Improved Algorithms via Approximations of Probability Distributions*

Suresh Chari† Pankaj Rohatgi‡ Aravind Srinivasan§

Abstract

We present two techniques for approximating probability distributions. The first is a simple method for constructing the small-bias probability spaces introduced by Naor & Naor. We show how to efficiently combine this construction with the *method of conditional probabilities* to yield improved NC algorithms for many problems such as set discrepancy, finding large cuts in graphs, finding large acyclic subgraphs etc. The second is a construction of small probability spaces approximating general independent distributions, which is of smaller size than the constructions of Even, Goldreich, Luby, Nisan & Veličković. Such approximations are useful, e.g., for the derandomization of certain randomized algorithms.

Keywords. Derandomization, parallel algorithms, discrepancy, graph coloring, small sample spaces, explicit constructions.

1 Introduction

Derandomization, the development of general tools to derive efficient *deterministic* algorithms from their randomized counterparts, has blossomed greatly in the last decade. The best-known deterministic solutions for several classical problems such as testing undirected graph connectivity within limited space bounds, take this approach [24]. Two motivations for research in this area

*A preliminary version of this work appeared as part of a paper of the same title in the *Proc. ACM Symposium on the Theory of Computing*, pages 584–592, 1994.

†Dept. of Computer Science, Cornell University, Ithaca NY 14853, USA. Supported in part by NSF grant CCR-9123730. E-mail: `chari@dahlgren-19.eng.sun.com`.

‡IBM T. J. Watson Research Center, Hawthorne, NY 10532, USA. Work done while at Cornell University, supported in part by NSF grant CCR-9123730. E-mail: `rohatgi@watson.ibm.com`.

§Department of Information Systems and Computer Science, National University of Singapore, Lower Kent Ridge Road, Singapore 119260, Republic of Singapore. Parts of this work were done at: (i) the Dept. of Computer Science, Cornell University, Ithaca, NY 14853, USA, supported by an IBM Graduate Fellowship; (ii) the School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540, USA, supported by grant 93-6-6 of the Alfred P. Sloan Foundation to the Institute for Advanced Study; (iii) DIMACS (NSF Center for Discrete Mathematics and Theoretical Computer Science), supported by NSF grant NSF-STC91-19999 to DIMACS and by support to DIMACS from the New Jersey Commission on Science and Technology, and (iv) the National University of Singapore. E-mail: `aravind@iscs.nus.sg`.

are the non-availability of "perfect" random sources, and the need for absolute (non-probabilistic) guarantees of correctness in, say, critical applications. The fact that computers do not use "real" random sources prevents randomized algorithms from having a sound footing; it has been shown that if algorithms such as randomized Quicksort are not implemented carefully when used with some existing pseudorandom generators, their expected running times can be high [17]. There have also been reports of Monte-Carlo simulations giving quite different results under different random-number generators [12], and direct implementations of certain RNC algorithms taking longer time than expected due to the pseudorandom nature of computer-generated "random" bits [15, 14]. In this paper, we present two new techniques for derandomization. The first leads to improved NC algorithms for many basic problems such as finding large cuts in graphs, set discrepancy, $(\Delta + 1)$-vertex coloring of graphs and others while the second improves the constructions due to [11] of small sample spaces for use in derandomization. We also show how the first method could potentially be used to solve some other open problems in parallel computation. The second method yields smaller sample spaces for the efficient derandomization of randomized algorithms.

Three major known approaches to derandomization are the techniques of limited independence ([18, 19, 1]), the method of conditional probabilities ([26, 29]), and small-bias probability spaces ([22, 3, 2]). We now discuss these methods briefly. Let the set $\{1, 2, \ldots, n\}$ be denoted by $[n]$.

Definition 1 *Random variables $X_1, X_2, \ldots, X_n$ are k–wise independent if for any set $I \subseteq [n]$ of at most k indices and for any choice of $v_1, v_2, \ldots,$ we have $Pr(\bigwedge_{i \in I}(X_i = v_i)) = \prod_{i \in I} Pr(X_i = v_i)$.*

Efficient constructions of sample spaces $S \subseteq \{0, 1\}^n$ of size usually much smaller than 2^n, such that the distribution induced on $\{0, 1\}^n$ by sampling uniformly at random from S is k–wise independent, are given, e.g., in [16, 19, 1]. The idea here is to analyze a given randomized algorithm and to show that its behavior is good enough if the X_is are k-wise independent for a suitably large $k = k(n)$, rather than completely independent. One can then search over all points in k-wise independent sample space and *deterministically* output a good sample point for the randomized algorithm.

However, if k is too large–in particular if k is not bounded but grows with n, then in most cases, k-wise independent sample spaces for the X_is have superpolynomial (in n) size: too large for efficient exhaustive search. Motivated by the fact that randomized algorithms are usually robust to small changes in the probabilities, Naor & Naor observed that it often suffices if the sample space is *almost k-wise independent* [22], captured by

Definition 2 *(a) A sample space $S \subseteq \{0, 1\}^n$ is said to be ϵ-approximate if when $\vec{X} = (X_1, \ldots, X_n)$ is sampled uniformly from S, for all $I \subseteq [n]$ and for all $b_1, \ldots, b_{|I|} \in \{0, 1\}$ we have $|Pr(\bigwedge_{i \in I}(X_i = b_i)) - 2^{-|I|}| \leq \epsilon$. (b) S is called k-wise ϵ-approximate, or roughly as "almost k-wise independent", if the inequality in (a) holds for all I such that $|I| \leq k$.*

The important work of [22] actually presents efficient constructions of "ϵ-biased" sample spaces $S \subseteq \{0, 1\}^n$ (to be defined in §2), and shows that they are ϵ-approximate; thus we have efficient constructions of almost k-wise independent spaces that are often much smaller than their k-wise independent counterparts. Small-bias spaces have since become very useful in derandomization, as several randomized algorithms are robust to such small changes in the probabilities. For instance, another major derandomization approach, which uses *dispersers* and *extractors* that we do not define here, also uses small-bias spaces: see, e.g., the survey of Nisan [23].

A third key approach to derandomization, the method of conditional probabilities, can be described informally as follows. Suppose a randomized algorithm chooses n random bits $\vec{X} = (X_1, \ldots, X_n)$ and outputs a function $f(\vec{X})$. Typically, by analysis we can show that $E(f(\vec{X}))$ is "large". By definition, there exists $\vec{X}_0$ such that $f(\vec{X}_0) \geq E(f(\vec{X}))$. The goal is to find such a point $\vec{X}_0$; this "deterministic sample" can then be used to make the algorithm deterministic. The method of conditional probabilities is to find this point by setting the components of $\vec{X}_0$ one bit at a time deterministically. We do this so that at every stage, the expectation of $f(\vec{X})$ conditioned on fixing the values of the components up to now is non–decreasing. This ensures that when we have assigned all n components we have found an $\vec{X}_0$ such that $f(\vec{X}_0) \geq E(f(\vec{X}))$.

In [20], Luby observed that using limited independence directly in some parallel algorithms leads to a high processor complexity, and introduced a way of combining limited independence with the method of conditional probabilities which led to processor-efficient algorithms. His method has been used and extended to derive NC algorithms for fundamental problems such as set discrepancy [8, 21].

The motivation for our first method is similar to that of [20]. We observe that for a large class of problems, direct use of small-bias spaces leads to inefficient NC algorithms, and present a way to combine them with the method of conditional probabilities to get significantly better NC algorithms. Using this we obtain improved NC algorithms for many basic problems such as finding large cuts in graphs, set discrepancy, $(\Delta + 1)$-vertex coloring of graphs and others (here and from now on, when given an undirected graph G, Δ will denote its maximum degree). In each application, our method results in algorithms that are faster than previously known algorithms or match the running time of the fastest known algorithms with a significant reduction in the processor complexity. We assume the EREW PRAM model throughout, and to describe our results we denote an $O(p(n))$ processor, $O(t(n))$ time *deterministic* parallel algorithm by a $(p(n), t(n))$ algorithm.

The first application of our technique is to the classical *set discrepancy* (or *set balancing*) problem:

Definition 3 (Set Balancing): *Given a ground set X and subsets $S_1, \ldots, S_n$ of X where $|X| \leq n$ and $|S_i| \leq s$ for each i, find an assignment $\chi : X \to \{0,1\}$ such that $disc(\chi) \doteq \max_i |\sum_{j \in S_i} \chi(j) - \frac{|S_i|}{2}|$ is "small".*

Spencer [29] gave a polynomial–time algorithm to find an assignment χ with $disc(\chi) = O(\sqrt{s \log n})$. NC^3 algorithms to find an assignment χ with $disc(\chi) = O(s^{1/2+\delta}\sqrt{\log n})$, for any fixed $\delta > 0$, were first given in [8, 21] and small-bias spaces were used to do this in NC^1 [22], but with $\Omega(n^{3+2/\delta})$ processors. We derive an NC^1 algorithm with processor complexity $O(n^{1+\delta'+2/\delta})$ for any fixed $\delta' > 0$ which is an improvement of $\Omega(n^{2-\delta'})$ in the processor complexity. This yields improved algorithms for problems like vector–balancing [28], lattice approximation [26], and computing ϵ-nets and ϵ-approximations for range spaces with finite VC-dimension [10].

Set balancing is a special case of the following general problem. Let $X_1, X_2, \ldots, X_n$ be unbiased and independent random bits, and let $X = \sum_{i=1}^{n^a} f_i(X_{i,1}, X_{i,2}, \ldots, X_{i,b \log n})$ where a and b are constants. We wish to find a sample point $X_0 \in \{0,1\}^n$ deterministically such that $X_0 \geq$

$E[X]$. This general framework, which models the derandomization of many RNC algorithms, was introduced in [8] (see also [21]), where an $(n^{a+b}, t\log^3 n)$ algorithm is given (here we assume that each f_i is computable in $O(t)$ time with one processor). In most cases (e.g., set discrepancy), it suffices to find a sample point X_0 such that $X_0 \geq E[X] - n^{-\Theta(1)}$. In these cases our method yields an $O(n^{a+b+\delta}, t\log n)$ algorithm for any fixed $\delta > 0$, or at the other end of the spectrum, an $(n^{a+b}\log^2 n, t\log^2 n)$ algorithm. Small-bias spaces can be used here to obtain $O(t\log n)$ time NC algorithms, but the processor complexity would be $\Omega(n^{3a+2b}\log^2 n)$.

The problem of finding a heavy codeword, which generalizes the classical problem of finding a large cut in a graph, was introduced in [22]. Given a matrix $A \in Z_2^{m\times n}$, no row of which has only zeroes, the problem is to find an $x \in Z_2^n$ with Ax (over Z_2) having at least $\frac{m}{2}$ ones. Small-bias sample spaces are used in [22] to obtain a $(\min\{m^4n^2, m^3n^3\}, \log(m+n))$ algorithm, thus placing the problem in NC. Our method directly yields a range of better NC algorithms for this problem with a continuous processor–time tradeoff: the algorithm with the best processor-complexity is an $(n^2m, \log(n+m)\log m)$ algorithm, and the best time-complexity algorithm is an $(n^2m^{1+\delta}, \log(m+n))$ algorithm, for any fixed $\delta > 0$. (Note the significant improvement in the processor complexity, even for this latter algorithm that runs in $O(\log(m+n))$ time.) The problem of finding a *large cut* in a graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, is to find a cut in G with at least $\frac{m}{2}$ cut edges; this well-known problem is a special case of the heavy codeword problem. NC algorithms of complexity $((n+m), \log^2 n)$ are known for this problem (see, e.g., [20]). Small-bias spaces can be used to obtain an NC^1 algorithm but with a processor complexity of $\Theta(m^3\log^2 n)$. Similarly, direct use of pairwise independent sample spaces leads to an NC^1 algorithm, but with a processor complexity of mn. In contrast, our method yields an $((m+n)n^\delta, \log n)$ algorithm for this problem, for any fixed $\delta > 0$.

We combine our method with some ideas of Alon & Naor [4] to derive improved algorithms for some other basic problems. Given a graph $G = (V, E)$ with $|V| = n$, $|E| = m$ and having a maximum degree Δ, a $(\Delta + 1)$-vertex coloring of the graph can be computed by an $((n + m), \log^3 n \log\log n)$ algorithm [20]; an improved $((n+m)/\log\log n, \log^2 n\log\log n)$ algorithm that runs on the CREW PRAM, has been devised by Han [13]. We improve these time complexities by presenting an $((n+m)n^\delta, \log^2 n)$ algorithm where $\delta > 0$ is any constant. This gives faster algorithms for other problems such as the much harder Δ-vertex coloring problem [25]. We use similar ideas to derive a faster algorithm for approximating the maximum acyclic subgraph problem [9], incurring a small processor penalty.

Thus, our first method yields the fastest known NC algorithms for a large class of problems, without a big processor penalty. Though decreasing the running time at the expense of an increased processor complexity may be viewed as impractical, we view some of these results as progress toward pinpointing the best time complexity possible for such problems, while still having a relatively small (and certainly polynomial) processor complexity; for problems (such as maximal acyclic subgraph) for which we present the first NC^1 algorithms, note that the running time is optimal for any NC algorithm. Though our results hold for the PRAM, they use only basic operations like simple arithmetic over $GF[2^t]$ (where $t = O(\log n)$) and over $\Re$ and comparison, and we hope that they can be employed in realistic parallel networks. Our method also yields a potential approach to some open problems, where the conditional estimator is a sum (of a certain type) of $n^{O(\mathrm{polylog}(n))}$ terms, each depending on $O(\log n)$ unbiased and independent random bits. A key example of this arises in solving set discrepancy in NC, with discrepancy $O(\sqrt{s \log n})$–a challenging open question [8, 21]. We show that if a certain small-bias space S is efficiently constructible, then our method can solve the above class of problems in NC. This would be interesting since most current methods cannot tackle a superpolynomial number of terms. However, we do not yet know if S exists.

We now turn to the second type of derandomization technique presented. Given the utility of small spaces approximating the joint distribution of unbiased and independent random bits, the general problem of approximating *arbitrary* independent distributions was addressed in [11]. If $X_1, \ldots, X_n \in \{0, 1, \ldots, m-1\}$ are *independent* random variables with *arbitrary* individual distributions and joint distribution D, a sample space S is a (k, ϵ)-*approximation* for D if, for $\vec{Y} = (Y_1, \ldots, Y_n)$ sampled uniformly from S, for all index sets $I \subseteq \{1, \ldots, n\}$ with $|I| \leq k$, and for all $a_1, \ldots, a_{|I|} \in \{0, 1, \ldots, m-1\}$,

$$|Pr(\wedge_{i \in I}(Y_i = a_i)) - \prod_{i \in I} Pr_D(X_i = a_i)| \leq \epsilon.$$

Such sample spaces have obvious applications to reducing randomness and derandomization. Three constructions of such small sample spaces are presented in [11], each suitable for different ranges of the input parameters. We provide a construction which is always better than or as good as all of their constructions. Our sample space is of size $\text{poly}(\log n, 1/\epsilon, (\lceil k/\log(1/\epsilon) \rceil)^{\log(1/\epsilon)})$. As in [11], we reduce this problem to a geometric discrepancy problem; our improvement follows from our improved solution for this latter problem. As pointed out in §6.1, improved solutions for this geometric discrepancy problem also yield improved methods for certain classes of numerical integration problems.

The basic idea behind our first method is presented in §2, followed by some direct applications of it, in §3. Section 4 presents some extensions of the basic method, and potential applications of the first method are shown in §5. Next in §6, we present our second derandomization tool, and §7 concludes.

2 The first method: basic ideas

Notation. For any positive integer n and for any $a, b \in \{0, 1\}^n$, let $<a, b>$ denote the dot product, over Z_2, of a and b: $<a, b> = (\sum_i a_i b_i) \bmod 2$. For a binary vector y, $wt(y)$ denotes its Hamming weight, i.e., the number of components which are 1 in the vector. Logarithms are to the base two, unless specified otherwise.

Definition 4 *The bias of a binary random variable X is* $\text{bias}(X) \doteq Pr(X = 0) - Pr(X = 1)$. *Let α be a nonzero vector in Z_2^n and let $\vec{X} = (X_1, \ldots, X_n)$ be sampled uniformly from a sample space $S \subseteq \{0, 1\}^n$. The bias of S with respect to α is* $\text{bias}(<\vec{X}, \alpha>)$ *and is denoted* $\text{bias}_S(\alpha)$. *If* $|\text{bias}_S(\alpha)|$ *is bounded in magnitude by ϵ for all nonzero α, then S is said to be ϵ-biased.*

The biases of a sample space S with respect to the vectors $\alpha \in Z_2^n$ are the Fourier coefficients of the probability distribution D_S induced on $\{0, 1\}^n$ by sampling uniformly from S [22]. The Fourier bases are the functions f_α, where $f_\alpha(x) = (-1)^{<\alpha, x>}$ for $x \in \{0, 1\}^n$. If c_α is the

Fourier coefficient with respect to α (i.e., the bias of S with respect to α), then by standard Fourier inversion,

$$D_S(x) = 2^{-n} \sum_{\alpha \in \{0,1\}^n} c_\alpha f_\alpha(x)$$

for $x \in \{0,1\}^n$. From this it can be shown that if all the Fourier coefficients of D_S are bounded in magnitude by ϵ, then the sample space is ϵ-approximate. NC^1 constructions of sample spaces of size $O(n/\epsilon^3)$ with Fourier coefficients bounded in magnitude by ϵ were developed in [2], improving on those of [22]. NC^1 constructions of three different sample spaces of size $O(n^2/\epsilon^2)$, were presented by [3].

The basic idea behind our construction of approximate sample spaces is the following simple and well-known fact:

Fact 1 *Let $X_1, \ldots, X_k$ be independent binary random variables, with $bias(X_i) = \epsilon_i$. Then, the random variable $Y_k = X_1 \oplus X_2 \oplus \ldots \oplus X_k$ has bias $\epsilon_1 \cdot \epsilon_2 \cdots \epsilon_k$.*

Proof. We supply an elementary proof for completeness. The bias of Y_k is $Pr(Y_{k-1} = 0)Pr(X_k = 0) + Pr(Y_{k-1} = 1)Pr(X_k = 1) - Pr(Y_{k-1} = 0)Pr(X_k = 1) - Pr(Y_{k-1} = 1)Pr(X_k = 0) = bias(Y_{k-1})bias(X_k)$, and hence equals $\epsilon_1 \cdot \epsilon_2 \cdots \epsilon_k$ by induction on k. $\square$

The new construction of small-bias sample spaces uses this fact as described in the following lemma.

Lemma 1 *Let S' be an ϵ'-biased sample space and let $\vec{X_1}, \ldots, \vec{X_{t(\epsilon)}}$ be sampled independently and uniformly at random from S', where $t(\epsilon) = \lceil \log(\epsilon)/\log(\epsilon') \rceil$. Define $\vec{X} = \vec{X_1} \oplus \cdots \oplus \vec{X_t}$. Then the sample space (multi-set) S of all such $\vec{X}$ is ϵ-approximate.*

Proof. Fix any nonzero $\alpha \in \{0,1\}^n$. By definition, the bias of $<X, \alpha>$ is the bias of $<\vec{X_1}, \alpha> \oplus \cdots \oplus <\vec{X_t}, \alpha>$. Since the bias of each $<\vec{X_i}, \alpha>$ is bounded in magnitude by ϵ', the lemma follows from Fact 1. $\square$

Thus our construction of sample space S is to start with a sample space S' of much larger bias and then to XOR independent samples from S'. A space constructed like this will, in general, be larger than those obtained by direct constructions of ϵ-approximate spaces. However when we actually use this method in designing deterministic algorithms, we will take the different samples from the space S' using the method of conditional probabilities. Since the base sample space S' has $|S'| \ll |S|$, this will yield more efficient algorithms since we have much fewer points to try out while derandomizing.

Since our idea is to use a known construction of small-bias sample spaces to bootstrap itself, we describe a known construction of small bias sample spaces for completeness.

Theorem 1 ([3]) *Let* $\mathrm{bin}\colon GF(2^m) \mapsto \{0,1\}^m$ *be the standard binary representation of the field of 2^m elements which satisfies* $\mathrm{bin}(0) = 0^m$ *and* $\mathrm{bin}(u+v) = \mathrm{bin}(u) \oplus \mathrm{bin}(v)$. *Given any two field elements x and y, define the n-bit string $r(x,y)$ to be $r_0(x,y) \ldots r_{n-1}(x,y)$, where $r_i(x,y) = <\mathrm{bin}(x^i), \mathrm{bin}(y)>$. The sample space consisting of the string $r(x,y)$ for all choices of x and y is $(\frac{n-1}{2^m})$–biased. Thus there is an explicit ϵ–biased sample space of size $O(n^2/\epsilon^2)$.*

Similar definitions can be made for sample spaces that are k-wise ϵ-approximate. A sample space S is said to be *k-wise ϵ-biased* if $|bias_S(\alpha)| \le \epsilon$ for all nonzero $\alpha \in Z_2^n$ with $wt(\alpha) \le k$. A k-wise ϵ-biased space is k-wise ϵ-approximate [22]. Sample spaces of size $O(\min\{\frac{k \log n}{\epsilon^3}, \frac{(k \log n)^2}{\epsilon^2}\})$ which are k-wise ϵ-biased can be constructed in NC^1 [22, 3, 2]. If S' is k-wise ϵ'-biased in Lemma 1, then the resulting space S is k-wise ϵ-biased. Thus our construction extends directly to yield k–wise ϵ–biased sample spaces.

3 Direct Applications of the First Method

Throughout this section we use $G = (V, E)$ to denote a graph on n vertices and m edges, with maximum degree Δ. $\delta, \delta', \delta'', \delta_1, \delta_2$ etc. denote arbitrarily small positive constants.

3.1 Cuts in Graphs and the heavy codeword problem

As a straightforward application of our method, we consider the *heavy codeword* problem [22]: given $A \in Z_2^{m \times n}$, find an $x \in Z_2^n$ such that Ax (over Z_2) has at least $\frac{m}{2}$ ones. In the following a_i denotes the ith row of A, and we assume that the a_i's are nonzero. As pointed out in [22], if x is picked from an ϵ-biased space S with $\epsilon = \frac{1}{2m}$, then

$$\mathrm{E}[wt(Ax)] = \sum_{i=1}^{m} Pr(<a_i, x> = 1) = \sum_{i=1}^{m} \left(\frac{1 - \mathrm{bias}(<a_i, x>)}{2} \right) \ge m\,(1/2 - \epsilon/2)$$

and thus, there exists an $x \in S$ with $wt(Ax) \ge \lceil m\,(1/2 - \epsilon/2) \rceil = \lceil m/2 \rceil$. But, an $\frac{1}{2m}$-biased space has $\Omega(\min\{m^3 n, m^2 n^2\})$ points if we use the known constructions and for each point, the obvious derandomizing algorithm needs mn processors to compute Ax. Thus, direct use of small-bias spaces results in high processor complexity. Instead, we start with an $m^{-\delta}$-biased sample space S_0 and pick x, as in §2, to be the bit-wise XOR of $\ell = \lceil (\log 2m)/(\log m^\delta) \rceil = O(1)$ many *independent* samples $Y_1, Y_2, \ldots, Y_\ell$ from S_0.

We show that the method of conditional probabilities can be used to find a "good" sequence $Y_1^*, \ldots, Y_\ell^*$ *efficiently* in NC. Assume that we have already fixed the values of $Y_1^*, \ldots, Y_{j-1}^*$. Define the conditional estimator at stage j to be

$$f_{j-1}(Y_1^*, \ldots, Y_{j-1}^*) \doteq \sum_{i=1}^{m} \frac{(1 - \mathrm{bias}(<a_i, x>|Y_k = Y_k^*, 1 \le k \le j-1))}{2}$$

We want a $Y_j^* \in S_0$ which maximizes $f_j(Y_1^*, \ldots, Y_j^*)$. We need only show that $f_j(Y_1^*, \ldots, Y_j^*)$ can be computed efficiently for all points $Y_j^* \in S_0$. Fix $Y_j^* \in S_0$, and let $b = \bigoplus_{k=1}^{j} Y_k^*$. Thus, $x = b \oplus z$, where $z = Y_{j+1} \oplus \cdots \oplus Y_\ell$ with $Y_{j+1}, \ldots, Y_\ell$ picked uniformly and independently from S_0. Thus, $\mathrm{bias}(<a_i, x>|Y_k = Y_k^*, 1 \le k \le j)$ equals $\mathrm{bias}(<a_i, z>)$ if $<a_i, b> = 0$, and $-\mathrm{bias}(<a_i, z>)$ otherwise. Note the crucial role of the independence of the different samples. From Fact 1, $\mathrm{bias}(<a_i, z>) = (\mathrm{bias}_{S_0}(a_i))^{\ell - j}$. The algorithm is described in detail in Figure 1.

The size of S_0 is $O(m^{3\delta} n)$ and hence we can precompute $\mathrm{bias}_{S_0}(a_i)$ for all the a_i's using $mn|S_0|$ processors in $O(\log n + \log m)$ time. At each of the ℓ stages we have $|S_0|$ choices for Y_j^* and

Construct a sample space S_0 which is $\epsilon' = m^{-\delta}$ biased.
For each i in parallel precompute $\text{bias}_{S_0}(a_i)$.
For j from 1 through $\ell = \lceil \frac{\log 2m}{\log m^\delta} \rceil$ do
    For each element y of S_0 in parallel do
        Compute $b = (\bigoplus_{k=1}^{j-1} Y_k^*) \oplus y$.
        For each i in parallel do
            If $< a_i, b > = 0$ then let $\text{bias}_i(y) = (\text{bias}_{S_0}(a_i))^{\ell-j}$
                else let $\text{bias}_i(y) = -(\text{bias}_{S_0}(a_i))^{\ell-j}$
    Let $Y_j^* \in S_0$ to be any $y \in S_0$ which maximizes $\sum_{i=1}^{m}(1 - \text{bias}_i(y))$
Output $\bigoplus_{k=1}^{\ell} Y_k^*$.

Figure 1: Efficient NC algorithm for the heavy codeword problem

$f_j(Y_1^*, \ldots, Y_j^*)$ can be evaluated in $O(\log(m+n))$ time using $O(mn)$ processors. Alternatively, we could choose S_0 of constant bias and the same idea (with $\ell = O(\log m)$) gives an $(n^2 m, \log(m+n)\log m)$ algorithm.

Theorem 2 *For any fixed $\delta > 0$, there are $(n^2 m^{1+\delta}, \log(m+n))$ and $(n^2 m, \log(m+n)\log m)$ algorithms for the* heavy codeword *problem.*

We reiterate the points which led to efficiency: (a) the search space S_0 has $|S_0| \ll \min\{m^3 n, m^2 n^2\}$, and (b) we directly compute a good point x, avoiding the costly matrix–vector multiplications.

Given a graph G, similar ideas work to find a cut with at least $\frac{m}{2}$ edges. If each vertex picks a bit to decide which side of the cut it lies on, and if the bits come from a 2-wise $1/(2m)$–biased sample space, then $\lceil E[\#\text{ cut edges}] \rceil \geq m/2$. Direct use of biased spaces yields an $(m^3 \log^2 n, \log n)$ algorithm. A better algorithm can be obtained by using a sample space which guarantees pairwise independence and this results in a $(mn, \log n)$ algorithm. In contrast, using our method we obtain

Theorem 3 *There is an $((n+m)\min\{(\log n)/\epsilon^3, (\log^2 n)/\epsilon^2\}, \log^2 n/\log(1/\epsilon))$ algorithm for any $\epsilon = \epsilon(n, m) < 1$, for the large cut problem. In particular (by choosing $\epsilon = n^{-\delta/3}$), the large cut problem can be solved by an $((n+m)n^\delta, \log n)$ algorithm, for any fixed $\delta > 0$.*

Proof. As for the heavy codeword problem, we imagine constructing the desired pairwise $1/(2m)$–biased space by starting with an efficiently constructed pairwise ϵ–biased space S_0 and taking the XOR of $\ell = O(\log n/\log(1/\epsilon))$ independent samples from S_0. As before, we derandomize this construction by fixing the ℓ choices–elements of S_0–one by one; each such fixing takes $O(\log n)$ time using $O(n+m)$ processors. Thus, the processor and time bounds follow. □

3.2 Applications using biases to compute probabilities

We now outline a more involved application where we use the biases of the base distribution to compute probabilities of events by Fourier inversion. Given subsets $S_1, \ldots, S_n$ of a set X where

$|X| \leq n$ and $|S_i| \leq s$ for each i, the *set discrepancy* problem is to find a $\chi : X \mapsto \{0,1\}$ such that $disc(\chi) \doteq \max_i |\chi(S_i) - \frac{|S_i|}{2}|$ is "small", where $\chi(S_i) = \sum_{j \in S_i} \chi(j)$. It is shown in [8, 21] that if we choose $k = 2\lceil \log(2n)/(\delta \log s) \rceil$ and choose the assignments $\chi(j), j \in X$, k-wise independently then

$$Pr(\ (\exists i)\ |\chi(S_i) - \frac{|S_i|}{2}| \geq s^{0.5+\delta}\sqrt{\log n}\ ) \leq \sum_{i=1}^{n} \frac{E[(\chi(S_i) - \frac{|S_i|}{2})^k]}{(s^{0.5+\delta}\sqrt{\log n})^k} < 1. \tag{1}$$

This is used to derive NC^3 algorithms which find a χ with $disc(\chi) \leq s^{0.5+\delta}\sqrt{\log n}$. It is shown in [22] that (1) holds even if the assignments $\chi(j), j \in S$ are k–wise ϵ-biased for $\epsilon < 1/(2n^{1+1/\delta})$, leading to an $(\Omega(n^{3+\frac{2}{\delta}}), \log n)$ algorithm. We significantly improve on this processor complexity while retaining the $O(\log n)$ running time, using Fourier coefficients to compute probabilities. Expanding $E[(\chi(S_i) - \frac{|S_i|}{2})^k]$ gives $O(n^{\frac{2}{\delta}})$ terms, each of the form $c \cdot Pr(X_{i_1} = \cdots = X_{i_t} = 1)$, with $t \leq k$ and c a constant. We know how to compute probabilities of events once we we know the Fourier coefficients as shown in § 2. In this case we can express the term $c \cdot Pr(X_{i_1} = \cdots = X_{i_t} = 1)$ as the sum of $2^t = n^{o(1)}$ *bias* terms, using the Fourier bases. Now exactly as in the case of the algorithm for the heavy codeword problem we can start with a k-wise $n^{-\delta''}$-biased S_0 and pick a "good" χ using $O(1)$ samples from S_0. At each stage we choose a point in S_0 which minimizes the expectation which we have expressed in terms of the biases. The number of sample points we need to try at each stage $|S_0| = n^{O(\delta'')}$ and the number of bias terms we need to consider is $n^{\frac{2}{\delta}+o(1)}$. The algorithm runs in time $O(\log n)$ and the number of processors we need for each stage is $O(n^{1+\frac{2}{\delta}+\delta'})$. Thus we can obtain the following.

Theorem 4 *There is an $(n^{1+\frac{2}{\delta}+\delta'}, \log n)$ algorithm to find a χ for the set discrepancy problem with $disc(\chi) \leq s^{0.5+\delta}\sqrt{\log n}$, for any fixed $\delta, \delta' > 0$.*

The set discrepancy abstraction has been used to solve other problems such as vector balancing [28], lattice approximation [26] (see also the description in [21]) and for computing ϵ-nets and ϵ-approximations for range spaces with finite VC–dimension [10]. Theorem 4 implies improved NC algorithms for all these problems. Our method also yields improvements in processor complexity for algorithms which find an assignment χ with $disc(\chi) = O(\sqrt{s \log n})$. We can derive a $(s^{\Theta(\log n)}, \log n)$ algorithm with the constant in the exponent of the processor complexity roughly half of what a direct implementation using regular ϵ-biased spaces can yield.

3.3 A general framework

Set discrepancy has been captured as part of a more general framework in [8], and we can extend the above method to this general framework to derive improved algorithms. Suppose $X_1, \ldots, X_n$ are unbiased and independent random bits and let $X = \sum_{i=1}^{n^a} f_i(X_{i,1}, X_{i,2}, \ldots, X_{i,b\log n})$. We wish to find a sample point X_0 with $X_0 \geq E[X]$, where a and b are constants.

An $(n^{a+b}, t\log^3 n)$ algorithm is presented for this problem in [8] if each f_i is computable in $O(t)$ time with one processor. In many cases, such as the set discrepancy problem above, if the X_is are $b\log n$-wise, ϵ-biased with $\epsilon = n^{-O(1)}$, we can find a sample point with $X \geq E[X] - n^{-\Theta(1)}$ which suffices. We can first express $E[X]$ as a sum of at most n^{a+b} bias terms, via the Fourier coefficient method used above for set discrepancy. Then, if we start with a sample space S_0

with $n^{-\delta/3}$ bias and use the same approach as for set discrepancy we can get an $(n^{a+b+\delta}, t\log n)$ algorithm. On the other extreme, if we start with a sample space which is of constant bias we can get an $(n^{a+b}\log^2 n, t\log^2 n)$ algorithm. In contrast, directly plugging in the required small-bias sample space only gives an $(\Omega(n^{3a+2b}), t\log n)$ algorithm.

As another example we can get from our improved algorithm for the general abstraction, an improved algorithm for a special case of the set discrepancy problem. If $s = \log^{O(1)} n$ in the set discrepancy problem, then an $(n^2 \log^{O(1)} n, s^2 \log^3 n)$ algorithm is given in [8] to find χ with $disc(\chi) = O(\sqrt{s\log n})$. The original algorithm uses the above general framework as a subroutine and replacing the original algorithm with ours results in the following improvement. We refer the reader to [7, 8] for details.

Theorem 5 *If $s = \log^{O(1)} n$ in the set discrepancy problem, then there is an $(n^{2+\delta}, s^2\log n)$ algorithm to find χ with $disc(\chi) = O(\sqrt{s\log n})$, for any fixed $\delta > 0$.*

4 Handling multivalued variables

In this section we consider a family of applications where the random variables of interest are uniform over $\{0,1\}^q$ for some $q > 1$. The parameter q is usually at most polylogarithmic in the input size N, in our applications. By combining our basic method with some ideas of [4] we get improved parallel algorithms for various problems. Some of these problems can also be solved using the above-seen general framework; however, the solutions we give here are better than those obtained as instances of the general framework.

4.1 The Profit/Cost problem

The first application we describe is related to the "General Pairs Benefit Problem" of [20], which was called the "General PROFIT/COST Problem" in the conference version of [20] (*Proc. IEEE Symposium on Foundations of Computer Science*, pages 162–173, 1988). This general problem, which models problems such as $(\Delta+1)$-vertex coloring of graphs [20], requires a few details that are not pertinent to our work (Δ here refers to the maximum degree of a given graph). It is shown in [20] that such details, e.g., the existence of suitable "pessimistic estimators", can be abstracted away, leading to the following problem, which we also focus upon. Our presentation is slightly different from, but essentially equivalent to, that of [20].

We are given a graph $G = (V, E)$, with $|V| = n$ and $|E| = m$. Also given are non-negative functions $Profit_v : \{0,1\}^q \to \Re^+$ and $Cost_{u,v} : \{0,1\}^q \times \{0,1\}^q \to \Re^+$, for all $v \in V$ and $\{u, v\} \in E$. The Profit and Cost functions are assumed to satisfy the following property: $\forall p \le q\ \forall u \in V\ \forall \{u,v\} \in E$ and for all $s_1, s_2 \in \{0,1\}^p$, $E_{x_1}[Profit_u(s_1 \circ x_1)]$ and $E_{x_1,x_2}[Cost_{u,v}(s_1 \circ x_1, s_2 \circ x_2)]$ can be computed in time t by one processor, where x_1 and x_2 are chosen uniformly at random and independently from $\{0,1\}^{q-p}$, and where $\circ$ denotes the concatenation of strings.

Now, given a labeling $\ell : V \to \{0,1\}^q$, define its "benefit" to be

$$Benefit(\ell) \doteq \sum_{v\in V} Profit_v(\ell(v)) - \sum_{u,v\in E} Cost_{u,v}(\ell(u), \ell(v)).$$

The problem is to efficiently find an $\hat{\ell}$ such that $Benefit(\hat{\ell}) \geq$ avgbenefit $\doteq E[Benefit(\ell)]$; this expectation is taken assuming that the labels $\{\ell(v) : v \in V\}$ are assigned uniformly at random and independently, from $\{0,1\}^q$. Note that we can assume that the labels $\ell(v)$ are uniform and *pairwise* independent over $\{0,1\}^q$, to compute avgbenefit.

The strategy used in [20] is to set one bit of each of the labels $\hat{\ell}(v)$ per stage without using too many processors. This results in an NC algorithm using $O(n+m)$ processors and running in $O(q(t+\log^2 n))$ time.

Our goal could be to develop an $O(t+\log n+q)$ time algorithm to find $\hat{\ell}$ with $Benefit(\hat{\ell}) \geq E[Benefit(\ell)] - n^{-\Theta(1)}$ for those instances of the the problem where the estimator functions $Profit$ and $Cost$ are bounded in value by a polynomial in n. We need this extra "polynomiality" requirement on $Profit$ and $Cost$ since we will work with sample spaces that are ϵ-biased and we can handle only the case where $\epsilon \geq 1/\text{poly}(n)$ in NC (and in polynomial time). Such polynomial bounds on $Profit$ and $Cost$ indeed hold for all the applications considered before and here. For some of the applications, a time bound of $O(t+q+\log n)$ would require the use of too many processors; thus, the time bound that we aim for is $(\gamma^{-1}(t+\log n))$, where γ will be a positive parameter that is at most one. The processor complexity will of course be a decreasing function of γ, and by taking $\gamma = \delta_1$ for a sufficiently small but fixed positive δ_1, we can also achieve an $O(t+\log n)$ time bound where needed, while keeping the processor count reasonably small.

A direct approach following the earlier algorithms would be to start with a randomly chosen string

$$y_{1,1}y_{1,2}\cdots y_{1,q}y_{2,1}\cdots y_{2,q}\cdots y_{n,1}y_{n,2}\cdots y_{n,q}$$

from a $2q$-wise $2^{-\Theta(q)}\cdot n^{-\Theta(1)}$-biased sample space for nq-length bit strings; we may then set $\ell(i) \doteq y_{i,1}y_{i,2}\cdots y_{i,q}$. However, the methods of the previous section require us to compute probabilities by first converting them to the appropriate bias terms; the large number of Fourier coefficients would then increase the processor complexity by a 2^{2q} factor.

Instead, we adopt the following strategy, motivated by a technique of [4] (see also [5]). The idea is to set γq bits of each of the labels $\hat{\ell}(v)$ all at once in each stage, and repeat γ^{-1} times. As said before, γ denotes a positive parameter that is at most one, whose value we choose based on the application. We assume that γq is an integer, for convenience (otherwise take its floor). Let $y_{i,j}$ and $\ell(i)$ be as in the previous paragraph. For simplicity of our expressions, we let α and β denote γq and γ^{-1} respectively. Let M denote the maximum value of $Profit$ or $Cost$; by assumption, M is polynomially bounded in n. Fix $\epsilon = \gamma 2^{-2\alpha} n^{-2-c} M^{-1}$ for any desired constant $c > 0$, and fix a 2α-wise ϵ-biased sample space S_1 for $n\alpha$–length bit strings.

We will show how to set, in stages $s = 1, 2, \ldots, \beta$, the vector

$$r_s = (y_{1,(s-1)\alpha+1}, \ldots, y_{n,(s-1)\alpha+1}, y_{1,(s-1)\alpha+2}, \ldots, y_{n,(s-1)\alpha+2}, \ldots, y_{1,s\alpha}, \ldots, y_{n,s\alpha});$$

our final labeling ℓ will be given by $\ell(i) \doteq y_{i,1}y_{i,2}\cdots y_{i,q}$. We now show how to deterministically set the vectors $r_1 := w_1, r_2 := w_2, \ldots, r_\beta := w_\beta$ in that order in NC, suitably. For $s = 1, 2, \ldots, \beta$, let $R_s \doteq (r_s, \ldots, r_\beta)$, let U_s and U_S denote the uniform distributions on $\{0,1\}^{n\alpha(\beta+1-s)}$ and S_1 respectively, and let $BU_s \doteq U_S \times U_{s+1}$. We will pick $w_1, w_2, \ldots$, such that for $s = 0, 1, \ldots, \beta$,

$$E[Benefit(\ell)|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim U_{s+1}] \geq \text{avgbenefit} - s\gamma n^{-c}. \qquad (2)$$

(Recall that if D is a probability distribution, then $X \sim D$ means that the random variable X is picked according to D.) Thus, establishing this for $s = \beta$ will suffice to show that the final vector $\vec{w}$ is good.

Inequality (2) is established by induction on s, as in [4]; the basis $s = 0$ holds trivially. Assume that (2) is true for s; we now show how to pick $w_{s+1} \in S_1$ such that it holds for $s+1$. Suppose we pick R_{s+1} according to BU_{s+1}. Focus on some term $E[Profit_v(\ell(v))]$ in the benefit function. Note that we now assume that w_{s+1} is picked from the ϵ-biased space S_1, and that $w_{s'}$, for all $s' > s$, is unbiased and independent. Thus, it not hard to see, using the bias properties of S_1 and by the boundedness of the $Profit$ functions by M, that

$$E[Profit_v(\ell(v))|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim BU_{s+1}]$$

is at least

$$E[Profit_v(\ell(v))|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim U_{s+1}] - 2^\alpha \epsilon M.$$

Similarly, for any $\{u, v\} \in E$,

$$E[Cost_{u,v}(\ell(v), \ell(w))|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim BU_{s+1}]$$

is at most

$$E[Cost_{u,v}(\ell(v), \ell(w))|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim U_{s+1}] + 2^{2\alpha} \epsilon M.$$

Hence, the maximum possible additive "error" is at most $(2^\alpha \epsilon n + 2^{2\alpha} \epsilon m)M \leq \gamma n^{-c}$, by the definition of ϵ. More precisely,

$$E[Benefit(\ell)|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim BU_{s+1}]$$

is at least

$$E[Benefit(\ell)|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim U_{s+1}] - \gamma n^{-c}.$$

Thus, by the induction hypothesis, we see that $E[Benefit(\ell)|r_1 = w_1, \ldots, r_s = w_s, \text{ and } R_{s+1} \sim BU_{s+1}]$ is at least avgbenefit $-\,(s+1)\gamma n^{-c}$. Hence, there exists $w \in S_1$ such that

$$E[Benefit(\ell)|r_1 = w_1, \ldots, r_s = w_s, r_{s+1} = w, \text{ and } R_{s+2} \sim U_{s+2}] \geq \text{avgbenefit} - (s+1)\gamma n^{-c}.$$

Finding w reduces to computing

$$E[Benefit(\ell)|r_1 = w_1, \ldots, r_s = w_s, r_{s+1} = w', \text{ and } R_{s+2} \sim U_{s+2}] \tag{3}$$

for each $w' \in S$ and picking the w with the largest conditional expectation. By our assumption about the functions $Profit$ and $Cost$, such a conditional expectation can be computed in time t using $O(n+m)$ processors.

As before, though we need to sample the points w' from S_1, we start with much smaller sample spaces: S_1 is constructed by starting from an $(n2^{\gamma q})^{-\delta_2}$-biased sample space S_0 and taking $O(1)$ samples, as in § 3.1. In each stage we have to find a point in $w \in S_1$ which maximizes a sum of $(n+m)2^{2\gamma q}$ bias terms (this sum is got by expanding each of the $n+m$ terms using the Fourier bases as explained before). To find such a w, we use the method of conditional probabilities by sampling $O(1)$ times from the space S_0. Thus each stage can be implemented using $(n+m)2^{O(\gamma q)}|S_0|$ processors and the running time is $O(t + \log n + \gamma q)$ for each stage. Plugging in the size of S_0 from the standard construction of [3] and using the fact that there are γ^{-1} stages, we get

Theorem 6 *For any parameter $\gamma \in (0,1)$ and any fixed $\delta > 0$, there is an $((n+m)2^{\delta\gamma q}n^{\delta}, \gamma^{-1}(t+\log n) + q)$ algorithm to find a labeling $\hat{\ell}$ with $Benefit(\hat{\ell}) \geq E[Benefit(\ell)] - n^{-\Theta(1)}$ in the PROFIT/COST problem, for the case where the functions $Profit$ and $Cost$ are bounded in value by a polynomial of n. In particular (by taking γ as an arbitrarily small positive constant), for any fixed $\delta > 0$, there exists an $((n+m)2^{\delta q}n^{\delta}, t + \log n + q)$ algorithm to find such a labeling, if $Profit$ and $Cost$ are bounded by $n^{O(1)}$.*

As instantiations of this general framework we can derive the following as corollaries, via Theorem 6 and some results from [20] and [25]. Recall that any graph with maximum degree Δ can be colored using $\Delta+1$ colors. The first linear-processor NC algorithm for $(\Delta+1)$-coloring was presented in [20], with a running time of $O(\log^3 n \log\log n)$. It is shown in [20] that: (a) the General Profit/Cost problem can be solved using $O(n+m)$ processors in $O(q(t+\log n \log\log n) + \log^2 n \log\log n)$ time, and that: (b) $(\Delta+1)$-coloring can be solved by a $(P(n,m), \log n(\log n + T(n,m)))$ algorithm, if there is a $(P(n,m), T(n,m))$ NC algorithm for the Profit/Cost problem with $q = t = \log n$. These two points yield the bounds for Luby's algorithm. (For this problem, $Profit$ and $Cost$ actually lie in $[0,1]$.) Using these two points in conjunction with Theorem 6 by making γ an arbitrarily small constant, we get

Corollary 1 *$(\Delta+1)$-vertex coloring is solvable by an $((n+m)n^{\delta}, \log^2 n)$ algorithm, for any fixed $\delta > 0$.*

Recall that a connected graph is vertex-colorable with Δ colors iff it is neither an odd cycle nor a complete graph. The proof of Theorem 4 of [25] shows that if there is a $P(n,m)$–processor NC algorithm for $(\Delta+1)$-coloring that runs in $O(T(n,m))$ time, then we can find a Δ-coloring in NC using $P(n,m)$ processors in $O((\log n + T(n,m))\log^2 n/\log\Delta)$ time; see §5.1 in [25]. Thus, by Corollary 1, we get

Corollary 2 *For any fixed $\delta > 0$, there exists an $((n+m)n^{\delta}, \log^4 n/\log\Delta)$ algorithm for Δ-vertex coloring, for graphs that are Δ-vertex colorable.*

4.2 The maximum acyclic subgraph problem

Consider the Maximum Acyclic Subgraph problem: given a directed graph $G = (V, A)$, we wish to find a subset $\hat{A}$ of A such that the subgraph $G' = (V, \hat{A})$ is acyclic. The problem of finding a maximum-sized such $\hat{A}$ is known to be NP–hard. Berger [7] derives an NC algorithm which finds a subset $\hat{A}$ of size at least

$$target \doteq \frac{|A|}{2} + c_0\left(\sum_{i=1}^{n}\sqrt{deg(i)} + \sum_{i=1}^{n}|d_{\mathrm{out}}(i) - d_{\mathrm{in}}(i)|\right),$$

where $n = |V|$, $d_{\mathrm{out}}(v)$ and $d_{\mathrm{in}}(v)$ denote the out- and in-degrees of v in G, $deg(v) = d_{\mathrm{out}}(v) + d_{\mathrm{in}}(v)$, and $c_0 > 0$ is an absolute constant. This description in §4.4 of [7], is an expanded version of the work of [9]. It is first shown in [7] that we may assume without loss of generality that G has no pair of anti-parallel arcs, i.e., no pair of the form $\{(u,v),(v,u)\}$. The essential idea in their algorithm is as follows. Each vertex v first chooses a random label $r_v \in \{1,2,\ldots,\sigma\}$ in parallel, where σ is the smallest power of two that is at least as high as n. Each vertex v is then processed in parallel as follows. Let $N'_{\mathrm{out}}(v) = \{(v,w) \in A : r_w > r_v\}$, and let $N'_{\mathrm{in}}(v) = \{(w,v) \in A : r_w > r_v\}$. If $|N'_{\mathrm{out}}(v)| \geq |N'_{\mathrm{in}}(v)|$, then the contribution of v to $\hat{A}$ is $N'_{\mathrm{out}}(v)$; else the contribution is $N'_{\mathrm{in}}(v)$. The resulting set of arcs $\hat{A}$ clearly defines an acyclic graph; using the fourth moment method, it is shown in [7] that if the labels are assigned even 5–wise independently, then $E[|\hat{A}|]$ is at least as high as the above-defined *target*. Thus, the goal is to efficiently derandomize the above algorithm in NC, by computing appropriate values for the labels r_v.

We now present only the results of [7] that are directly relevant to us; more details can be obtained from §4.4 of [7]. Given an event B, let $\mathcal{I}(B)$ denote the *indicator variable* for B, i.e., $\mathcal{I}(B)$ equals 1 if B holds, and is 0 if B does not hold. For any $v \in V$, let $N(v) = \{w \in V : (v,w) \in A \text{ or } (w,v) \in A\}$, and let $\Delta = \max_{v \in V} |N(v)|$. Also let $V = \{1,2,\ldots,n\}$. The following theorem follows from §4.4 of [7]:

Theorem 7 ([7]) (a) *For any (random or deterministic) choice of the labels r_v in the above algorithm, the set of arcs $\hat{A}$ that is output, satisfies $|\hat{A}| \geq f(r_1, r_2, \ldots, r_n)$, where*

$$f(r_1,\ldots,r_n) \doteq |A|/2 - \Big(\sum_{(u,v)\in A} \mathcal{I}(r_u = r_v)\Big)/2 + \sum_{v \in V; S \subseteq N(v); |S| \leq 4} c_{v,S} \cdot \mathcal{I}(r_v < \min\{r_w : w \in S\});$$

the coefficients $\{c_{v,S}\}$ can be computed by an $(n\Delta^4, \log n)$ algorithm.

(b) *If the r_v's are picked uniformly at random and (5-wise) independently from $\{1,2,\ldots,\sigma\}$, then $E[f(r_1,\ldots,r_n)] \geq target$.*

Thus, the goal is to compute a suitable set of labels $\{r_v : v \in V\}$ in NC, so that $f(r_1,\ldots,r_n) \geq$ *target* holds. Let $q = \log \sigma = \lceil \log n \rceil$, and let us denote each r_v by a bit-string $y_{v,1} y_{v,2} \cdots y_{v,q}$, with each $y_{v,j} \in \{0,1\}$. To derandomize the above algorithm, the work of [7] makes a suitable choice for the vector $z_i = (y_{1,i} y_{2,i} \cdots y_{n,i})$ in stages $i = 1,2,\ldots,q$, via the method of conditional probabilities; it is also shown in [7] that any term such as $E[\mathcal{I}(r_u = r_v) | z_1, z_2, \ldots, z_{i-1}]$ or $E[\mathcal{I}(r_v < \min\{r_w : w \in S\}) | z_1, z_2, \ldots, z_{i-1}]$ can be computed in constant time using one processor. (Note that the problem on hand is an instance of the general framework considered in §3.3.) This results in an $(n\Delta^4, \log^3 n)$ algorithm for this problem [7]. We could replace their invocation of the general framework by ours and this would result in an algorithm of complexity $(n^{1+\delta}\Delta^4, \log^2 n)$ for any fixed $\delta > 0$. However, we now use a strategy similar to the one for the general profit/cost problem, leading to an NC^1 algorithm.

As in our approach for the general profit/cost problem, we proceed in $\beta = \gamma^{-1}$ stages, where γ here is a suitably small but fixed positive value. In each stage, we set γq bits of the labels of each of the vertices all at once. In stage s, we will suitably set the vector

$$g_s = (y_{1,(s-1)\gamma q+1}, \ldots, y_{n,(s-1)\gamma q+1}, y_{1,(s-1)\gamma q+2}, \ldots, y_{n,(s-1)\gamma q+2}, \ldots, y_{1,s\gamma q}, \ldots, y_{n,s\gamma q}).$$

We now sketch how to deterministically set the vectors $g_1 := w_1, g_2 := w_2, \ldots, g_\beta := w_\beta$ in that order in NC, suitably. For $s = 1, 2, \ldots, \beta$, let $G_s \doteq (g_s, \ldots, g_\beta)$, and let U_s denote the uniform distribution on $\{0,1\}^{n\gamma q(\beta+1-s)}$. We will pick $w_0, w_1, \ldots$ such that for $s = 0, 1, \ldots, \beta$,

$$E[f(r_1, \ldots, r_n) | g_1 = w_1, \ldots, g_s = w_s, \text{ and } G_{s+1} \sim U_{s+1}] \geq \mathit{target} - s\gamma n^{-c},$$

for some suitably large constant $c > 0$. Since the algorithm is very similar to our algorithm for the general Profit/Cost problem we merely sketch the details.

In each stage we need to sample from a sample space which is $(5\gamma q)$–wise $n^{-O(1)}$ biased sample space and we wish to maximize the sum of $O(n\Delta^4)$ functions each depending on at most $5\gamma q$ bits. First we will reduce this to maximizing $O(n\Delta^4 \times 2^{5\gamma q})$ bias terms. In each stage we start with a sample space which is $(5\gamma q)$–wise $n^{-\delta_1}$ biased and take the XOR of $O(1)$ samples. Each stage uses $n\Delta^4 n^{O(\gamma+\delta_1)}$ processors and runs in time $O(\log n + \log \Delta + q) = O(\log n)$. There are a constant number β of such stages, and thus we get

Theorem 8 *Given a directed graph $G = (V, A)$ and any fixed $\delta > 0$, there is an $(n^{1+\delta}\Delta^4, \log n)$ algorithm to find an acyclic subgraph of G with at least $|A|/2 + c_0(\sum_{i=1}^n \sqrt{deg(i)} + \sum_{i=1}^n |d_{\text{out}}(i) - d_{\text{in}}(i)|)$ arcs.*

5 A potential approach to some open problems

Consider the following abstract problem. Let $X_1, \ldots, X_n$ be independent random bits and define an *abstract estimator* to be a function of the form

$$X = \sum_{i=1}^{\log^c n} a_i \sum_{A \subseteq [n], |A| = i} \text{bias}(\oplus_{j \in A} X_j),$$

where the a_i are constants bounded in magnitude by $2^{\text{polylog}(n)}$. We wish to find a sample point with $X \leq E[X]$. An obvious instance of this framework is the set discrepancy problem where we wish to find an assignment χ with $disc(\chi) = O(\sqrt{s \log n})$ using the $O(\log n)$-th moment method, i.e., using variables which are $O(\log n)$ wise independent [8, 21]. For this application, we have a sum of n functions of this type, with $c = 1$. All the currently known methods fail to solve this problem in NC since they need one processor per term and there are $n^{\Theta(\log^c n)}$ terms. In most cases, however, it suffices to find a sample point with $X \leq E[X] + n^{-\Theta(1)}$. Therefore we can do with a $\log^c n$-wise $n^{-\Theta(\text{polylog}(n))}$-biased space. However, it can be shown that the size of such a sample space must be superpolynomial [3].

We therefore suggest the following approach. Suppose there exists a $\log^c n$-wise ϵ-biased space S_0 such that

(a) $\epsilon =$ constant, $|S_0| = n^{O(1)}$, and

(b) For any $\ell \leq \log^c n$ and any $S_1, S_2 \subseteq [n]$ with $|S_1| = |S_2| = \ell$, $\text{bias}(\oplus_{i \in S_1} X_i) = \text{bias}(\oplus_{i \in S_2} X_i)$ $(\leq \epsilon)$, for a random $\vec{X} \in S_0$.

If such a sample space exists and is efficiently constructible, then we claim that this problem is in NC. As before we take $\log^{O(1)} n$ samples $Y_1, Y_2, \ldots$ from S_0 using our method. Suppose we have picked $Y_1, \ldots, Y_i$ with $\oplus_{j=1}^{i} Y_j = b$. Then, the conditional benefit is easily computed since, for each $\ell \leq \log^c n$, we just need to know the *number* of sets $S \subseteq [n]$ with $|S| = \ell$ and $\oplus_{j \in S} b_j = 1$, which can be combinatorially evaluated.

Theorem 9 *The* abstract estimator *problem is in NC if there is an efficiently constructible S_0 satisfying (a) and (b). The special case $c = 1$ solves set discrepancy in NC.*

However, we do not even know if such a sample space S_0 exists! A $\log^c n$-wise independent space satisfies condition (b), but is too big. Property (b) seems too stringent: we leave as open the question whether exact equality is necessary in condition (b).

6 Approximating general distributions

We have so far studied small-bias sample spaces, i.e., small spaces that approximate the distribution of independent and uniform binary random variables. As said before, such efficiently constructible spaces have become a key tool in derandomization. With this motivation, the work of [11] considers the general problem of approximating the distribution of independent multivalued random variables:

Definition 5 *Let $X_1, \ldots, X_n \in \{0, \ldots, m-1\}$ be* independent *random variables with* arbitrary *individual distributions and joint distribution $\mathcal{D}$. A sample space S is a (k, ϵ)-approximation of $\mathcal{D}$ if, for $\vec{Y} = (Y_1, \ldots, Y_n)$ sampled uniformly from S, and any set I of at most k indices we have, for all $a_1, \ldots, a_{|I|}$ in $[0, m-1]$, $|Pr_{\vec{Y} \in S}(\bigwedge_{i \in I}(Y_i = a_i)) - \prod_{i \in I} Pr_{\mathcal{D}}(X_i = a_i)| \leq \epsilon$.*

(The case $m = 2$ with $\mathcal{D}$ being the uniform distribution is handled by small-bias spaces [22, 3, 2].) We are interested in such approximating sample spaces that are "small" and efficiently constructible. Indeed, as pointed out in [11], a simple probabilistic argument shows that *there exists* a (k, ϵ)-approximation of cardinality as small as $O(n \log(n/\epsilon)/\epsilon^2)$ for any $\mathcal{D}$ and any $k \leq n$. The crucial point is that as usual, we require that such a space also be *efficiently constructible*, for the purpose of derandomization. We now improve on the cardinality of the previous-best constructions of (k, ϵ)-approximations.

6.1 A geometric discrepancy problem

The efficient construction of (k, ϵ)-approximations can be reduced to the following *discrepancy* problem, as shown in [11]. An *axis-parallel rectangle* $R \subseteq [0, 1)^n$ is a cross-product of the form $[u_1, v_1) \times [u_2, v_2) \times \cdots \times [u_n, v_n)$, where $0 \leq u_i < v_i \leq 1$; the *volume* $vol(R)$ of R is defined naturally to be $\prod_i (v_i - u_i)$. Also, R is said to be *nontrivial in dimension* i iff either $u_i > 0$ or $v_i < 1$ holds. For any positive integer $k \leq n$, let $\mathcal{R}_n^k$ denote the set of axis-parallel rectangles $R \subseteq [0, 1)^n$ that are nontrivial in at most k dimensions.

The following discrepancy problem for rectangles is natural. Given a finite set $S \subseteq [0, 1)^n$, let $\vec{Y}$ be chosen uniformly at random from S. For any axis-parallel rectangle R, define the discrepancy of R w.r.t. S, $\Delta_S(R)$, to be $\mid Pr(\vec{Y} \in R) - vol(R) \mid$. Note that $vol(R)$ is the probability that a point chosen randomly from $[0,1)^n$ falls in R. Thus, $\Delta_S(R)$ is the discrepancy between the two measures of sampling uniformly from $[0,1)^n$ and sampling uniformly from S, w.r.t. R. We define

$$\Delta_S(\mathcal{R}_n^k) = \max_{R \in \mathcal{R}_n^k} \Delta_S(R).$$

The study of such discrepancy problems from a different viewpoint, has also been conducted in discrepancy theory (see, e.g., Beck & Chen [6]). Such problems are also relevant to certain numerical integration problems. To compute the integral of a function over a certain body, one common method in numerical integration is to approximate this integral by the arithmetic mean of this function evaluated at a finite set of points chosen judiciously within this body, times the volume of the body. Small-discrepancy sets are obvious candidates to be such a good set of points, and, indeed, there are bounds on the error in the above process, that show that a small discrepancy implies a small error.

The following lemma of [11] shows how the above geometric discrepancy problem is relevant for (k, ϵ)-approximations:

Lemma 2 ([11]) *Suppose, for integers k, n and for some parameter $\epsilon > 0$, a finite set $S \subset [0,1)^n$ satisfies $\Delta_S(\mathcal{R}_n^k) \leq \epsilon$. For any positive integer m, let $X_1, \ldots, X_n \in \{0, \ldots, m-1\}$ be* independent *random variables with* arbitrary *individual distributions and joint distribution $\mathcal{D}$. Then, there exists a (k, ϵ)-approximation S' for D, such that: (a) $|S'| = |S|$, and (b) given a uniformly random sample from S, a uniformly random sample from S' can be generated deterministically in time polynomial in n and m.*

Given Lemma 2, we shall focus on the problem of efficiently constructing a finite set $S \subseteq [0,1)^n$ such that

$$\text{for any } R \in \mathcal{R}_n^k,\ \Delta_S(R) \leq \epsilon \text{ holds.} \tag{4}$$

Of course, we also require S to be efficiently constructible, and $|S|$ to be "small". The major goal here is to construct S such that

$$|S| = \text{poly}(k, \log n, 1/\epsilon); \tag{5}$$

even a construction with cardinality $\text{poly}(n, 1/\epsilon)$ will be very interesting. However, these seem to be difficult problems for now. The work of [11] describes efficient constructions of three sample spaces $S_1, S_2, S_3 \subset [0,1)^n$ to satisfy (4) with

$$|S_1| = \text{poly}(\log n, 2^k, 1/\epsilon),\ |S_2| = (n/\epsilon)^{O(\log(1/\epsilon))},\ \text{and } |S_3| = (n/\epsilon)^{O(\log n)}. \tag{6}$$

(The sample spaces S_2 and S_3 actually guarantee (4) for all $k \leq n$.) In contrast, our main result, given by Theorem 10, is an explicit construction of a sample space of size

$$\text{poly}(\log n, 1/\epsilon, (\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$$

that guarantees (4); this is an improvement over the previous constructions.

6.2 Useful preliminaries

We start with the simple

Proposition 1 *Suppose: (a) $a_1 \le x_1 \le a_1 + b_1$ and $a_2 \le x_2 \le a_2 + b_2$, or (b) $a_1 - b_1 \le x_1 \le a_1$ and $a_2 - b_2 \le x_2 \le a_2$ holds, for $b_1, b_2 \ge 0$. Then, $|x_1 - x_2| \le |a_1 - a_2| + |b_1 - b_2| + b_2$.*

Proof. It is evident that whether case (a) or case (b) holds, we have $|x_1 - x_2| \le |a_1 - a_2| + \max\{b_1, b_2\}$. We now use the fact that $\max\{b_1, b_2\} \le |b_1 - b_2| + b_2$ to complete the proof. $\square$

We next present a useful lemma, whose proof closely follows that of Theorem 2 of [11] (see also Theorem 2.6 in [27]):

Lemma 3 *Let $X_1, X_2, \ldots, X_t$ be* independent *binary random variables with $Pr(X_i = 1) = p_i$; let $Z_1, Z_2, \ldots, Z_t$ be arbitrary binary random variables. Then, for any positive integer $k \le t$, $|\ Pr(\bigwedge_{i\in[t]}(Z_i = 0)) - Pr(\bigwedge_{i\in[t]}(X_i = 0))\ |$ is at most*

$$2^{-k} + e \cdot e^{-k/(2e)} + \sum_{\ell=1}^{k} \sum_{A\subseteq[t]:|A|=\ell} |\ Pr(\bigwedge_{i\in A}(Z_i = 1)) - Pr(\bigwedge_{i\in A}(X_i = 1))\ | .$$

Proof. For any positive integer $\ell \le t$, let

$$B_\ell = \sum_{A\subseteq[t]:|A|=\ell} Pr(\bigwedge_{i\in A}(Z_i = 1)), \text{ and } C_\ell = \sum_{A\subseteq[t]:|A|=\ell} Pr(\bigwedge_{i\in A}(X_i = 1)).$$

Case I: $\sum_{i\in[t]} p_i \le k/(2e)$. One consequence of the proof of Theorem 2 of [11] is that in this case,

$$C_k \le 2^{-k}. \tag{7}$$

Let $a_1 = 1 - B_1 + B_2 - \cdots + (-1)^{k-1}B_{k-1}$, $b_1 = B_k$, $a_2 = 1 - C_1 + C_2 - \cdots + (-1)^{k-1}C_{k-1}$, and $b_2 = C_k$. By standard inclusion-exclusion, we know that

$$\begin{aligned} a_1 &\le Pr(\bigwedge_{i\in[t]}(Z_i = 0)) \le a_1 + b_1 \text{ and } a_2 \le Pr(\bigwedge_{i\in[t]}(Z_i = 0)) \le a_2 + b_2, \text{ if } k \text{ is even;} \\ a_1 - b_1 &\le Pr(\bigwedge_{i\in[t]}(Z_i = 0)) \le a_1 \text{ and } a_2 - b_2 \le Pr(\bigwedge_{i\in[t]}(Z_i = 0)) \le a_2, \text{ if } k \text{ is odd.} \end{aligned}$$

Thus, we can use (7) to see, via Proposition 1, that

$$|\ Pr(\bigwedge_{i\in[t]}(Z_i = 0)) - Pr(\bigwedge_{i\in[t]}(X_i = 0))\ | \le |a_1 - a_2| + |b_1 - b_2| + 2^{-k}.$$

Finally, we use the fact that $|a_1 - a_2| \le \sum_{\ell=1}^{k-1} |B_\ell - C_\ell|$ to conclude the proof for Case I.

Case II: $\sum_{i\in[t]} p_i > k/(2e)$. Here, let $t' < t$ be the first index such that $k/(2e) - 1 < \sum_{i\in[t']} p_i \le k/(2e)$. In this case, since $\sum_{i\in[t']} p_i \le k/(2e)$, it follows as in Case I that

$$|\ Pr(\bigwedge_{i\in[t']}(Z_i = 0)) - Pr(\bigwedge_{i\in[t']}(X_i = 0))\ | \le 2^{-k} + \sum_{\ell=1}^{k} \sum_{A\subseteq[t']:|A|=\ell} |\ Pr(\bigwedge_{i\in A}(Z_i = 1)) - Pr(\bigwedge_{i\in A}(X_i = 1))\ | . \tag{8}$$

It also follows from the proof of Theorem 2 of [11] that since $\sum_{i\in[t']} p_i > k/(2e) - 1$, we have

$$Pr(\bigwedge_{i\in[t']}(X_i = 0)) < \left(1 - \frac{(k/2e) - 1}{t'}\right)^{t'} \leq e \cdot e^{-k/(2e)}.$$

This fact, combined with (8), and the observation that

$$0 \leq Pr(\bigwedge_{i\in[t]}(Z_i = 0)) \leq Pr(\bigwedge_{i\in[t']}(Z_i = 0)),\ 0 \leq Pr(\bigwedge_{i\in[t]}(X_i = 0)) \leq Pr(\bigwedge_{i\in[t']}(X_i = 0)),$$

shows that $\mid Pr(\bigwedge_{i\in[t]}(Z_i = 0)) - Pr(\bigwedge_{i\in[t]}(X_i = 0)) \mid$ is at most

$$2^{-k} + e \cdot e^{-k/(2e)} + \sum_{\ell=1}^{k} \sum_{A\subseteq[t']:|A|=\ell} \mid Pr(\bigwedge_{i\in A}(Z_i = 1)) - Pr(\bigwedge_{i\in A}(X_i = 1)) \mid .$$

Since $t' < t$, this concludes the proof for Case II also. $\square$

We also require the simple

Proposition 2 *For $x > 0$ and positive integers r, t, $r \leq t$, we have $\sum_{i=0}^{r} \binom{t}{i} x^i \leq (te^x/r)^r$.*

Proof. For any i, $0 \leq i \leq r$, we have $\binom{t}{i} x^i \leq (t/r)^r (r/t)^i \binom{t}{i} x^i$. Thus, $\sum_{i=0}^{r} \binom{t}{i} x^i \leq \sum_{i=0}^{t} (t/r)^r \binom{t}{i} (xr/t)^i = (t/r)^r (1 + xr/t)^t \leq (te^x/r)^r$. $\square$

6.3 The construction

We present a construction whose size either matches or is smaller than those of [11]. We start with a useful lemma which, in conjunction with Lemma 3, will help validate our construction.

Lemma 4 *Suppose that a finite set $S \subset [0,1)^n$ satisfies $\Delta_S(\mathcal{R}_n^{k'}) \leq \epsilon'$, for some k' and ϵ'. Let $\vec{Y} = (Y_1, Y_2, \ldots, Y_n)$ be sampled uniformly at random from S. Then, for any $J \subseteq [n]$ with $|J| = s \leq k'$ and any set $\{u_i, v_i :\ i \in J,\ 0 \leq u_i < v_i \leq 1\}$,*

$$\mid Pr(\bigwedge_{i\in J}(Y_i \notin [u_i, v_i)) - \prod_{i\in J}(1 - (v_i - u_i)) \mid\leq 2^s \epsilon'.$$

Proof. We assume without loss of generality that $J = [s]$. For each $i \in J$, let $I_{i,0} = [0, u_i)$, and $I_{i,1} = [v_i, 1)$. Since the event "$Y_i \notin [u_i, v_i)$" is the *disjoint* union of the events "$Y_i \in I_{i,0}$" and "$Y_i \in I_{i,1}$", we can verify that

$$Pr(\bigwedge_{i\in J}(Y_i \notin [u_i, v_i))) = \sum_{\sigma\in\{0,1\}^s} Pr(\bigwedge_{i\in J}(Y_i \in I_{i,\sigma_i})). \tag{9}$$

Let $\ell_{i,0} = u_i$ and $\ell_{i,1} = 1 - v_i$. It is clear that

$$\prod_{i\in J}(1 - (v_i - u_i)) = \sum_{\sigma\in\{0,1\}^s} \prod_{i\in J} \ell_{i,\sigma_i}. \tag{10}$$

Now, for any $\sigma \in \{0,1\}^s$, we have $| \; Pr(\bigwedge_{i\in J}(Y_i \in I_{i,\sigma_i})) - \prod_{i\in J} \ell_{i,\sigma_i} \; | \leq \epsilon'$, since $\Delta_S(\mathcal{R}_n^{k'}) \leq \epsilon'$. This fact, combined with equations (9) and (10) and the triangle inequality, concludes the proof. $\square$

We now present our main result:

Theorem 10 *There is an explicit* $S \subseteq [0,1)^n$ *of size* $\mathrm{poly}(\log n, 1/\epsilon, (\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$*, that satisfies (4). Thus, the joint distribution of any* n independent *discrete random variables, has an explicit* (k,ϵ)*-approximating sample space of cardinality* $\mathrm{poly}(\log n, 1/\epsilon, (\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$.

Proof. Define

$$k' = \min\{p \in Z^+ : \; 2^{-p} + e \cdot e^{-p/(2e)} \leq \epsilon/2\}, \text{ and } \epsilon' = \frac{\epsilon}{2}(\frac{k'}{ke^2})^{k'}. \tag{11}$$

Note that $k' = O(\log(1/\epsilon))$, and that $1/\epsilon' = \mathrm{poly}(1/\epsilon, (\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$. Thus, the sample space S_1 (see (6)) shows an explicit finite set $T \subset [0,1)^n$ such that

$$\Delta_T(\mathcal{R}_n^{k'}) \leq \epsilon', \text{ and } |T| = \mathrm{poly}(\log n, 1/\epsilon, (\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)}). \tag{12}$$

We now claim that $\Delta_T(\mathcal{R}_n^k) \leq \epsilon$, which will conclude the proof. To show this, let $R = [u_1,v_1) \times [u_2,v_2) \times \cdots \times [u_n,v_n)$ be an arbitrary element of $\mathcal{R}_n^k$. Without loss of generality, suppose R is nontrivial in dimensions $1,2,\ldots,k$. Let $\vec{Y} = (Y_1, Y_2, \ldots, Y_n)$ be sampled uniformly at random from T. For each $i \in [k]$, let: (i) $Z_i \in \{0,1\}$ be a random variable that is 1 iff $Y_i \notin [u_i,v_i)$, and (ii) $X_i \in \{0,1\}$ be a random variable that is 1 iff a point drawn uniformly at random from $[0,1)^n$ *does not* lie in $[u_i,v_i)$. Thus, our goal is to show that

$$| \; Pr(\bigwedge_{i\in[k]}(Z_i = 0)) - Pr(\bigwedge_{i\in[k]}(X_i = 0)) \; | \leq \epsilon. \tag{13}$$

By Lemmas 3 and 4, we see that

$$\begin{aligned} | \; Pr(\bigwedge_{i\in[k]}(Z_i = 0)) - Pr(\bigwedge_{i\in[k]}(X_i = 0)) \; | &\leq 2^{-k'} + e \cdot e^{-k'/(2e)} + \sum_{\ell=1}^{k'} \sum_{A\subseteq[k]:|A|=\ell} 2^\ell \epsilon' \\ &= 2^{-k'} + e \cdot e^{-k'/(2e)} + \epsilon' \sum_{\ell=1}^{k'} \binom{k}{\ell} 2^\ell \\ &\leq 2^{-k'} + e \cdot e^{-k'/(2e)} + (ke^2/k')^{k'}\epsilon' \text{ (Proposition 2)}, \end{aligned}$$

which is at most ϵ by (11). Thus, since (13) is established, the proof is complete. $\square$

Thus, our construction either matches or is better in size than all three of [11]. For instance if $k = \log^b n$ and $\epsilon = n^{-\Theta(1)}$ with $b \geq 2$, our construction has size $n^{O(\log\log n)}$, while those of [11] have size $n^{O(\log n)}$.

7 Discussion

We have devised a new probabilistic technique: the construction of a sample space whose Fourier coefficients are close to those of a independent random source on n random variables. Though the size of the sample space so constructed is larger than those of previous constructions, its definition is particularly amenable to the application of the method of conditional probabilities. Using this we derived efficient parallel algorithms for several combinatorial optimization problems. For a large class of problems our method yields algorithms which are faster than previously known algorithms or match the running time of the best known algorithms with a significant reduction in the processor complexity.

The above result is primarily concerned with approximating the distribution of a set of n independent binary and unbiased random variables, using a small sample space. An obvious generalization is to study such approximations for n independent multi-valued random variables that have arbitrary marginal distributions. Such constructions again have pseudorandom generators and derandomization as their main motivations. For this problem, we have presented a construction that improves on the previous-best results, due to [11].

Acknowledgments. We thank Juris Hartmanis, Ronitt Rubinfeld and David Shmoys for their guidance and support, and Noga Alon, Bernard Chazelle and Moni Naor for valuable discussions.

References

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

[2] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth, Construction of asymptotically good, low-rate error-correcting codes through pseudo-random graphs, *IEEE Trans. Info. Theory*, 38:509–516, 1992.

[3] N. Alon, O. Goldreich, J. Håstad, and R. Peralta, Simple constructions of almost k–wise independent random variables, *Random Structures and Algorithms*, 3(3):289–303, 1992.

[4] N. Alon and M. Naor, Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions, *Algorithmica*, 16:434–449, 1996.

[5] N. Alon and A. Srinivasan, Improved parallel approximation of a class of integer programming problems, To appear in *Algorithmica*. A preliminary version of this work appears in the *Proc. International Colloquium on Automata, Languages and Programming*, pages 562–573, 1996.

[6] J. Beck and W. Chen, Irregularities of distribution, Cambridge University Press, 1987.

[7] B. Berger, *Using randomness to design efficient deterministic algorithms*, PhD thesis, Department of Electrical Engineering and Computer Science, MIT, May 1990.

[8] B. Berger and J. Rompel, Simulating $(\log^c n)$-wise independence in NC, *Journal of the ACM*, 38:1026–1046, 1991.

[9] B. Berger and P. W. Shor, Approximation algorithms for the maximum acyclic subgraph problem, in *Proc. ACM/SIAM Symposium on Discrete Algorithms*, pages 236–243, 1990.

[10] B. Chazelle and J. Matoušek, On linear-time deterministic algorithms for optimization problems in fixed dimension, in *Proc. ACM/SIAM Symposium on Discrete Algorithms*, pages 281–290, 1993.

[11] G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Veličković, Approximations of general independent distributions, in *Proc. ACM Symposium on Theory of Computing*, pages 10–16, 1992.

[12] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, Monte Carlo simulations: Hidden errors from "good" random number generators, *Physical Review Letters*, 69(23):3382–3384, 1992.

[13] Y. Han, A fast derandomization scheme and its applications, *SIAM J. Comput.*, 25:52–82, 1996.

[14] T.-s. Hsu, *Graph augmentation and related problems: theory and practice*, PhD thesis, Department of Computer Sciences, University of Texas at Austin, October 1993.

[15] T.-s. Hsu, V. Ramachandran, and N. Dean, Parallel implementation of algorithms for finding connected components, in *DIMACS International Algorithm Implementation Challenge*, pages 1–14, 1994.

[16] A. Joffe, On a set of almost deterministic k–independent random variables, *The Annals of Probability*, 2(1):161–162, 1974.

[17] H. J. Karloff and P. Raghavan, Randomized algorithms and pseudorandom numbers, *Journal of the ACM*, 40(3):454–476, 1993.

[18] R. M. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, *Journal of the ACM*, 32:762–773, 1985.

[19] M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.*, 15(4):1036–1053, 1986.

[20] M. Luby, Removing randomness in parallel computation without a processor penalty, *Journal of Computer and System Sciences*, 47(2):250–286, 1993.

[21] R. Motwani, J. Naor, and M. Naor, The probabilistic method yields deterministic parallel algorithms, *Journal of Computer and System Sciences*, 49:478–516, 1994.

[22] J. Naor and M. Naor, Small–bias probability spaces: efficient constructions and applications, *SIAM J. Comput.*, 22(4):838–856, 1993.

[23] N. Nisan, Extracting Randomness: How and Why, in *Proc. IEEE Conference on Computational Complexity (formerly "Structure in Complexity Theory")*, pages 44–58, 1996.

[24] N. Nisan, E. Szemerédi, and A. Wigderson, Undirected connectivity in $O(\log^{1.5} n)$ space, in *Proc. IEEE Symposium on Foundations of Computer Science*, pages 24–29, 1992.

[25] A. Panconesi and A. Srinivasan, The local nature of Δ–colorings and its algorithmic applications, *Combinatorica*, 15:255–280, 1995.

[26] P. Raghavan, Probabilistic construction of deterministic algorithms: approximating packing integer programs, *Journal of Computer and System Sciences*, 37:130–143, 1988.

[27] J. P. Schmidt, A. Siegel, and A. Srinivasan, Chernoff-Hoeffding bounds for applications with limited independence, *SIAM J. Discrete Math.*, 8:223–250, 1995.

[28] J. H. Spencer, Balancing vectors in the max norm, *Combinatorica*, 6:55–65, 1986.

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