wise independent, we need a set of labels $a_1, \ldots, a_n$ such that every k of them are linearly independent. (By Theorem 2.8, this gives us k-wise independence of the X_i's.) In fact, it suffices to get an

$n \times r$ matrix over $GF(2^s)$ with the property that any $\log n$ rows are linearly independent. Letting a_i be the ith row with each element $\alpha_0 + \alpha_1 x + \cdots + \alpha_{s-1} x^{s-1} \in GF(2^s)$ expanded out to $\langle \alpha_0, \ldots, \alpha_{s-1} \rangle$ gives length $l = rs$ labels such that any $\log n$ are linearly independent over Z_2. Several well-known ways of constructing such matrices, for $l = O(k \log n)$ are described in [1, 6, 19].

In fact, almost all $n \times O(k \log n)$ matrices will work, as is demonstrated by the following theorem:

THEOREM 2.9. *If $l > k \log(n/k) + 2k + t$, then a random set of labels $\{a_1, \ldots, a_n\} \subseteq Z_2^l$, are k-wise linearly independent with probability at least* $1 - 2^{-t}$.

PROOF. Pick $A = \langle a_1, \ldots, a_n \rangle \in (Z_2^l)^n$ at random.

$$\begin{aligned}
&\Pr[\text{every } k \text{ of } a_1, \ldots, a_n \text{ are linearly independent}] \\
&\geq 1 - \sum_{\substack{S \subseteq \{1, \ldots, n\} \\ |S| = k}} \Pr[\{a_i \mid i \in S\} \text{ is linearly dependent}] \\
&= 1 - \binom{n}{k} \Pr\left[\exists \alpha_1, \ldots, \alpha_k \text{ not all } 0, \text{ such that } \sum_{i=1}^{k} \alpha_i a_i = 0\right] \\
&\geq 1 - \binom{n}{k} \sum_{\substack{\alpha_1, \ldots, \alpha_k \in Z_2 \\ \text{not all } 0}} \Pr\left[\sum_{i=1}^{k} \alpha_i a_i = 0\right].
\end{aligned}$$

Consider the a_i corresponding to the last non-zero α_i. Only one value for this a_i gives a 0 sum, so $\Pr[\sum_{i=1}^{k} \alpha_i a_i = 0] = 2^{-l}$. Therefore, continuing the above sequence of relations, we have that

$$\begin{aligned}
&1 - \binom{n}{k} \sum_{\substack{\alpha_1, \ldots, \alpha_k \in Z_2 \\ \text{not all } 0}} \Pr\left[\sum_{i=1}^{k} \alpha_i a_i = 0\right] \\
&= 1 - \binom{n}{k} (2^k - 1) 2^{-l} \\
&\geq 1 - 2^{k(\log n + 1 - \log k) + k - l} \\
&\geq 1 - 2^{-t}. \qquad \square
\end{aligned}$$

Clearly, a random set of labels of length $l = k \log(4n)$ almost certainly gives us k-wise independence.

Since any k-wise independent distribution on n random variables must have a sample space of size $\Omega((n/k)^{\lfloor k/2 \rfloor})$ [1, 7], the labels $a_1, \ldots, a_n$ must be $\Omega(k \log n - k \log k)$ bits long.

2.6. ZEROING IN ON A GOOD SAMPLE POINT.

Now that we have a k-wise independent distribution, we explain how to do a binary search on it efficiently in parallel. We employ the method of conditional probabilities, described earlier, for zeroing in on a "good" sample point; that is, an ω such that $F(X) \geq E[F(X)]$. Even though the method is inherently sequential, since ω is only $O(\log^2 n)$ bits long, our resulting algorithm will be in NC.

To zero in on a good ω, one bit of ω is determined at a time, thereby performing a binary search on the ω's. This is done as follows. At the beginning of iteration t, assume we have set $\omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}$. Then, we compute $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 0]$ and $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 1]$. We then set ω_t to the $s_t \in \{0, 1\}$ which maximizes $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = s_t]$. We show how to compute these conditional expectations in Section 2.7.

LEMMA 2.10. *After step t of the above procedure, $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t] \geq E[F(X)]$.*

PROOF (by induction on $|s|$). The case $|s| = 0$ is clearly true. Assume this lemma is true for $t - 1$; that is, we have

$$E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}] \geq E[F(X)].$$

Then

$$\begin{aligned} E[F(X) \mid \omega_1 &= s_1, \ldots, \omega_t = s_t] \\ &= \max(E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 0], \\ &\qquad E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 1]) \\ &\geq (E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 0] \\ &\qquad + E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 1])/2 \\ &= E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}] \\ &\geq E[F(X)] \text{ (by inductive hypothesis).} \end{aligned}$$

□

COROLLARY 2.11. *The output of the above procedure is an X such that $F(X) \geq E[F(X)]$.*

2.7. COMPUTING CONDITIONAL EXPECTATIONS. In general, computing conditional expectations is hard to do and separates when one can and cannot use the method of conditional probabilities to zero in on a good sample point. Fortunately, in the case of discrepancy, we have devised a simple and efficient approach for computing conditional expectations. Recall that to solve discrepancy, we need to compute conditional expectations $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t]$ where $F(X)$ is of the special form

$$F(X) = -\sum_{A \in \mathscr{A}} \sum_{i_1 \in A} \cdots \sum_{i_k \in A} (-1)^{X_{i_1} + \cdots + X_{i_k}}.$$

Using linearity of expected value, we can break this up into components

$$\begin{aligned} h_{i_1 \cdots i_k}(s) &= E\left[(-1)^{\sum_{j=1}^{k} X_{i_j}} \mid \omega_1 = s_1, \ldots, \omega_t = s_t\right] \\ &= E\left[(-1)^{\sum_{j=1}^{k} a_{i_j} \cdot \omega} \mid \omega_1 = s_1, \ldots, \omega_t = s_t\right] \\ &= E\left[(-1)^{\bar{a} \cdot \omega} \mid \omega_1 = s_1, \ldots, \omega_t = s_t\right], \end{aligned}$$

where $\bar{a} = \sum_{j=1}^{k} a_{i_j}$. Let r be the last position that contains a 1 in $\bar{a}$. If $t < r$, then $\bar{a} \cdot \omega$ is unbiased, and therefore $h_{i_1 \cdots i_k}(s) = 0$. Otherwise, $\bar{a} \cdot \omega$ is the same for all ω that extend s, and hence $h_{i_1 \cdots i_k}(s) = (-1)^{\bar{a} \cdot \omega}$. Assuming we have precomputed $\bar{a}$ and r, we can compute $h_{i_1 \cdots i_k}(s)$ in constant time by extending a partial sum $\sum_{j=1}^{t} \bar{a}_j s_j$ at each iteration and outputting $h_{i_1 \cdots i_k}(s) = 0$ if $t < r$ and $h_{i_1 \cdots i_k}(s) = (-1)^{\sum_{j=1}^{t} \bar{a}_j s_j}$, if $t \geq r$.

To compute $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t]$, we need one processor for each possible $\langle A, i_1, \ldots, i_k \rangle$, that is, at most $n\Delta^k$ total. Therefore, k must be $O(\log n / \log \Delta)$. Letting k be the minimum possible, $2\lceil \log n / 2\epsilon \log \Delta \rceil$, implies that $n^{3+1/\epsilon}$ processors are sufficient.

Then, we can compute all $h_{i_1 \cdots i_k}(s)$ terms in parallel in constant time and sum them to get $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t]$ in $O(\log n)$ time. Thus, we spend $O(l \log n)$ time in the l iterations of our procedure. In addition, we can perform the precomputation required above in $O(\log n)$ time as well. Since $l = O(\log^2 n)$, this yields an $O(\log^3 n)$ algorithm for discrepancy.

2.8. Application to Edge Coloring. An *edge coloring* of a graph $G = (V, E)$ is an assignment of colors to all edges of the graph, so that any two edges that share a common vertex are assigned different colors. Let Δ be the maximum degree in G. Observe that any coloring requires at least Δ colors. In fact, Vizing [23] implicitly gives a polynomial time algorithm to $\Delta + 1$ color any graph. Karloff and Shmoys [12] provide a parallel implementation of this algorithm to get a $\Delta + 1$ coloring of any graph in time $O(\Delta^{O(1)} \log^{O(1)} n)$ using a polynomial number of processors. Also of interest, there exist *NC* algorithms for optimally coloring bipartite graphs with Δ colors [2, 8, 10, 14]. Furthermore, there is a trivial *NC* algorithm to $2\Delta - 1$ color any graph by $\Delta + 1$ vertex coloring [16] the line graph. Berger and Shor (unpublished notes) and Karloff and Naor (private communication) found *NC* algorithms to $\Delta + \Delta / \log^{O(1)} n$ color any graph.

Of particular interest here, there is an *RNC* algorithm in [12] which $\Delta + \Delta^{1/2+\epsilon}$ colors any graph. We remove the randomness from this algorithm by using the techniques discussed above. The *RNC* algorithm, Algorithm A, is as follows:

(1) If $\Delta < (\log n)^{1/\epsilon}$, then use the Karloff–Shmoys $\Delta + 1$ deterministic algorithm [12].
(2) Run an *RNC* algorithm for graph discrepancy, randomly picking χ until $disc(\chi) \le \Delta^{1/2+\epsilon}$. Let $A = \{v \mid \chi(v) = +1\}$ and $B = \{v \mid \chi(v) = -1\}$. This partition gives us two graphs, both with vertex set V: bipartite graph G_1, which has all the edges of G between A and B; Graph G_2 which has all the other edges of G.
(3) Color G_1 using the Δ coloring algorithm for bipartite graphs.
(4) Run Algorithm A recursively on G_2, using a new set of colors.

This algorithm works because the above partition implies that both G_1 and G_2 have maximum degree at most $\Delta/2 + \Delta^{1/2+\epsilon}$.

To make Algorithm A deterministic, we need only demonstrate a deterministic method for graph discrepancy, which we said in Section 2.1 was a special case of the set discrepancy problem. Plugging in the set discrepancy results with $\epsilon' = \epsilon/2$, we get a χ such that $disc(\chi) \le \Delta^{1/2+\epsilon'} \sqrt{\log n}$. Note that $\Delta \ge (\log n)^{1/\epsilon}$, since we handled the other case in Step 1 of Algorithm A. Thus, $\sqrt{\log n} \le \Delta^{\epsilon/2}$. So we have $disc(\chi) \le \Delta^{1/2+\epsilon}$, which implies $\chi(N(v)) \le \Delta^{1/2+\epsilon}$ for all $v \in V$.

3. *Setting up a General Framework*

For discrepancy-based problems, we considered a very specific class of functions; namely, those of the form $\sum_{i=1}^{n^a} (-1)^{\sum_{j=1}^{k} X_{i_j}}$, and showed how to achieve the expected value for these. What can we do in general? In particular, for which functions can we compute conditional expectations (the method of Section 2.6 will then apply to achieve the expected value)? In order to give ourselves a fighting chance, we restrict our attention to functions of the form

$$F(X) = \sum_{i=1}^{m} f_i(X_{i,1}, X_{i,2}, \dots, X_{i,k}).$$

These are exactly the functions for which we can apply Lemma 2.4 to show that k-wise independence gives the same expected value as full independence. Since we require at least one processor for each f_i term, we insist that m be polynomial in n. In Section 3.1, we show how to compute conditional expectations for arbitrary f_i when $k = O(\log n)$. In Section 3.2, we describe the f_i's for which we can handle the case $k = O(\log^c n)$.

3.1. LOGARITHMIC NUMBER OF 0/1 VARIABLES. In this section, we present two different methods for computing conditional expectations for functions of the form

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1}, \dots, X_{i,b\log n}).$$

We present both methods because, depending on what problem we wish to solve, either of the two methods may be more efficient.

The first method is to rewrite F to be of the form solved in Section 2.7. Let $g: 2^{\{1,\dots,k\}} \to \mathbf{R}$ be such that

$$g(A) = f_i(X_{i,1}, \dots, X_{i,k}) \qquad \text{such that} \qquad X_{i,j} = \begin{cases} 1 & \text{if } j \in A \\ 0 & \text{if } j \notin A, \end{cases}$$

(i.e., g of a set is f_i applied to its characteristic vector). The next proposition follows from the theory of harmonic analysis on the cube.

PROPOSITION 3.1 [11]. *$g(A) = \sum_{S\subseteq\{1,\dots,k\}} \alpha_S(-1)^{|S\cap A|}$, where $\alpha_S = 2^{-k}\sum_{B\subseteq\{1,\dots,k\}} g(B)(-1)^{|S\cap B|}$.*

Thus,

$$\begin{aligned} f_i(X_{i,1}, \dots, X_{i,k}) &= g(\{j \mid X_{i,j} = 1\}) \\ &= \sum_S \alpha_S(-1)^{|\{j \mid X_{i,j}=1\}\cap S|} \qquad \text{(by Proposition 3.1)} \\ &= \sum_S \alpha_S(-1)^{\sum_{j\in S} X_{i,j}}. \end{aligned}$$

Since we have now written F as $\sum_{i=1}^{n^{a+b}} \alpha_i(-1)^{\sum_j X_{i,j}}$, we can apply the technique of Section 2.7 to compute conditional expectations. This gives us the following theorem:

THEOREM 3.2. *There is an NC algorithm which given any F: $Z_2^n \to \mathbf{R}$ of the form*

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1}, \dots, X_{i,b\log n}),$$

outputs an X with $F(X) \geq E[F(X)]$.

An alternative method for computing conditional expectations for F is as follows: First, note that, by linearity of expectation, it suffices to compute the conditional expectations of the individual f_i and the sum. Assume we wish to compute

$$E\left[f_i(X_{i,1},\dots,X_{i,b\log n}) \mid \omega_1 = s_1,\dots,\omega_t = s_t\right].$$

Let x be the vector $\langle X_{i,1},\dots,X_{i,b\log n}\rangle$, and let A be the matrix whose rows are the corresponding labels $a_{i,1},\dots,a_{i,b\log n}$. Then $x = A\omega$. So

$$\begin{aligned}E&\left[f_i(X_{i,1},\dots,X_{i,b\log,n}) \mid \omega = s_1,\dots,\omega_t = s_t\right]\\ &= \sum_x f_i(x)\Pr\left[A\omega = x \mid \omega_1 = s_1,\dots,\omega_t = s_t\right].\end{aligned}$$

If we let $\omega' = \langle\omega_1,\dots,\omega_t\rangle$, $\omega'' = \langle\omega_{t+1},\dots,\omega_l\rangle$, A' and A'' be the first t and last $l-t$ columns of A, respectively, and $s = \langle s_1,\dots,s_t\rangle$, then

$$\begin{aligned}E\left[f_i(x) \mid \omega_1 = s_1,\dots,\omega_t = s_t\right] &= \sum_x f_i(x)\Pr\left[A'\omega' + A''\omega'' = x \mid \omega' = s\right]\\ &= \sum_x f_i(x)\Pr\left[A''\omega'' = x - A's\right].\end{aligned}$$

For each x, we can test if the linear system $A''\omega'' = x - A's$ is solvable; if it is, $\Pr[A''\omega'' = x - A's] = 2^{-\operatorname{rank}(A'')}$; otherwise, $\Pr[A''\omega'' = x - A's] = 0$. Since we can compute the contribution of each of the x's in parallel, we can compute the desired conditional expectation in NC, thus giving an alternate proof for Theorem 3.2.

3.2. Polylogarithmic Number of 0/1 Variables. Now, we consider the case of functions depending on a polylogarithmic number of variables. A simple counting argument shows that in NC we cannot compute all functions of $\log^c n$ variables, when $c > 1$, let alone compute conditional expectations of them. In fact, both techniques of Section 3.1 require evaluating f_i at every point; if f_i depends on more than a logarithmic number of variables, there will be a superpolynomial number of points to evaluate. However, there are some special cases for which we can compute conditional expectations.

The first special case we can handle is

$$f_i(X_{i,1},\dots,X_{i,k}) = (-1)^{\Sigma_j X_{i,j}}.$$

This function can be evaluated using the techniques of Section 2.7, even if $k = \log^c n$. In Section 3.1, we showed how to transform any function into a linear combination of these; if this transformation is already known and provides only a polynomial number of non-zero α's, we can use this technique.

The next special case is based on the second technique of Section 3.1. Recall, we had

$$\begin{aligned}E&\left[f_i(X_{i,1},\dots,X_{i,k}) \mid \omega_1 = s_1,\dots,\omega_t = s_t\right]\\ &= \sum_x f_i(x)\Pr\left[A''\omega'' = x - A's\right].\end{aligned}$$

We can restrict our attention to those x for which $f_i(x) \neq 0$. If there are a polynomial number of these, we can compute conditional expectations of f_i for $k = \log^c n$. Some examples of this are logical AND and NOR of a polylogarithmic number of variables (each has one nonzero point).

A variant of the above, $f_i(X_{i,1}, \ldots, X_{i,k}) = X_{i,1}X_{i,2} \cdots X_{i_k}$ (i.e., the special case of monomials), was subsequently considered by Motwani et al. [18]. This is equivalent to the logical AND just described. Note that handling monomials is strictly weaker than the case above since, for example, it is impossible to write a polylogarithmic variable NOR as a linear combination of a polynomial number of monomials.

Finally, we give a type of f_i which can simulate all the above and more. Consider functions of the form

$$f_i(x) = \begin{cases} 1 & \text{if} \quad x = y_i + T_i z \quad \text{for some} \quad z \in Z_2^k, \\ 0 & \text{otherwise}, \end{cases}$$

for some $y_i \in Z_2^k$, $T_i \in Z_2^{k \times k}$.

These are the characteristic functions of affine subspaces. Included are characteristic functions of all single points; we can write any function with a polynomial number of nonzero points as a linear combination of these. Functions $(-1)^{\sum_j X_{i,j}}$ can also be put in this form. To compute conditional expectations, we use a variant of our linear algebra method:

$$\begin{aligned} &E[f_i(x) \mid \omega_1 = s_1, \ldots, \omega_t = s_t] \\ &\quad = \Pr[A\omega = y_i + T_i z \text{ has a solution } \mid \omega' = s] \\ &\quad = \Pr[A''\omega'' + T_i z = y_i'' - A's \text{ has a solution}], \end{aligned}$$

which can be computed by performing Gaussian Elimination to determine how many bits of ω'' are free to vary. This gives us the following theorem:

THEOREM 3.3. *There is an NC algorithm that given any* F: $Z_2^n \to \mathbf{R}$ *of the form*

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1}, \ldots, X_{i,b\log^c n}),$$

where each f_i *is the characteristic function of some affine subspace of* $Z_2^{\log^c n}$, *outputs an* X *with* $F(X) \geq E[F(X)]$.

4. *Handling Multivalues—The Hypergraph Coloring Problem*

In the previous sections, we were only concerned with the case where the random variables took on values 0 and 1 each with probability 1/2. Yet, for many problems, this model is too restrictive. In this section, we expand our framework to consider random variables drawn from a uniform distribution over a larger set of values. This can then be used to simulate nonuniform distributions. We demonstrate our techniques for handling multivalued random variables on the following problem:

A *hypergraph* $\mathscr{H} = (V, \mathscr{E})$ is a system $\mathscr{E}$ of subsets of V called *edges*. $\mathscr{H}$ is d-uniform if every edge has d elements. Erdös and Kleitman [9] and Alon et al. [1] define the *large d-partite subhypergraph* problem as follows: Given a d-uniform hypergraph $\mathscr{H} = (V, \mathscr{E})$, find a d-coloring of V such that the number of edges in $\mathscr{E}$ having precisely one vertex of each color is at least $|\mathscr{E}|\, d!/d^d$. Alon et al. [1] showed this problem is in *NC* for constant d. We show in this section that this problem is in *NC* for all d. Since the case of $d > \ln|\mathscr{E}| + \Omega(\lg\lg|\mathscr{E}|)$ is trivially satisfied by any coloring that colors one hyperedge correctly, we will henceforth restrict our attention to the case $d \leq \ln|\mathscr{E}| + O(\lg\lg|\mathscr{E}|)$.

4.1. RANDOMIZED ALGORITHM. In this section, we give a randomized parallel algorithm and prove that the expected number of properly colored edges is as desired. The randomized algorithm is as follows: Randomly assign to each vertex an $l = \log(2 \mid \mathscr{E} \mid d!d^2) = O(\log \mid \mathscr{E} \mid \log\log \mid \mathscr{E} \mid)$ bit label. Designate these as random variables $Y = \langle Y_1, \ldots, Y_{|V|} \rangle$. Let $\rho = \lfloor 2^l/d \rfloor$. A vertex is mapped to color i if its label is in $C_i = \{(i-1)\rho, \ldots, i\rho - 1\}$. Note that every color has ρ values associated with it. Vertices with values in the range $d\rho$ to $2^l - 1$ are uncolored. Note that fewer than d of the 2^l possible values yield an uncolored node.

Now for the analysis. We define a function, $G(Y)$, which is the sum of terms $g_e(Y)$, one for each $e \in \mathscr{E}$. Each $g_e(Y)$ is 1 if the vertices of edge e are assigned d different colors, and 0, otherwise. This is similar to the notion of the *benefit function* employed by Karp and Wigderson [13], and then Luby [16], and the *pessimistic estimator* used by Raghavan [20].

In calculating the expected value of $g_e(Y)$, we get

$$\begin{aligned} E[g_e(Y)] &\geq \Pr[g_e(Y) = 1 \mid \text{all vertices on edge } e \text{ properly colored}] \\ &\quad \times \Pr[\text{all vertices on edge } e \text{ properly colored}] \\ &\geq \frac{d!}{d^d}\left(1 - \frac{d^2}{2^l}\right). \end{aligned}$$

Therefore,

$$\begin{aligned} E[G(Y)] &= \mid \mathscr{E} \mid \frac{d!}{d^d} - \frac{\mid \mathscr{E} \mid d!d^2}{d^d 2^l} \\ &> \frac{\mid \mathscr{E} \mid d! - 1}{d^d} \qquad \left(\text{since } l > \log(2 \mid \mathscr{E} \mid d!d^2)\right). \end{aligned}$$

Then, since $G(Y)$ is integral, $G(Y) \geq E[G(Y)]$ implies that $G(Y) \geq \mid \mathscr{E} \mid d!/d^d$, which is exactly what is desired.

4.2. THE BASIC APPROACH. In this section, we discuss various approaches for determinizing algorithms that use multivalued random variables. These approaches have different advantages and disadvantages and may all prove useful in applications.

The easiest approach to handling functions of multivalued random variables is to represent each variable by a collection of Boolean random variables. In particular, for the large d-partite subhypergraph problem, if d is a power of 2, $d = O(\log \mid \mathscr{E} \mid / \log\log \mid \mathscr{E} \mid)$, we can represent the color of each vertex by $\lg d$ Boolean random variables. Then, each g_e would become a function of $d \lg d = O(\log \mid \mathscr{E} \mid)$ Boolean variables, allowing us to apply the general framework of Section 3 to find a good coloring.

A second approach we might consider would be to replace Z_2 in our distribution with some other finite field $GF(q)$. For the large d-partite subhypergraph problem, if d is any prime power, $d = O(\log \mid \mathscr{E} \mid / \log\log \mid \mathscr{E} \mid)$, we can replace Z_2 with $GF(d)$. Theorem 2.8 will still hold and all of the approaches to get labels can be easily modified to work, giving us a distribution with $(\log n)$-wise independent random variables uniformly distributed over $GF(d)$. Since d is small (we only require polynomial in n), we can try each possible value for the next element of ω in parallel and pick the one with the best conditional expected benefit G. To evaluate the conditional expectations, we can still use the linear algebra method of Section 3.1 to find the probability a collection of d random variables take on some particular value. We can do this for each possible value, since d^d is polynomial in n. Thus, we can still efficiently zero in on a good sample point.

To find a good coloring for any d up to $\log|\mathscr{E}|$, we must use a more complicated approach, one which is similar to the one used by Luby for $\Delta + 1$ vertex coloring [16]. In essence, we repeatedly use the 0/1 problem as a subroutine to set one bit of the random variables at a time. We have multivalued random variables $Y = \langle Y_1, \dots, Y_{|V|} \rangle$ where $Y_i = Y_{i1}Y_{i2} \cdots Y_{il}$. We compute the Y_i's bit by bit. At step t, we compute $X^{(t)}$ such that

$$E\left[G(Y) \mid Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t\right]$$
$$\ge E\left[G(Y) \wedge Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t-1\right].$$

If we let

$$F^{(t)}(X^{(t)}) = E\left[G(Y) \mid Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t\right],$$

then the above is equivalent to finding an $X^{(t)}$ with $F^{(t)}(X^{(t)}) \ge E[F^{(t)}(X^{(t)})]$. Letting

$$f_e^{(t)}(X^{(t)}) = E\left[g_e(Y) \mid Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t\right]$$

allows us to write $F^{(t)}(X^{(t)})$ as a sum of $|\mathscr{E}|$ functions, each depending on at most $d \le \lg|\mathscr{E}|$ random variables $X_i^{(t)}$. Assuming that, given $X^{(1)}, \dots, X^{(t-1)}$, we can construct functions $f_e^{(t)}$ (we show how to do this in the next section), we can find a good $X^{(t)}$ using the general framework of Section 3.1.

A simple inductive argument shows that for all t,

$$E\left[G(Y) \mid Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t\right] \ge E\left[G(Y)\right].$$

It follows that letting Y be such that $Y_{ij} = X_i^{(j)}$ for all i and j implies that $G(Y) \ge E[G(Y)]$.

4.3. The Deterministic Algorithm. To apply the last multivalued approach described in the previous section, we must show how to construct, for any t and for any settings of the first $t-1$ bits $X^{(1)}, \dots, X^{(t-1)}$, functions

$$f_e^{(t)}(X^{(t)}) = E\left[g_e(Y) \mid Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t\right].$$

To do so, we show how to compute

$$E\left[g_e(Y) \mid Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t\right];$$

it then suffices to plug in the given $X^{(1)}, \dots, X^{(t-1)}$ and every possible setting of the variables $\{X_i^{(t)} \mid i \in e\}$ to construct $f_e^{(t)}$.

Given edge e and the first t bits of each label, $f_e^{(t)}$ is the probability that the edge is properly colored. To calculate $f_e^{(t)}$, we sort the vertices of edge e into groups having the same t-bit prefix. For each t bit string α, we let S_α be the set of vertices that have prefix α and let I_α be the set of 2^{l-t} values that have prefix α. We let T_α be $\{1 + \sum_{\beta<\alpha}|S_\beta|, \dots, \sum_{\beta\le\alpha}|S_\beta|\}$. Observe that edge e is properly colored if and only if for each α the vertices in S_α are assigned the colors in T_α.

Now we can calculate $f_e^{(t)}$ as follows:

$$\begin{aligned} f_e^{(t)}(X^{(t)}) &= \prod_{|\alpha|=t} \Pr[\text{vertices in } S_\alpha \text{ are assigned colors in } T_\alpha] \\ &= \prod_{|\alpha|=t} |S_\alpha|! \prod_{i \in T_\alpha} \Pr[\text{vertex in } S_\alpha \text{ gets color } i] \\ &= \prod_{|\alpha|=t} |S_\alpha|! \prod_{i \in T_\alpha} \frac{|C_i \cap I_\alpha|}{2^{l-t}}. \end{aligned}$$

THEOREM 4.1. *The large d-partite subhypergraph problem, finding a coloring of V that properly colors at least* $|\mathscr{E}|\, d!/d^d$ *edges, is in NC.*

5. *Improved Discrepancy Algorithms and Bounds*

The techniques and results for discrepancy-related problems can be improved in several ways. For example, in this section we show how, in *NC*, to bound the discrepancy of each set in terms of its size (rather than the size of the largest set), solve weighted discrepancy, and match the sequential discrepancy bound when Δ is polylogarithmic. These improvements have additional applications such as lattice approximation.

5.1. IMPROVED DISCREPANCY BOUND FOR VARIABLE-SIZED SETS. In Section 2, we bounded discrepancy in terms of Δ, the maximum cardinality of any set $A \in \mathscr{A}$. Here, on the other hand, we achieve, for each $A \in \mathscr{A}$, a similar bound on $\chi(A)$ in terms of $|A|$. To do so, it is more convenient to reformulate the discrepancy problem as follows: Given vectors $v_1, \ldots, v_n \in \{0,1\}^n$, find an $x \in \{-1, +1\}^n$ such that for all i, $|v_i \cdot x| \le \Delta^{1/2+\epsilon}\sqrt{\log n}$, where $\Delta = \max_i \|v_i\|_1$. As this is equivalent to set discrepancy, the algorithm presented in Section 2 can be easily recast to solve this problem. The existing algorithm will in fact work even if we allow the v_i's to be in $\{-1, 0, +1\}^n$—the key observation being that Lemma 2.5 will still hold.

Whereas the discrepancy bound in Section 2 is achieved in terms of the maximum L_1-norm, here we achieve the bound in terms of the L_1-norm of each vector.

THEOREM 5.1. *Given vectors* $v_1, \ldots, v_n \in \{-1, 0, +1\}^n$, *there exists an NC algorithm to find an* $x \in \{-1, +1\}^n$ *such that for all* i, $|v_i \cdot x| \le \|v_i\|_1^{1/2+\epsilon}\sqrt{\log n}$.

PROOF. We modify our discrepancy algorithm as follows: We let

$$F(X) = \sum_{i=1}^{n} \frac{(v_i \cdot x)^{k_i}}{E[(v_i \cdot x)^{k_i}]},$$

where

$$k_i = 2\left\lceil \frac{\log n}{2\epsilon \log \|v_i\|_1} \right\rceil.$$

Then, getting $F(X) \leq E[F(X)] = n$ implies $(v_i \cdot x)^{k_i} \leq nE[(v_i \cdot x)^{k_i}] \leq n(k_i \| v_i \|_1)^{k_i/2}$. This, in turn, implies $| v_i \cdot x | \leq \| v_i \|_1^{1/2+\epsilon} \sqrt{\log n}$. □

5.2. Weighted Discrepancy. Still adhering to the notation introduced in the previous section, we now consider the *weighted discrepancy* problem, that is, the case when the entries of the v_i's are arbitrary real weights between -1 and $+1$. Spencer [21, 22] provides a polynomial-time algorithm for this problem that has the same performance as the polynomial-time unweighted case. To obtain an *NC* algorithm, we reduce weighted discrepancy to the unweighted case we considered above.

Theorem 5.2. *Given vectors $v_1, \ldots, v_n \in \mathbf{R}^n$, $\| v_i \|_\infty \leq 1$, there exists an NC algorithm to find an $x \in \{-1, +1\}^n$ such that for all i, $| v_i \cdot x | \leq O(\| v_i \|_1^{1/2+\epsilon} \sqrt{\log n})$.*

Proof. Without loss of generality, we can assume the v_i's are normalized so that $\| v_i \|_\infty = 1$ for all i. We can then round each entry to $\log n$ bits. This will induce a total error of at most 1, which is negligible since $\| v_i \|_1 \geq 1$. Next, by taking the binary expansions, we replace each v_i with $\log n$ vectors $v_{i0}, \ldots, v_{i, \log n - 1} \in \{-1, 0, +1\}^n$ such that $v_i = \sum_{j=0}^{\log n - 1} 2^{-j} v_{ij}$ and $\| v_i \|_1 = \sum_{j=0}^{\log n - 1} 2^{-j} \| v_{ij} \|_1$. Finally, we apply the discrepancy algorithm of Theorem 5.1 to the v_{ij}'s (with the $n \log n$ vectors padded out with zeroes to length $n \log n$). The x returned is such that for all i

$$\begin{aligned}
| v_i \cdot x | &= \Big| \sum_j 2^{-j} (v_{ij} \cdot x) \Big| \\
&\leq \sum_j 2^{-j} | v_{ij} \cdot x | \\
&\leq \sum_j 2^{-j} \| v_{ij} \|_1^{1/2+\epsilon} \sqrt{\log(n \log n)} \\
&= O\Big(\sqrt{\log n} \sum_j (2^{1/2-\epsilon})^{-j} \big(2^{-j} \| v_{ij} \|_1\big)^{1/2+\epsilon} \Big) \\
&\leq O\Big(\sqrt{\log n} \sum_j (2^{1/2-\epsilon})^{-j} \| v_i \|_1^{1/2+\epsilon} \Big) \\
&= O\big(\| v_i \|_1^{1/2+\epsilon} \sqrt{\log n} \big).
\end{aligned}$$

□

An application of weighted discrepancy is *lattice approximation*: Given vectors $v_1, \ldots, v_n \in [-1, 1]^n$ and $x \in [0, 1]^n$, find an $\hat{x} \in \{0, 1\}^n$ such that for all i, $| v_i \cdot (x - \hat{x}) |$ is small. Several researchers provide polynomial-time algorithms for this problem. Beck and Fiala [3] and Raghavan [20] use the method of conditional probabilities to construct algorithms that find an $\hat{x}$ such that every $| v_i \cdot (x - \hat{x}) |$ is bounded by $O(\sqrt{n \log n})$ and $O(\sqrt{\| v_i \|_1 \log n})$, respectively. Beck and Spencer [4] (see also [22, pp. 40–42]) show how to reduce the lattice approximation problem to the discrepancy problem, obtaining an algorithm that outputs an $\hat{x}$ such that, for all i, $| v_i \cdot (x - \hat{x}) | \leq O(\sqrt{i} \log i)$. Motwani et al. [18] apply the Beck–Spencer reduction to get an

NC algorithm for the special case where $v_1, \ldots, v_n \in \{0,1\}^n$; their algorithm outputs an $\hat{x}$ such that $|v_i \cdot (x - \hat{x})| \leq O(\Delta^{1/2+\epsilon}\sqrt{\log n})$, where $\Delta = \max_{i=1}^{n} \|v_i\|_1$. Using the algorithm of Theorem 5.2 for the more general case of weighted discrepancy, the Beck–Spencer reduction can be immediately applied to obtain NC lattice approximation in its most general form.

COROLLARY 5.3. *Given vectors $v_1, \ldots, v_n \in [-1,1]^n$ and $x \in [0,1]^n$, there exists an NC algorithm to find an $\hat{x} \in \{0,1\}^n$ such that for all i, $|v_i \cdot (x - \hat{x})| \leq O(\|v_i\|^{1/2+\epsilon}\sqrt{\log n})$.*

5.3. AN $O(\sqrt{\Delta \log n})$ DISCREPANCY ALGORITHM FOR SMALL Δ. This discrepancy algorithm of Section 2 can be improved to yield a $2\sqrt{\Delta \log n}$ bound in the special case where $\Delta = \log^c n$. The improved algorithm, besides achieving a better discrepancy bound, is a nice example of how to apply the techniques of this paper.

THEOREM 5.4. *There exists a parallel algorithm using $O(\Delta^2 \log^3 n)$ time on $O(n^2 \log^c n)$ processors, which, given a set system $\mathcal{A}$ with maximum degree Δ, outputs an χ with $disc(\chi) \leq 2\sqrt{\Delta \ln n}$.*

PROOF. For each $A \in \mathcal{A}$, we let $g_A(X)$ be 0 if $|\chi(A)| \leq 2\sqrt{\Delta \ln n}$, and 1 otherwise. Let $G(X) = \sum_{A \in \mathcal{A}} g_A(X)$, that is, $G(X)$ is the number of unbalanced sets. We want to find an X such that $G(X) \leq E[G(X)] < 1$. To do this, we use an approach similar to the one in Section 4 for multivalues. We first partition Γ (i.e., the underlying points) into $r = O(\Delta^2)$ subsets $\Gamma_1, \ldots, \Gamma_r$ such that the intersection of each Γ_j with any $A \in \mathcal{A}$ is less than $\log n/\log \Delta$ (we show how to do this below). Then, for each j in sequence, we construct a function $F^{(j)}(X^{(j)})$, where $X^{(j)}$ are the random variables corresponding to Γ_j, which is the expected value of $G(X)$ conditioned upon the values of $X^{(1)}, \ldots, X^{(j-1)}$ set already and the given $X^{(j)}$. Each $F^{(j)}$ is a sum of functions depending on at most $\log n/\log \Delta$ variables each, so we can apply our general framework to find a good $X^{(j)}$. A simple inductive argument shows that when we are done, we have a good X.

It remains to show how to construct $\Gamma_1, \ldots, \Gamma_r$ such that the intersection of each Γ_j with any $A \in \mathcal{A}$ is less than $\log n/\log \Delta$. We think of coloring the elements of Γ with $O(\Delta^2)$ colors such that no $A \in \mathcal{A}$ has $\log n/\log \Delta$ or more elements of any one color. Equivalently, we want to color so that no $(\log n/\log \Delta)$-subset of any $A \in \mathcal{A}$ is monochromatic.

To accomplish this, we let $Y = \langle Y_1, \ldots, Y_n \rangle$, where Y_i is a random variable taking on values $1, \ldots, \Delta^2$ uniformly. Let

$$H(Y) = \sum_{A \in \mathcal{A}} \sum_{\substack{B \subseteq A \\ |B| = \log n/\log \Delta}} h_B(Y),$$

where $h_B(Y)$ is 1 if B is monochromatic, and 0 otherwise. Note that $H(Y)$ represents the number of monochromatic $(\log n/\log \Delta)$-subsets, when coloring Γ according to Y.

We first show how to get a setting of Y such that at most Δ^2 of the subsets are monochromatic, and then we eliminate this entirely. We begin by computing $E[H(Y)]$. We note that, by linearity of expected value,

$$\begin{aligned} E[H(Y)] &= \sum_{A \in \mathcal{A}} \sum_{\substack{B \subseteq A \\ |B| = \log n / \log \Delta}} E[h_B(Y)] \\ &\leq n \binom{\Delta}{\log n / \log \Delta} (\Delta^2)^{1 - \log n / \log \Delta} \\ &\leq n^2 \frac{\Delta^2}{n^2} \\ &= \Delta^2. \end{aligned}$$

Note that $H(Y)$ is a sum of terms depending on $\log n / \log \Delta$ Y_i's each, and that each Y_i can be represented as $2 \log \Delta$ binary random variables. Thus, we can apply our general framework to find a Y with $H(Y) \leq \Delta^2$. Alternatively, we can set the Y_i's one bit at a time calling the general framework as a subroutine, as in Section 4.2. The latter approach is more processor efficient.

Now we have a coloring such that at most Δ^2 of the subsets are monochromatic. We take one element from each of the monochromatic ($\log n / \log \Delta$)-subsets and give each a new color. This adds at most Δ^2 additional colors, and leaves no monochromatic ($\log n / \log \Delta$)-subsets.

This discrepancy algorithm makes $\Delta^2 + 2 \log \Delta$ calls to our general framework (assuming one call for each bit of the Y_i's). Thus, the running time is $O(\Delta^2 \log^3 n)$. The number of processors is dominated by the partitioning phase, which can be done using $O(n^2 \log^c n)$ processors. □

COROLLARY 5.5. *There exists an NC algorithm that, given a set system $\mathcal{A}$ with maximum degree $\Delta \leq \log^c n$, outputs an χ with disc(χ) $\leq 2\sqrt{\Delta \ln n}$.*

ACKNOWLEDGMENTS. We would like to thank David Shmoys for suggesting parallel edge coloring, and Joel Spencer for suggesting parallel set discrepancy. We would also like to thank Mihir Bellare for suggesting a more elegant proof of Lemma 2.5. We would especially like to thank Tom Leighton and Silvio Micali for helpful discussions.

REFERENCES

1. ALON, N., BABAI, L., AND ITAI, A. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algor. 7* (1987), 567–583.
2. AWERBUCH, B., ISRAELI, A., AND SHILOACH, Y. Finding Euler circuits in logarithmic parallel time. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing* (Washington, D.C., Apr. 30–May 2). ACM, New York, 1984, pp. 249–257.
3. BECK, J., AND FIALA, T. Integral approximation sequences. *Math. Prog. 30* (1984), 88–98.
4. BECK, J. AND SPENCER, J. Interger-making theorems. *Disc. Appl. Math. 3* (1981), 1–8.
5. BERGER, B. Data structures for removing randomness. Tech. Rep. MIT/LCS/TR-436. Laboratory for Computer Science, MIT, Cambridge, Mass., Dec. 1988.
6. BERGER, B. Using randomness to design efficient deterministic algorithms. PhD dissertation. Dept. Elect. Eng. Comput. Sci., MIT, Cambridge, Mass., May 1990.
7. CHOR, B., GOLDREICH, O., HASTAD, J., FRIEDMAN, J., RUDICH, S., AND SMOLENSKY, R. The bit extraction problem or t-resilient functions. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science* (Oct.). IEEE, New York, 1985, pp. 396–407.
8. COLE, R., AND HOPCROFT, J. On edge coloring bipartite graphs. *SIAM J. Comput. 11* (1982), 540–546.
9. ERDÖS, P., AND KLEITMAN, D. On coloring graphs to maximize the proportion of multi-colored k-edges. *J. Combin. Theory 5*, 2 (Sept. 1968), 164–169.

9a. ERDÖS, P., AND SELFRIDGE, J. L. On a combinatorial game. *J. Combin. Theory B14,* (1973), 298–301.
10. GABOW, H. N., AND KARIV, O. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM J. Comput. 11* (1982), 117–129.
11. KAHN, J., KALAI, G., AND LINIAL, N. The influence of variables on Boolean functions. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science* (Oct.). IEEE, New York, 1988, pp. 68–80.
12. KARLOFF, H. J., AND SHMOYS, D. B. Efficient parallel algorithms for edge-coloring problems. *J. Algor. 8* (1987), 39–52.
13. KARP, R. M., AND WIGDERSON, A. A fast parallel algorithm for the maximal independent set problem. *J. ACM 32*, 4 (Oct. 1985), 762–773.
14. LEV, G. F., PIPPENGER, N. AND VALIANT, L. G. A fast parallel algorithm for routing in permutation networks. *IEEE Trans. Comput. 30* (1981), 93–100.
15. LUBY, M. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput 15*, 4 (Nov. 1986), 1036–1053.
16. LUBY, M. Removing randomness in parallel computation without a processor penalty. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science* (Oct.). IEEE, New York, 1988, pp. 162–173.
17. MOTWANI, R., NAOR, J., AND NAOR, M. A generalized technique for derandomizing parallel algorithms. Unpublished manuscript, Jan. 1989.
18. MOTWANI, R., NAOR, J., AND NAOR, M. The probabilistic method yields deterministic parallel algorithms. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (Oct.). IEEE, New York, 1989, pp. 8–13
19. RABIN, M. O. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM 36*, 2 (Apr. 1988), 335–348.
20. RAGHAVAN, P. Probabilistic construction of deterministic algorithms. Approximating packing integer programs. *J. Comput Syst. Sci. 37*, 4 (Oct. 1988), 130–143.
21. SPENCER, J Balancing games. *J. Combin. Theory, B 23* (1977), 68–74.
22. SPENCER, J. *Ten Lectures on the Probabilistic Method.* SIAM, Philadelphia, Pa , 1987.
23. VIZING, V. G. On the estimate of the chromatic class of a P-graph. *Diskret. Anal. 3* (1964), 25–30 (in Russian).

RECEIVED FEBRUARY 1989; REVISED FEBRUARY 1990; ACCEPTED JUNE 1990