Improved Algorithms via Approximations of Probability Distributions[1]

Suresh Chari[2] and Pankaj Rohatgi[3]
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

and

Aravind Srinivasan[4]
Bell Laboratories, Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974-0636

We present two techniques for constructing sample spaces that approximate probability distributions. The first is a simple method for constructing the small-bias probability spaces introduced by Naor and Naor. We show how to efficiently combine this construction with the *method of conditional probabilities* to yield improved parallel algorithms for 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 are of smaller size than the constructions of Even, Goldreich, Luby, Nisan and Veličković.

Key Words: derandomization, parallel algorithms, discrepancy, graph coloring, small sample spaces, explicit constructions

[1]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.

[2]The work described here was done while the author was at the Dept. of Computer Science, Cornell University, Ithaca, NY 14853, USA and supported in part by NSF grant CCR-9123730. E-mail: schari@watson.ibm.com

[3]The work described here was done while the author was at the Dept. of Computer Science, Cornell University, Ithaca, NY 14853, USA and supported in part by NSF grant CCR-9123730 and an IBM Graduate Fellowship. E-mail: rohatgi@watson.ibm.com

[4]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, supported in part by National University of Singapore Academic Research Fund Grant RP970607. E-mail: srin@research.bell-labs.com

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, use this approach [33, 8]. Two motivations for research in this area are the non-availability of "perfect" random sources, and the need for absolute (non-probabilistic) guarantees of correctness in critical applications. The theory of *pseudorandomness* offers a sound theoretical framework to analyze the quality of pseudorandom generators where one takes a small truly random seed and "stretches" it into a much longer stream of pseudorandom bits [14, 39]; under unproven (but plausible) complexity-theoretic assumptions, the theory shows how certain such generators are sufficient to run various classes of randomized algorithms. This sound notion of pseudorandomness should be distinguished from "pseudorandom" generators available in computers, which are based on some heuristics that will not work in all situations. The fact that computers do not use "real" random sources prevents randomized algorithms from having a sound practical footing. For example, it has been shown that if algorithms such as randomized Quicksort are implemented with prevalent "pseudorandom" generators, their expected running times can be high [24]. It has also been observed that Monte-Carlo simulations yield remarkably different results under different "random-number generators" [19], and that direct implementations of certain randomized parallel algorithms take longer time than expected [22, 21].

In this paper, we present two new techniques for derandomization. The first leads to improved parallel algorithms for many basic problems such as finding large cuts in graphs, set discrepancy, $(\Delta + 1)$-vertex coloring of graphs etc., while the second improves the constructions due to [18] of small sample spaces for use in derandomization. The second method yields smaller sample spaces for the efficient derandomization of randomized algorithms.

We denote the probability of an event A by $\Pr(A)$, and the expected value of a real-valued random variable X by $\mathbf{E}[X]$. We also let $[t]$ denote the set $\{1, 2, \ldots, t\}$. Three major known approaches to derandomization are the techniques of limited independence ([26, 1, 17, 27, 2]), the method of conditional probabilities ([35, 38]), and small-bias probability spaces ([32, 4, 3]).

DEFINITION 1.1. 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 $S \subseteq \{0,1\}^n$ whose size is 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 known [23, 17, 27, 2]. The basic principle in derandomization is that in several cases we can analyze a given randomized algorithm and show that its behavior is good enough if the random bits input to this algorithm are k-wise independent for a suitably large $k = k(n)$, rather than completely independent. Using explicit constructions of such spaces, one can then search over all points in a k-wise independent sample space and *deterministically* output a good sample point for the randomized algorithm. However, it has been shown that if k is large, in particular if k grows with n, then k-wise independent

sample spaces for the X_is must have superpolynomial size, which is too large for efficient exhaustive search [16, 2].

Another promising approach to derandomization has been the concept of approximately independent sample spaces, which are closely related to "small-bias sample spaces", as discussed in Section 2. Motivated by the fact that randomized algorithms are usually robust to small changes in the probabilities, Naor and Naor observed that it often suffices if the sample space is *almost k-wise independent* [32], captured by

DEFINITION 1.2. A sample space $S \subseteq \{0,1\}^n$ is called 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

$$\mid \Pr(\bigwedge_{i \in I}(X_i = b_i)) - 2^{-|I|} \mid \leq \epsilon.$$

S is called k-wise ϵ-approximate or "almost k-wise independent", if the above inequality holds for all I such that $|I| \leq k$.

The important work of [32] presents efficient constructions of sample spaces $S \subseteq \{0,1\}^n$ which are k-wise ϵ-approximate. Thus we have efficient constructions of almost k-wise independent spaces and these sample spaces are often much smaller than their k-wise independent counterparts. Such almost k-wise independent spaces have since become very useful in derandomization, as several randomized algorithms are robust to such small changes in the probabilities.

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 $f(\vec{X})$ and, by analysis, we can show that $\mathbf{E}[f(\vec{X})]$ is large, where the measure of largeness depends on the problem. By definition, there exists $X^* \in \{0,1\}^n$ such that $f(X^*) \geq \mathbf{E}[f(\vec{X})]$: the main goal is to efficiently and *deterministically* find such a point X^*. The method of conditional probabilities finds this point by setting the components of X^* one bit at a time deterministically. We do this such that at every stage, the expectation of $f(\vec{X})$ conditioned on fixing the values of the components up till now is non–decreasing. This ensures that when we have assigned all n components we have found an X^* such that $f(X^*) \geq \mathbf{E}[f(\vec{X})]$.

In [29], 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 efficient parallel algorithms for fundamental problems such as set discrepancy [12, 31].

The motivation for our first method is similar to that of [29]. We observe that for a large class of problems, direct use of approximately independent spaces leads to inefficient parallel algorithms, and present a way to combine them with the method of conditional probabilities to get significantly better algorithms. Using this we obtain improved parallel algorithms for many basic problems such as finding large cuts in graphs, set discrepancy, $(\Delta + 1)$-vertex coloring of graphs and others where Δ is the maximum degree of the graph. 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.

Throughout this paper, the model for parallelism is the exclusive read, exclusive write EREW PRAM model of parallel computation [25]. Recall that a problem is in NC if it can be solved in $\log^{O(1)}(n)$ time using at most $p(n)$ processors on this model, where n denotes the length of the input to the problem and $p(\cdot)$ is any fixed polynomial. 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 1.3. **(Set Balancing)**: Given a ground set V and subsets $A_1, \ldots, A_n$ of V where $|V| \leq n$ and $|A_i| \leq s$ for each i, find an assignment $\chi : V \to \{0, 1\}$ such that the *discrepancy*

$$\mathrm{disc}(\chi) \doteq \max_i |(\sum_{j \in A_i} \chi(j)) - |A_i|/2|$$

is "small".

Spencer [38] gave a polynomial–time algorithm to find an assignment χ with discrepancy $O(\sqrt{s \log n})$. For any fixed $\delta > 0$, an assignment χ with $\mathrm{disc}(\chi) = O(s^{1/2+\delta}\sqrt{\log n})$ can be constructed by $O(\log^3 n)$ time NC algorithms, via limited independence ideas [12, 31]. The method of approximately independent spaces was used to improve the time complexity to $O(\log n)$ [32], but the processor complexity of this algorithm is $\Omega(n^{3+2/\delta})$. We derive an $O(\log n)$ time NC algorithm with processor complexity $O(s^2 n^{1+\delta'+1/\delta})$ for any fixed $\delta' > 0$ which is a significant improvement in the processor complexity. Since set balancing is a primitive for several other problems, this yields improved algorithms for problems like vector–balancing [37], lattice approximation [35], and computing ϵ-nets and ϵ-approximations for range spaces with finite VC-dimension [15]. Our set-balancing approach is then extended to a more general framework in Section 3.3.

A recent breakthrough result has led to NC algorithms whose discrepancy bound matches the sequential $O(\sqrt{s \log n})$ bound [30]: the running time is $O(\log^2 n)$ and the processor count is somewhat high—$O(n^{23})$. The main point of the work of [30] is to show how for discrepancy and some of its generalizations, one can indeed match the corresponding sequential bounds in NC; this had been a major open question. Our focus in this paper is on constructing efficient NC algorithms that deliver the weaker discrepancy guarantee of $O(s^{1/2+\delta}\sqrt{\log n})$.

The problem of finding a heavy codeword, which generalizes the classical problem of finding a large cut in a graph, was introduced in [32]. 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 $m/2$ ones. Approximately independent sample spaces are used in [32] 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 $(n^2 m, \log(n+m)\log m)$ algorithm, and the best time-complexity algorithm is an $(n^2 m^{1+\delta}, \log(m+n))$ algorithm, for any fixed $\delta > 0$. Thus we are able to improve significantly the processor complexity even in the

algorithm with the same time complexity. 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 $m/2$ cut edges. This well-known problem is a special case of the heavy codeword problem and NC algorithms of complexity $((n+m), \log^2 n)$ have been designed (see, e.g., [29]). Small-bias spaces can be used to obtain an $O(\log n)$ time NC algorithm but with a processor complexity of $\Theta(m^3 \log^2 n)$. Similarly, direct use of pairwise independent sample spaces leads to an $O(\log n)$ time NC 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 and Naor [5] to derive improved algorithms for other basic combinatorial 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 [29]. An improved $((n+m)/\log\log n, \log^2 n \log\log n)$ algorithm that runs on the CREW PRAM, has been devised [20]. 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 [34]. We use similar ideas to derive a faster algorithm for approximating the maximum acyclic subgraph problem [11], 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 processor complexity.

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 problem of approximating *arbitrary* independent distributions was addressed in [18].

DEFINITION 1.4. 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 $a_1, \ldots, a_{|I|} \in \{0, 1, \ldots, m-1\}$, $|\Pr(\bigwedge_{i \in I}(Y_i = a_i)) - \prod_{i \in I} Pr_D(X_i = a_i)| \leq \epsilon$.

Definition 1.2 shows that a k-wise ϵ-approximate set S is a (k, ϵ)-approximation for the uniform distribution on $\{0,1\}^n$. General (k, ϵ)-approximations have obvious applications to reducing randomness and to derandomization. Three constructions of such small sample spaces are presented in [18], 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 polynomial in $\log n$, $1/\epsilon$, and $(\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)}$. As in [18], we reduce this problem to a geometric discrepancy problem and our construction follows from our improved solution for this latter problem. Improved solutions for this geometric discrepancy problem also yield better methods for certain classes of numerical integration problems.

The rest of this work is organized as follows. The basic idea behind our first method is presented in section 2, followed by some direct applications of it, in

section 3. Section 4 presents some extensions of the basic method. In section 5, we present our second derandomization tool. Section 6 concludes.

2. THE FIRST METHOD: BASIC IDEAS

In this section we describe how to combine the method of conditional probabilities with the method of approximately independent sample spaces.

NOTATION 2.1. For any positive integer n and for any $a, b \in \{0,1\}^n$, $<a,b>$ denotes the dot product, over Z_2, of vectors a and b. 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.

The notion of small-bias sample spaces was developed in an important paper of Naor and Naor [32]:

DEFINITION 2.1. 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 defined to be $\text{bias}(<\vec{X}, \alpha>)$, and is denoted $\text{bias}_S(\alpha)$. If $|\text{bias}_S(\alpha)| \leq \epsilon$ for all nonzero α, then S is said to be ϵ-biased. Similarly S is said to be k-wise ϵ-biased if $|\text{bias}_S(\alpha)| \leq \epsilon$ for all α with $\text{wt}(\alpha) \leq k$.

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 [32]. 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 $x \in \{0,1\}^n$. Similarly, for any $t \leq n$, any $\{i_1, i_2, \ldots, i_t\} \subseteq [n]$ and any sequence of bits $b_1, b_2, \ldots, b_t$, $\Pr(\bigwedge_{j \in [t]}(X_{i_j} = b_j))$ can be rewritten as the sum of 2^t bias terms. These ideas will be useful in Sections 3.2, 3.3, and 4; they can also be used to show

THEOREM 2.1. **([32])** *If a set $S \subseteq \{0,1\}^n$ is k-wise ϵ-biased for some k and ϵ, then S is also k-wise ϵ-approximate.*

The following NC constructions of (n-wise) ϵ-biased sample spaces S are known: (i) $|S| = \text{O}(n/\epsilon^3)$ [3], improving on the sample space size of [32], and (ii) three different sample spaces S with $|S| = \text{O}(n^2/\epsilon^2)$ [4]. All these NC constructions run in $\text{O}(\log n)$ time.

The basis of our construction of approximate sample spaces is the following simple fact:

FACT 2.1. Let $X_1, \ldots, X_t$ be *independent* binary random variables, with $\text{bias}(X_i) = \epsilon_i$. Then, the random variable $Y_t = X_1 \oplus X_2 \oplus \ldots \oplus X_t$ has bias $\epsilon_1 \cdot \epsilon_2 \cdots \epsilon_t$.

Proof. We supply an elementary proof for completeness. Note that

$$\begin{aligned}\text{bias}(Y_t) &= \Pr(Y_{t-1}=0)\Pr(X_t=0)+\Pr(Y_{t-1}=1)\Pr(X_t=1)\\ &\quad -\Pr(Y_{t-1}=0)\Pr(X_t=1)-\Pr(Y_{t-1}=1)\Pr(X_t=0)\\ &= (\Pr(Y_{t-1}=0)-\Pr(Y_{t-1}=1))\cdot(\Pr(X_t=0)-\Pr(X_t=1))\\ &= \text{bias}(Y_{t-1})\cdot\text{bias}(X_t).\end{aligned}$$

Hence by induction, $\text{bias}(Y_t) \le \epsilon_1 \cdot \epsilon_2 \cdots \epsilon_t$. ■

That is, if we have many *independent* random bits each having a possibly "high" bias, we can XOR them to get a bit that is of much smaller bias. The new construction of small-bias sample spaces uses this fact, and is as follows:

LEMMA 2.1. *Let S' be an ϵ'-biased sample space and let $\vec{X_1}, \ldots, \vec{X_{t(\epsilon,\epsilon')}}$ be sampled independently and uniformly at random from S', where*
$t = t(\epsilon,\epsilon') = \lceil \log(\epsilon)/\log(\epsilon') \rceil$.
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. If the sample space S' is k-wise ϵ'-biased then the resulting space is k-wise ϵ-biased.

Proof. Fix any nonzero $\alpha \in \{0,1\}^n$. By definition,

$$\text{bias}(<\vec{X},\alpha>) = \text{bias}(<\textstyle\bigoplus_{i=1}^{t}\vec{X_i},\alpha>) = \text{bias}(<\vec{X_1},\alpha> \oplus \cdots \oplus <\vec{X_t},\alpha>).$$

Since the $\vec{X_i}$ are all independent and since $|\text{bias}(<\vec{X_i},\alpha>)| \le \epsilon'$ for each i, the first statement follows from Fact 2.1. The second statement follows if we restrict α to be a nonzero vector with $\text{wt}(\alpha) \le k$. ■

Thus our construction of a k-wise ϵ-biased sample space S is to start with a k-wise ϵ'-biased sample space S' of potentially much larger bias ϵ' and then to XOR $t(\epsilon,\epsilon')$ independent samples $\vec{X_1}, \vec{X_2}, \ldots,$ from S'.

A space constructed like this will, in general, be larger than those obtained by direct constructions of ϵ-approximate spaces: its cardinality is clearly $|S'|^{t(\epsilon,\epsilon')}$. However, when we actually use this method in derandomizing probabilistic algorithms, we will take the different samples $\vec{X_i}$ from the space S' *one at a time*. The choice of which particular $\vec{X_i}$ to choose in the current stage is done using the method of conditional probabilities: this essentially reduces to a simple exhaustive search over S' for a "good" point $\vec{X_i}$. Why does this lead to faster algorithms? Let S'' denote a generic k-wise ϵ-biased space constructed using previous methods; most previous-best algorithms for the problems considered, involve exhaustive search over S''. Since $\epsilon' \gg \epsilon$, we will typically have $|S'| \ll |S''|$, thus yielding more efficient algorithms since our search-space for each $\vec{X_i}$ is smaller. Also, $t(\epsilon,\epsilon')$ will typically be "small" in our applications; e.g., we sometimes just require it to be a sufficiently large constant.

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

THEOREM 2.2. **([4])** *Let* bin*: $GF(2^m) \mapsto \{0,1\}^m$ be the standard binary representation of the field of 2^m elements which satisfies* $\text{bin}(0) = 0^m$ *and* $\text{bin}(u+v) =$

$\mathrm{bin}(u) \oplus \mathrm{bin}(v)$. *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 $((n-1)/2^m)$–biased. Thus there is an explicit ϵ–biased sample space of size $\mathrm{O}(n^2/\epsilon^2)$.*

Similarly, k-wise ϵ-biased spaces of size $\mathrm{O}(\min\{\frac{k\log n}{\epsilon^3}, \frac{(k\log n)^2}{\epsilon^2}\})$ can be constructed by $\mathrm{O}(\log n)$ time NC algorithms [32, 4, 3].

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. We use $\delta, \delta', \delta'', \delta_1, \delta_2$ etc. to 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 [32]: 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 [32], if x is picked from an ϵ-biased space S with $\epsilon = 1/(2m)$, then

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

and thus, there exists an $x \in S$ with $wt(Ax) \geq \lceil m\,(1/2 - \epsilon/2)\rceil = \lceil m/2\rceil$. This yields a simple deterministic parallel algorithm where each processor tries out one point in the biased sample space. But, any known construction of an $1/(2m)$-biased space has $\Omega(\min\{m^3 n, m^2 n^2\})$ points, so a direct use of small-bias spaces results in high processor complexity.

Our approach is to start with an $m^{-\delta}$-biased sample space S_0 and pick x, as in section 2, to be the bit-wise XOR of $\ell = \lceil(\log 2m)/(\log m^\delta)\rceil \sim 1/\delta = \mathrm{O}(1)$ many *independent* samples $Y_1, Y_2, \ldots, Y_\ell$ from S_0. At each stage j, we will choose a "good" value for $Y_j^* \in S_0$ so that the resulting x after ℓ stages will yield a heavy codeword Ax with at least $\lceil\frac{m}{2}\rceil$ weight.

We show that the method of conditional probabilities can be used to find such a "good" sequence $Y_1^*, \ldots, Y_\ell^*$ *efficiently* in NC. To do this, we define a benefit function at each stage and the Y_j^* which we choose will be the point which maximizes it. Assume that we have already fixed the values of $Y_1^*, \ldots, Y_{j-1}^*$. Define the *conditional estimator* at stage j to be

$$\begin{aligned} f_j(y) &\doteq \mathbf{E}[\mathrm{wt}(Ax) | Y_k = Y_k^*, 1 \leq k \leq j-1 \text{ and } Y_j = y] \\ &= \sum_{i=1}^{m} \frac{(1 - \mathrm{bias}(<a_i, x> | Y_k = Y_k^*, 1 \leq k \leq j-1 \text{ and } Y_j = y))}{2}, \end{aligned}$$

where $\mathrm{bias}(X|\mathcal{A}) \doteq \Pr((X=0)|\mathcal{A}) - \Pr((X=1)|\mathcal{A})$ for a binary random variable X and event $\mathcal{A}$. Intuitively, $f_j(y)$ estimates the benefit of choosing y at the jth stage. We want a $Y_j^* \in S_0$ which maximizes $f_j(y)$: for this, we need only show that the function $f_j(\cdot)$ can be computed efficiently for all points in S_0.

Fix $y \in S_0$, and let $b = (\bigoplus_{k=1}^{j-1} Y_k^*) \oplus y$. 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. Now,

$$\text{bias}(<a_i, x> | Y_k = Y_k^*, 1 \le k \le (j-1) \text{ and } Y_j = y) = \text{bias}(<a_i, (b \oplus z)>,$$

which equals $\text{bias}(<a_i, z>)$ if $<a_i, b> = 0$, and $-\text{bias}(<a_i, z>)$ otherwise. From Fact 2.1, $\text{bias}(<a_i, z>) = (\text{bias}_{S_0}(a_i))^{\ell - j}$. Thus, for each point of S_0 we can compute the benefit function efficiently. The entire algorithm is thus as follows:

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

The size of S_0 is $O(m^{3\delta} n)$ and hence we can compute $\text{bias}_{S_0}(a_i)$ 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 $f_j(Y_j^*)$ can be evaluated in $O(\log(m+n))$ time using $O(mn)$ processors. Alternatively, in the algorithm, we could choose S_0 of constant bias. By Lemma 2.1 if we choose $\ell = O(\log m)$, the resulting sample space will have bias $m^{-O(1)}$. This gives an $(n^2 m, \log(m+n) \log m)$ algorithm.

THEOREM 3.1. *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.*

Thus, a key reason that led to this improved processor complexity is that our search space S_0 has size much less than $\min\{m^3 n, m^2 n^2\}$.

Given a graph G, we can use a similar algorithm to find a cut with at least $m/2$ 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, the expected number of cut edges is at least $m/2$. Direct use of biased spaces yields an $(m^3 \log^2 n, \log n)$ algorithm. Actually, 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 3.2. *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 construct 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 stage 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

In this section we outline an application where we use the biases of the base distribution to compute probabilities of events by Fourier inversion. Recall the definition of the set discrepancy problem: Given subsets $A_1, \ldots, A_n$ of a set V where $|V| \leq n$ and $|A_i| \leq s$ for each i, the problem is to find a $\chi : V \mapsto \{0,1\}$ such that $disc(\chi) \doteq \max_i |\chi(A_i) - |A_i|/2|$ is small, where $\chi(A_i) = \sum_{j \in A_i} \chi(j)$.

Let $V = \{x_1, x_2, \ldots, x_n\}$, X_i denote $\chi(x_i)$, and k be the smallest *even* integer that is at least as big as $\log(2n)/(\delta \log s)$. It is shown in [12, 31] that if we choose $X_1, X_2, \ldots, X_n$ k-wise independently with $\Pr(X_i = 1) = \Pr(X_i = 0) = 1/2$ for each i, then

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

This has been used to derive $O(\log^3 n)$ time NC algorithms that yield an assignment χ with discrepancy bounded by $s^{0.5+\delta}\sqrt{\log n}$. It is shown in [32] that (1) holds even if the distribution of $(X_1, X_2, \ldots, X_n)$ is k–wise ϵ-biased for $\epsilon < 1/(2n^{1+1/\delta})$, leading to an $O(\log n)$ time NC algorithm via exhaustive search of a k–wise ϵ-biased sample space. However, using any known construction of small-bias sample spaces, the processor complexity of such an algorithm will be $\Omega(n^{3+\frac{2}{\delta}})$. (Any known explicit construction of the desired sample space has $\Omega(n^{2+\frac{2}{\delta}})$ points; since there are n sets A_i, we need n processors to check in $O(\log n)$ time if a given point of the sample space is "good".) We show how to significantly improve on this processor complexity while retaining the $O(\log n)$ running time.

Note that we can assume $s \geq \log n$ without loss of generality. Indeed, since $disc(\chi) \leq s/2$ for any χ, we can trivially achieve $disc(\chi) \leq \sqrt{s \log n}$ if $s < \log n$. Next, suppose we generate $(X_1, X_2, \ldots, X_n)$ from a k–wise ϵ-biased sample space, where $\epsilon = (4n^{1+1/\delta})^{-1}$, say. For each $i \in [n]$, we can use linearity of expectation to rewrite $\mathbf{E}[(\chi(A_i) - \frac{|A_i|}{2})^k]$ as a sum of terms of the form $c \cdot \Pr(X_{i_1} = \cdots = X_{i_t} = 1)$, with $t \leq k$ and c a scalar. Since $|A_i| \leq s$ and $k \leq \log(2n)/(\delta \log s) + 2$, the number of such terms for any given A_i is at most

$$(s+1)^k \leq O(s^2 2^{\log(s+1)\log n/(\delta \log s)}) = O(s^2 2^{(\log s + O(1/s))\log n/(\delta \log s)}) = O(s^2 n^{1/\delta}),$$

since $s \geq \log n$ by assumption. For each of these terms we know how to compute probabilities of events once we we know the Fourier coefficients as shown in Section 2. In this case we can express the term $c \cdot \Pr(X_{i_1} = \cdots = X_{i_t} = 1)$ as the sum of $2^t = 2^{o(\log n)}$ *bias* terms (we get a $o(\cdot)$ in the exponent since $k = o(\log n)$). Now, exactly as in the case of the algorithm for the heavy codeword problem we can start with a k-wise $n^{-\delta''}$-biased sample space S_0 and define a good χ using as the XOR of $O(1)$ samples from S_0. By Lemma 2.1, the resulting sample space will be k-wise ϵ–biased. 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 is $|S_0| = n^{O(\delta'')}$ and the number of bias terms we need to consider for each A_i is $O(s^2 n^{1/\delta} 2^k)$; so, the total number of bias terms is $O(s^2 n^{1+1/\delta} 2^k)$. Thus, the algorithm runs in time $O(\log n)$ and the number of processors we need for each stage is $O(s^2 n^{1+1/\delta+\delta'})$. Thus we can obtain the following.

THEOREM 3.3. *There is an* $(s^2 n^{1+1/\delta+\delta'}, \log n)$ *algorithm to find a* χ *for the set discrepancy problem with* $disc(\chi) \leq s^{0.5+\delta}\sqrt{\log n}$, *for any fixed* $\delta, \delta' > 0$.

Since $disc(\chi) \leq s/2$ is trivially achievable as seen above, we can assume $\delta < 1/2$ without loss of generality. Thus, our processor requirement is a good improvement on the $\Omega(n^{3+2/\delta})$ requirement of [32].

The set discrepancy abstraction has been used to solve other problems such as vector balancing [37], lattice approximation [35, 31] and for computing ϵ-nets and ϵ-approximations for range spaces with finite VC–dimension [15]. Theorem 3.3 implies improved NC algorithms for all these problems.

3.3. A general framework

Set discrepancy has been captured as part of a more general framework in [12], and it is easy to extend the above method to this framework. Let $X_1, \ldots, X_n$ be unbiased and independent random bits, and let $\vec{X} = (X_1, X_2, \ldots, X_n)$. For positive constants a and b, consider a function such as $g(\vec{X}) = \sum_{i=1}^{n^a} f_i(\vec{X})$ where each f_i depends on at most $b \log n$ of the random variables $X_1, \ldots, X_n$. In several algorithms, we can define a function as above to capture the benefit in choosing a particular sample point [12]. By analysis we can show that if the X_is are unbiased and independent random bits, then $\mathbf{E}[g(\vec{X})]$ is a suitably large value v; to derandomize, we wish to deterministically find a sample point X^* with $g(X^*) \geq v$.

An $(n^{a+b}, t \log^3 n)$ algorithm is presented for this problem in [12] where each of the functions f_i is computable in $\mathrm{O}(t)$ time with one processor. Sometimes, it actually suffices to find a point X^* with $g(X^*) \geq v - n^{-\Theta(1)}$. Furthermore, once again as in set discrepancy, the functions f_i are often bounded in magnitude by $n^{\mathrm{O}(1)}$. Thus, it is not difficult to show that if we choose the X_is from a $b \log n$-wise n^{-C}-biased space S for a suitably large constant C, then $\mathbf{E}[g(\vec{X}) \geq v - n^{-\Theta(1)}$. Once again, our idea is to first express $\mathbf{E}[X]$ as a sum of at most n^{a+b} bias terms as detailed in Section 2. Then, if we start with a sample space S_0 with bias $n^{-\delta/3}$ 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 S_0 which is of constant bias we can derive an $(n^{a+b} \log^2 n, t \log^2 n)$ algorithm. In contrast, directly using the small-bias sample space S gives an $(\Omega(n^{3a+2b}), t \log n)$ algorithm.

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 at most logarithmic in the input size in our applications. By combining our basic method with some ideas of [5] we get improved parallel algorithms for various problems. Some of these problems can also be solved using the above general framework. However, the solutions we give here are better than those obtained as instantiations of the general framework.

4.1. The Profit/Cost problem

The first application we describe is for the "General Pairs Benefit Problem" of [29] (this problem is called the "General PROFIT/COST Problem" in [28]). This

models problems such as $(\Delta + 1)$-vertex coloring of graphs. The essence of this framework is the problem stated below.

We are given a graph $G = (V, E)$, with $|V| = n$ and $|E| = m$ and 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 $\{u, v\} \in E$. For any labeling $\ell : V \to \{0,1\}^q$,

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

The parameter q will be $O(\log n)$ in our applications.

The problem is to efficiently find an $\hat{\ell}$ such that $Benefit(\hat{\ell}) \geq E^*$, where E^* is the expected benefit when the labels $\{\ell(v) : v \in V\}$ are assigned uniformly at random and independently, from $\{0,1\}^q$. Since the profit and cost functions depend only on the labeling of at most two vertices, the expected benefit will be the same if the labels are uniform and *pairwise* independent over $\{0,1\}^q$.

The strategy used in [29] 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 is to develop an $O(t + \log n + q)$ time algorithm to find an $\hat{\ell}$ with $Benefit(\hat{\ell}) \geq E^* - n^{-\Theta(1)}$ for those instances of the problem where the functions $Profit$ and $Cost$ are bounded in magnitude by a polynomial in n. In all the applications of this framework, the associated functions do satisfy this constraint. Also, let t be a parameter such that the following holds. Suppose, for an arbitrary $j \in [q]$, the first j bits of the labels of all the variables have been fixed at arbitrary given values, and that the remaining $n(q - j)$ bits of the labeling ℓ are chosen uniformly at random from $\{0,1\}^{n(q-j)}$. Then, for any given $v \in V$ and $\{u, v\} \in E$, the conditional expectations $\mathbf{E}[Profit_v(\ell(v))]$ and $\mathbf{E}[Cost_{u,v}(\ell(u), \ell(v))]$ can be computed in time t each with one processor.

A direct approach following the earlier algorithms would be to start with a randomly chosen bit 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, ϵ-biased sample space for nq-length bit strings, for a suitably chosen ϵ. We can 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 [5, 6]. 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. Here, γ denotes a positive parameter that is at most one, whose value we choose based on the application. In the following, we assume that γq is an integer.

Let $\alpha = \gamma q$ and $\beta = \lceil \gamma^{-1} \rceil$. Our approach is to choose at each stage j, $1 \leq j \leq \beta$, the bit string

$$r_j = (y_{1,(j-1)\alpha+1}, \ldots, y_{n,(j-1)\alpha+1}, y_{1,(j-1)\alpha+2}, \ldots, y_{n,(j-1)\alpha+2}, \ldots, y_{1,j\alpha}, \ldots, y_{n,j\alpha}).$$

The final labeling on the vertices will be ℓ defined by $\ell(i) \doteq y_{i,1}y_{i,2} \cdots y_{i,q}$.

At stage j of the algorithm, we have already chosen the first $(j-1)\alpha$ bits of the labeling of each vertex. The following lemma will show that in stage j, if we choose r_j from an sample space S_0 with small bias then the penalty incurred from not choosing r_j uniformly at random is small. Let M denote the maximum magnitude of the values taken by the functions $Profit$ or $Cost$; by assumption, $M \leq n^{\mathrm{O}(1)}$. Given a finite set A, let $Z \sim U(A)$ denote random variable Z being picked uniformly at random from A.

LEMMA 4.1. *Fix $\epsilon = \gamma 2^{-2\alpha} n^{-2-c} M^{-1}$, where c is is a positive constant. Let S_0 be any 2α-wise ϵ-biased sample space for $n\alpha$–length bit strings. There exist $r_1^*, r_2^*, \ldots, r_\beta^* \in S_0$ such that for each $j \in [\beta]$,*

$$\begin{aligned} & E[\ Benefit(\ell) \ \mid \ r_k = r_k^*, 1 \leq k \leq j \\ & \qquad\qquad \text{and } (r_{j+1}, r_{j+2}, \ldots, r_\beta) \sim U(\{0,1\}^{n\alpha(\beta-j)})\ ] \\ & \geq \ E^* - j\gamma n^{-c}. \end{aligned} \tag{2}$$

Proof. The proof is by induction on j. The base case $j = 0$ is immediate; we prove the lemma for j, $1 \leq j \leq \beta$, by assuming it for $j-1$. Let $r_1^*, \ldots, r_{j-1}^* \in S_0$ be the values given by the induction hypothesis for $j-1$.

Since S_0 is also 2α-wise ϵ–approximate, it can be seen that for any fixed vertex v,

$$\begin{aligned} & \mathbf{E}[\ Profit_v(\ell(v)) \ \mid \ r_k = r_k^*,\ 1 \leq k \leq (j-1), \\ & \qquad\qquad (r_j,\ r_{j+1},\ \ldots,\ r_\beta) \ \sim \ U(S_0 \times \{0,1\}^{n\alpha(\beta-j)})\ ] \\ & \geq \ \mathbf{E}[Profit_v(\ell(v)) \ \mid \ r_k = r_k^*,\ 1 \leq k \leq (j-1), \\ & \qquad\qquad (r_j,\ r_{j+1},\ \ldots,\ r_\beta) \ \sim \ U(\{0,1\}^{n\alpha(\beta-j+1)})\ ] \\ & \qquad -2^\alpha \epsilon M. \end{aligned}$$

This is because the function $Profit$ is bounded in value by M, the sample space S_0 is 2α-wise ϵ-approximate and there are only α ($\leq 2\alpha$) bits of the labeling of the vertex v that are being fixed by the string r_j. Similarly for any $\{u, v\} \in E$,

$$\begin{aligned} & \mathbf{E}[\ Cost_{u,v}(\ell(u), \ell(v)) \ \mid \ r_k = r_k^*, 1 \leq k \leq (j-1), \\ & \qquad\qquad (r_j,\ r_{j+1},\ \ldots,\ r_\beta) \ \sim \ U(S_0 \times \{0,1\}^{n\alpha(\beta-j)})\ ] \\ & \geq \ \mathbf{E}[\ Cost_{u,v}(\ell(u), \ell(v)) \ \mid \ r_k = r_k^*, 1 \leq k \leq (j-1), \\ & \qquad\qquad (r_j,\ r_{j+1},\ \ldots,\ r_\beta) \ \sim \ U(\{0,1\}^{n\alpha(\beta-j+1)})\ ] \\ & \qquad -2^{2\alpha} \epsilon M. \end{aligned}$$

Thus by linearity of expectation,

$$\begin{aligned} & \mathbf{E}[\ Benefit(\ell) \ \mid \ r_k = r_k^*, 1 \leq k \leq (j-1), \\ & \qquad\qquad (r_j\ , r_{j+1}\ ,\ \ldots,\ r_\beta) \ \sim \ U(S_0 \times \{0,1\}^{n\alpha(\beta-j)})\ ] \\ & \geq \ \mathbf{E}[\ Benefit(\ell) \ \mid \ r_k = r_k^*, 1 \leq k \leq (j-1), \\ & \qquad\qquad (r_j\ , r_{j+1}\ , \ldots\ , r_\beta) \ \sim \ U(\{0,1\}^{n\alpha(\beta-j+1)})\ ] \\ & \qquad - \ M\epsilon(2^\alpha n + 2^{2\alpha} m), \end{aligned}$$

which in turn is at least $E^* - j\gamma n^{-c}$ (via the induction hypothesis and the fact that $M\epsilon(2^\alpha n + 2^{2\alpha} m) \leq \gamma n^{-c}$). So there exists $r_j^* \in S_0$ such that (1) holds. ■

The above lemma gives us an algorithm to pick the labeling of the vertices γq bits at a time all at once. At stage j, to find the point r_j^*, we need to compute

$$f_j(w) \;=\; \mathbf{E}[\; Benefit(\ell) \quad | \quad r_k = r_k^*, 1 \leq k \leq (j-1),\; r_j = w, \\ (r_{j+1}\;, r_{j+2}\;, \ldots\;, r_\beta) \;\sim\; U(\{0,1\}^{n\alpha(\beta-j)})\;]$$

for each point $w \in S_0$ and pick r_j^* as the point w which maximizes the value of the function. Recalling the definition of t, the function $f_j(\cdot)$ can be computed in time $\mathrm{O}(t + \log n)$ using $\mathrm{O}(n+m)$ processors.

To further optimize the number of processors used, we will not directly sample the point w from the sample space S_0. We construct the sample space S_0 by starting from an 2α-wise $n^{-\delta_2}$-biased sample space S_1 for $n\alpha$–length bit strings and taking $s = \mathrm{O}(1)$ samples from S_1, as in Section 3.1. (We assume here, as is true in our applications, that $q = \mathrm{O}(\log n)$.)

We claim that via the approach of Sections 3.2 and 3.3, each of the s samples can be chosen appropriately using $\mathrm{O}((n+m)2^{4\alpha}|S_1|)$ processors and $\mathrm{O}(t + \log n)$ time. To justify this, we just need to show that for any $s' \leq s$, conditional on:

- $r_k = r_k^*, 1 \leq k \leq (j-1)$,
- any given assignment for the first s' samples, and
- the remaining $s - s'$ samples being chosen independently at random from S_1, and, independently, $(r_{j+1}, r_{j+2}, \ldots, r_\beta) \sim U(\{0,1\}^{n\alpha(\beta-j)})$,

the expectation of a term such as $Profit_v(\ell(v))$ or $Cost_{u,v}(\ell(u), \ell(v))$ can be computed by a $(2^{4\alpha}, t + \log n)$ algorithm. Consider, say, a term such as $Cost_{u,v}(\ell(u), \ell(v))$. The result of choosing the remaining $s - s'$ samples from S_1 will lead to a value for r_j; since r_j will fix only 2α of the bits of $(\ell(u), \ell(v))$, the conditional expectation of $Cost_{u,v}(\ell(u), \ell(v))$ can be written as a sum

$$\sum_{\psi \in [2^{2\alpha}]} a_\psi p_\psi,$$

where the a_ψ are scalars and the p_ψ are probabilities that add up to 1. By the definition of t, all the a_ψ can be computed in $\mathrm{O}(t)$ time using $2^{2\alpha}$ processors. Each p_ψ in turn can be expressed as a sum of $2^{2\alpha}$ bias terms, where each individual bias term can be computed in constant time by one processor. Thus, the expectation of a term such as $Profit_v(\ell(v))$ or $Cost_{u,v}(\ell(u), \ell(v))$ can be computed by a $(2^{4\alpha}, t + \log n)$ algorithm; so each of the $s = \mathrm{O}(1)$ stages can be implemented by an $((n+m)2^{4\alpha}|S_1|, t + \log n)$ algorithm.

Plugging in the size of S_1 from the standard construction of [4] and using the fact that there are γ^{-1} stages, we get

THEOREM 4.1. *For any fixed $\gamma, \delta \in (0,1)$ and $q = \mathrm{O}(\log n)$, there is an $((n+m)2^{4\gamma q} n^\delta, \gamma^{-1}(t + \log n))$ algorithm to find a labeling $\hat{\ell}$ with $Benefit(\hat{\ell}) \geq E^* - n^{-\Theta(1)}$ in the PROFIT/COST problem, for the case where the functions Profit and Cost are bounded in value by a polynomial in n. In particular, there exists an $((n+m)n^\delta, t + \log n)$ algorithm to find such a labeling.*

As instantiations of this general framework we can derive the following as corollaries, via Theorem 4.1 and some results from [29] and [34]. 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 [29], with a running time of $O(\log^3 n \log\log n)$. It is shown in [29] that if there is a $(P(n,m), T(n,m))$ NC algorithm for the Profit/Cost problem when q and t are both $O(\log n)$, then $(\Delta+1)$-coloring can be solved by a $(P(n,m), \log n(\log n + T(n,m)))$ algorithm. In this instantiation the functions $Profit$ and $Cost$ actually lie in $[0,1]$. Using this in conjunction with Theorem 4.1 we get

COROLLARY 4.1. *There is an $((n+m)n^\gamma, \log^2 n)$ algorithm for $(\Delta+1)$-vertex coloring, where γ is any positive constant.*

A connected graph is vertex-colorable with Δ colors if and only if it is neither an odd cycle nor a complete graph. It is shown in [34] that if there is a $P(n,m)$–processor NC algorithm for $(\Delta+1)$-coloring that runs in $O(T(n,m))$ time, then there is a $(P(n,m), (\log n + T(n,m))\log^2 n/\log\Delta)$ algorithm for Δ-coloring graphs that are Δ-colorable. Thus, by Corollary 4.1, we get

COROLLARY 4.2. *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

Given a directed graph $G = (V, A)$, the maximum acyclic subgraph problem is to find a maximum sized subset of edges $\hat{A}$ such that the subgraph $G' = (V, \hat{A})$ is acyclic. This problem is known to be NP–hard. Let us assume that $V = [n]$. Building on the work of [13], Berger [11] derives an elegant NC algorithm which finds a subset $\hat{A}$ of size at least

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

where $d_{\text{out}}(v)$ and $d_{\text{in}}(v)$ denote the out- and in-degrees of vertex v in G, $deg(v) = d_{\text{out}}(v) + d_{\text{in}}(v)$, and $c_0 > 0$ is an absolute constant.

We now only present the portion of the work of [11] that is most relevant to our results. In the algorithm of [11], each vertex v is assigned 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 (other suitable values also exist for σ [11]). Each vertex v is then processed in parallel as follows. Let $N'_{\text{out}}(v) = \{(v,w) \in A : \ r_w > r_v\}$, and let $N'_{\text{in}}(v) = \{(w,v) \in A : \ r_w > r_v\}$. If $|N'_{\text{out}}(v)| \geq |N'_{\text{in}}(v)|$, then we choose the outedges from v to be in $\hat{A}$, else we choose the inedges. The resulting set of arcs $\hat{A}$ clearly defines an acyclic graph. It is shown in [11] that if the labels are assigned 5–wise independently with each label being uniformly random in $[\sigma]$, then $\mathbf{E}[|\hat{A}|] \geq target(G)$.

We will use the same approach as in the Profit/Cost problem to efficiently derandomize the assignment of labels. Since the approach is similar, we will merely outline a proof following some results in [11].

NOTATION 4.1. Given an event B, $\mathcal{I}(B)$ 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 $\Delta = \max_{v \in V} |N(v)|$.

The values $c_{v,S}$ in Theorem 4.2 can all be precomputed using $n\Delta^4$ processors, in $O(\log n)$ time.

THEOREM 4.2. **([11])** *For any choice of the labels r_v in the above algorithm, the resulting set of arcs $\hat{A}$ satisfies $|\hat{A}| \geq f(r_1, r_2, \ldots, r_n)$, where $f(r_1, \ldots, r_n)$ equals*

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

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

Thus, the goal is to compute a suitable set of labels $\{r_v : v \in V\}$ in NC, so that $f(r_1, \ldots, r_n) \geq target(G)$. As observed in [11], we could derive an $O(\log n)$ time NC algorithm from the above by exhaustive search of a 5-wise independent sample space, but that would use $O(n^5)$ processors. An $(n\Delta^4, \log^2 n \log \Delta)$ algorithm is presented in [11]; we now present an $(n^{1+\delta}\Delta^4, \log n)$ algorithm for any fixed $\delta > 0$.

Let $q = \log \sigma = \lceil \log n \rceil$. Suppose, for an arbitrary $j \in [q]$, the first j bits of the labels of all the variables have been fixed at arbitrary given values, and that the remaining $n(q-j)$ bits of the labels are chosen uniformly at random from $\{0,1\}^{n(q-j)}$. Then, it is shown in [11] that any conditional expectation such as $\mathbf{E}[\mathcal{I}(r_u = r_v)]$ or $\mathbf{E}[\mathcal{I}(r_v < \min\{r_w : w \in S\})]$ can be computed in $O(\log n)$ time using one processor. Motivated by this, we present a strategy similar to the one for the general profit/cost problem.

Let $\gamma \in (0,1)$ be a sufficiently small constant. We proceed in $\beta = \gamma^{-1}$ stages, and in each stage, we set γq bits of the labels of each of the vertices all at once. In each stage we need to sample from a sample space S_0 which is $(5\gamma q)$–wise $n^{-O(1)}$–biased, and we wish to maximize the sum of $O(n\Delta^4)$ functions (conditional expectations), each depending on at most $5\gamma q$ bits. As in Section 4.1, we will instead construct S_0 using the XOR of $O(1)$ samples from a $(5\gamma q)$–wise $n^{-\delta_1}$–biased sample space S_1. Proceeding identically as in Section 4.1, each stage can be implemented using $n\Delta^4 n^{O(\gamma+\delta_1)}$ processors and $O(\log n)$ time. We omit the details since the algorithm and the proof are almost exactly as the ones for the Profit/Cost problem.

THEOREM 4.3. *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 $|A|/2 - 1 + c_0(\sum_{i=1}^{n} \sqrt{deg(i)} + \sum_{i=1}^{n} |d_{\text{out}}(i) - d_{\text{in}}(i)|)$ arcs.*

5. APPROXIMATING GENERAL DISTRIBUTIONS

We have so far considered small spaces that approximate the distribution of independent and uniform binary random variables. With potentially similarly applications, the general problem of approximating the joint distribution D of independent multivalued random variables is considered in [18]: see Definition 1.4. We are interested in such approximating sample spaces that are small and efficiently constructible. An elementary probabilistic argument shows the *existence* of

a (k, ϵ)-approximation of cardinality as small as $O(n \log(n/\epsilon)/\epsilon^2)$ for any D and any $k \leq n$. However, for the purpose of derandomization, we need that these small sample spaces are *efficiently constructible* and no such constructions are known. Even though the constructions we present are of size bigger than the theoretically possible result, they are significantly smaller than the previously known constructions of (k, ϵ)-approximations.

Henceforth let e denote the base of the natural logarithm.

5.1. A geometric discrepancy problem

The efficient construction of (k, ϵ)-approximations can be reduced to the following *discrepancy* problem, as shown in [18]. 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 $0 \leq u_i < v_i \leq 1$. (As in [18], it is more convenient to work with intervals such as $[u_i, v_i)$, than with, say, $[u_i, v_i]$. One reason is that the complement of such a half-open interval $[u_i, v_i)$ within the universe $[0, 1)$, is the disjoint union of two such half-open intervals: $[0, u_i)$ and $[v_i, 1)$.) The volume of R, denoted $vol(R)$, is defined naturally to be $\prod_i (v_i - u_i)$. R is said to be *nontrivial* in dimension i if either $u_i > 0$ or $v_i < 1$. For any positive integer $k \leq n$, let $\mathcal{R}_n^k$ denote the set of axis-parallel rectangles that are nontrivial in at most k dimensions. 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 with respect to S, denoted $\Delta_S(R)$, to be $\left| \Pr(\vec{Y} \in R) - vol(R) \right|$. Note that $vol(R)$ is the probability that a point chosen randomly from $[0,1)^n$ lies 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, with respect to R. We define

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

This problem also arises in discrepancy theory [10] in the context of 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. For certain kinds of functions, there are bounds that show that sampling points from a small discrepancy set implies a small error.

The following lemma shows a link between the above geometric discrepancy problem and (k, ϵ)-approximations:

LEMMA 5.1. **([18])** *Let $\epsilon > 0$ and $S \subset [0,1)^n$ be such that $\Delta_S(\mathcal{R}_n^k) \leq \epsilon$, for integers k, n. For a 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 5.1, we shall focus on the problem of efficiently constructing a finite set $S \subseteq [0,1)^n$ such that it has a small "rectangle discrepancy", *i.e.*,

$$\text{for any } R \in \mathcal{R}_n^k, \Delta_S(\mathcal{R}_n^k) \leq \epsilon. \tag{3}$$

Of course, we also require S to be efficiently constructible, and be of small size. A major open question in derandomization theory is to construct S of size polynomial in k, $\log n$, and ϵ^{-1}. Actually, even a construction with cardinality polynomial in n and ϵ^{-1} will be very interesting. However, these seem to be difficult problems for now. The work of [18] describes efficient constructions of three sample spaces $S_1, S_2, S_3 \subset [0,1)^n$ to satisfy (3) with

$$|S_1| = \text{poly}(\log n, 2^k, 1/\epsilon),\ |S_2| = (n/\epsilon)^{\mathrm{O}(\log(1/\epsilon))},\ \text{and}\ |S_3| = (n/\epsilon)^{\mathrm{O}(\log n)}. \quad (4)$$

The sample spaces S_2 and S_3 actually guarantee (3) for all $k \leq n$. In contrast, our main result, given by Theorem 5.1 below, is an explicit construction of a sample space of size polynomial in $\log n$, $1/\epsilon$ and $(\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)}$, and which has discrepancy at most ϵ.

Let us first see why this size is at least as good as $|S_1|$, $|S_2|$ and $|S_3|$. As for $|S_1|$, we need only show that

$$\lceil k/\log(1/\epsilon)\rceil^{\log(1/\epsilon)} \leq 2^{\mathrm{O}(k)}.$$

This is immediate if $k \leq \log(1/\epsilon)$. If $k > \log(1/\epsilon)$, then for a fixed value of k, $(k/\log(1/\epsilon))^{\log(1/\epsilon)}$ is maximized when $\log(1/\epsilon) = k/e$. Thus, if $k > \log(1/\epsilon)$, we have

$$\lceil k/\log(1/\epsilon)\rceil^{\log(1/\epsilon)} \leq (2k/\log(1/\epsilon))^{\log(1/\epsilon)} \leq 2^{k/e}e^{k/e} \leq 2^k.$$

Next, it is easy to see that our construction is at least as good as S_2. Finally, as for S_3, note that if even if $k = n$, the logarithm of the size of our construction is $\mathrm{O}((\log n)\cdot\log(1/\epsilon))$; however, $\log|S_3| = \mathrm{O}(\log^2 n + (\log n)\cdot\log(1/\epsilon))$. Thus, our construction is an improvement over the previous ones. It also provides significant improvements in some cases. For instance, if $k = \log^b n$ and $\epsilon = n^{-\Theta(1)}$ with $b \geq 2$, our construction has size $n^{\mathrm{O}(\log\log n)}$, while the previously known constructions have size $n^{\Omega(\log n)}$.

The basic idea in our construction is to reduce the problem of finding a (k,ϵ)-approximate sample space to that of finding a (k',ϵ')-approximate sample space where $k' \ll k$ and ϵ' is not much smaller than ϵ. Then we can use the sample space S_1 of [18].

5.2. Construction of the sample space

We start with the following technical fact.

Proposition 5.1. *Suppose: (a) $a_1 \leq x_1 \leq a_1 + b_1$ and $a_2 \leq x_2 \leq a_2 + b_2$, or (b) $a_1 - b_1 \leq x_1 \leq a_1$ and $a_2 - b_2 \leq x_2 \leq a_2$ holds, for $b_1, b_2 \geq 0$. Then, $|x_1 - x_2| \leq |a_1 - a_2| + |b_1 - b_2| + b_2$.*

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

As stated above, our basic goal is to reduce the construction of (k,ϵ)-approximate sample spaces to that of constructing (k',ϵ')-approximate spaces with $k' \ll k$ and with ϵ' not much smaller than ϵ. We call this reduction of dimension from k to k' a *dimension reduction*.

We start with Lemma 5.2, which closely follows the proof of Theorem 2 in [18]. Lemma 5.2 will be crucial to our dimension reduction, and its basic setup is as follows. We have arbitrary binary random variables $Z_1, Z_2, \ldots, Z_k$, and *independent* binary random variables $X_1, X_2, \ldots, X_k$, and we want some conditions under which $\Pr(\bigwedge_{i\in[k]}(X_i = 0))$ is a "good" approximation for $\Pr(\bigwedge_{i\in[k]}(Z_i = 0))$. Lemma 5.2 shows that this is a good approximation if, for some "large" $k' \le k$ and for each $A \subseteq [k]$ with $|A| \le k'$, $\Pr(\bigwedge_{i\in A}(Z_i = 1))$ and $\Pr(\bigwedge_{i\in A}(X_i = 1))$ are sufficiently close to each other. See the statement of Lemma 5.2 for the actual bound.

Theorem 2 of [18] handles the version of this lemma where for each i, $\Pr(Z_i = 1) = \Pr(X_i = 1)$. We need the more general version of this lemma for our construction; however, as mentioned above, our proof closely follows that of Theorem 2 in [18].

LEMMA 5.2. *Let $X_1, X_2, \ldots, X_k$ be* independent *binary random variables and let $Z_1, Z_2, \ldots, Z_k$ be arbitrary binary random variables. Then, for any positive integer $k' \le k$, $\left|\, \Pr(\bigwedge_{i\in[k]}(Z_i = 0)) - \Pr(\bigwedge_{i\in[k]}(X_i = 0)) \,\right|$ is bounded by*

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

Proof. For any positive integer $\ell \le k'$, define

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

Let $\Pr(X_i = 1) = p_i$. We will consider two cases depending on the value of $\sum_i p_i$.

Case I: $\sum_{i\in[k]} p_i \le \frac{k'}{2e}$. As in [18], we have

$$\begin{aligned} C_{k'} &= \sum_{1\le i_1<\cdots<i_{k'}\le k} p_{i_1}\cdots p_{i_{k'}} \\ &\le \binom{k}{k'} \cdot ((\sum_i p_i)/k)^{k'} \\ &\le \frac{(k'/2e)^{k'}}{k'!} \\ &\le \frac{(k'/2e)^{k'}}{(k'/e)^{k'}} = 2^{-k'}. \end{aligned} \tag{5}$$

The first inequality follows from the fact that subject to a fixed value for $\sum_{i\in[k]} p_i$, $C_{k'}$ is maximized when all the p_i are the same.

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

if k' is even, then

$$a_1 \le \Pr(\bigwedge_{i\in[k]}(Z_i = 0)) \le a_1 + b_1 \text{ and } a_2 \le \Pr(\bigwedge_{i\in[k]}(X_i = 0)) \le a_2 + b_2;$$

if k' is odd, then

$$a_1 - b_1 \le \Pr(\bigwedge_{i\in[k]}(Z_i = 0)) \le a_1 \text{ and } a_2 - b_2 \le \Pr(\bigwedge_{i\in[k]}(X_i = 0)) \le a_2.$$

Thus, Proposition 5.1 shows that

$$\left| \Pr(\bigwedge_{i\in[k]}(Z_i=0)) - \Pr(\bigwedge_{i\in[k]}(X_i=0)) \right| \leq |a_1-a_2| + |b_1-b_2| + C_{k'}.$$

The bounds (5), $|a_1-a_2| \leq \sum_{\ell=1}^{k'-1}|B_\ell - C_\ell|$ and $|b_1-b_2| = |B_{k'}-C_{k'}|$ show that in Case I, the lemma holds.

Case II: In this case we have $\sum_{i\in[k]} p_i > \frac{k'}{2e}$. We will consider only enough of the p_is so that Case I can be applied and then bound the error in ignoring the remaining terms. Let $t < k$ be the first index such that $\frac{k'}{2e} - 1 < \sum_{i\in[t]} p_i \leq \frac{k'}{2e}$. Now, since $\sum_{i\in[t]} p_i \leq \frac{k'}{2e}$, it follows from Case I that

$$\begin{aligned} & \left|\Pr(\bigwedge_{i\in[t]}(Z_i = 0)) - \Pr(\bigwedge_{i\in[t]}(X_i = 0))\right| \\ \leq\ & 2^{-k'} + \sum_{\ell=1}^{k'} \sum_{A\subseteq[t]:|A|=\ell} \left| \Pr(\bigwedge_{i\in A}(Z_i=1)) - \Pr(\bigwedge_{i\in A}(X_i=1)) \right| \end{aligned} \tag{6}$$

Since $\sum_{i\in[t]} p_i > k'/(2e) - 1$ by assumption, we have

$$\begin{aligned} \Pr(\bigwedge_{i\in[t]}(X_i=0)) &= \prod_{i\in[t]}(1-p_i) \\ &\leq \left(1 - \frac{\sum_{i\in[t]} p_i}{t}\right)^t \\ &< \left(1 - \frac{(k'/2e)-1}{t}\right)^t \\ &\leq e\cdot e^{-k'/(2e)}. \end{aligned} \tag{7}$$

Now, the absolute value of the difference between $\Pr(\bigwedge_{i\in[k]}(Z_i=0))$ and $\Pr(\bigwedge_{i\in[k]}(X_i=0))$ will be maximized if one of these two can be zero and the other, as high as possible. Thus, by the obvious constraints

$$0 \leq \Pr(\bigwedge_{i\in[k]}(Z_i=0)) \leq \Pr(\bigwedge_{i\in[t]}(Z_i=0))$$

and

$$0 \leq \Pr(\bigwedge_{i\in[k]}(X_i=0)) \leq \Pr(\bigwedge_{i\in[t]}(X_i=0)),$$

we have

$$\begin{aligned} & \left|\Pr(\bigwedge_{i\in[k]}(Z_i=0)) - \Pr(\bigwedge_{i\in[k]}(X_i=0))\right| \\ \leq\ & \max\{\Pr(\bigwedge_{i\in[t]}(Z_i=0)), \Pr(\bigwedge_{i\in[t]}(X_i=0))\} \\ \leq\ & \left|\Pr(\bigwedge_{i\in[t]}(Z_i=0)) - \Pr(\bigwedge_{i\in[t]}(X_i=0))\right| + \Pr(\bigwedge_{i\in[t]}(X_i=0)). \end{aligned}$$

This, along with (5) and (6), completes the proof. ■

Suppose we are given n, k and ϵ, and wish to construct a $S \subset [0,1)^n$ such that $\Delta_S(\mathcal{R}_n^k) \leq \epsilon$. The basic idea of Theorem 5.1 below is to show, for some $k' = O(\log(1/\epsilon))$ and ϵ' "not much less than" ϵ, that it suffices to take any $S \subset [0,1)^n$ such that $\Delta_S(\mathcal{R}_n^{k'}) \leq \epsilon'$. The actual values of ϵ' and k' will be given in 11. Let us start with a bit of notation.

NOTATION 5.1. Fix any rectangle $R = [u_1, v_1) \times [u_2, v_2) \times \cdots \times [u_n, v_n)$ which is nontrivial in at most k dimensions: say, without loss of generality, in dimensions $1, 2, \ldots, k$. Consider any $S \subset [0,1)^n$ such that $\Delta_S(\mathcal{R}_n^k) \leq \epsilon$. Let $\vec{Y} = (Y_1, Y_2, \ldots, Y_n)$ be sampled uniformly at random from S. For each $i \in [k]$, let $Z_i \in \{0,1\}$ be the random variable that is 1 if $Y_i \notin [u_i, v_i)$, and let $X_i \in \{0,1\}$ be the random variable that is 1 if 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

$$\left| \Pr(\bigwedge_{i \in [k]} (Z_i = 0)) - \Pr(\bigwedge_{i \in [k]} (X_i = 0)) \right| \leq \epsilon. \tag{8}$$

By Lemma 5.2, it will suffice to show, for some $k' = C \cdot \log(1/\epsilon)$ where C is a sufficiently large constant, that for each $A \subseteq [k]$ with $|A| \leq k'$, $\Pr(\bigwedge_{i \in A}(Z_i = 1))$ and $\Pr(\bigwedge_{i \in A}(X_i = 1))$ are sufficiently close to each other. Since $\Delta_T(\mathcal{R}_n^{k'}) \leq \epsilon'$, we only know that for each $A \subseteq [k]$ with $|A| \leq k'$,

$$\left| \Pr(\bigwedge_{i \in A} (Z_i = 0)) - \Pr(\bigwedge_{i \in A} (X_i = 0)) \right| \leq \epsilon';$$

however, we wish to bound $\left| \Pr(\bigwedge_{i \in A}(Z_i = 1)) - \Pr(\bigwedge_{i \in A}(X_i = 1)) \right|$. Lemma 5.3 helps in this.

LEMMA 5.3. *Let $S \subset [0,1)^n$ be a set with $\Delta_S(\mathcal{R}_n^{k'}) \leq \epsilon'$, for some k' and ϵ' and 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 for* any *u_i, v_i with $0 \leq u_i < v_i \leq 1$, $i \in J$, we have*

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

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

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

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

Our construction of the small sample space is described now, following the above informal outline.

THEOREM 5.1. *There is an explicitly constructible set $S \subseteq [0,1)^n$ of size polynomial in $\log n$, $1/\epsilon$ and $(\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$, with discrepancy bounded by ϵ. Thus, the joint distribution of any n* independent *discrete random variables, has an explicit (k,ϵ)-approximating sample space of size polynomial in $\log n$, $1/\epsilon$ and $(\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$.*

Proof. Given k and ϵ, define

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

Observe that $k' = \mathrm{O}(\log(1/\epsilon))$, and hence $1/\epsilon' = \mathrm{poly}(1/\epsilon, (\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$. Given the parameters n, k' and ϵ', one could use the first construction S_1 presented in [18] to explicitly construct a set $S \subset [0,1)^n$, with size polynomial in $\log n$, $1/\epsilon'$, and $2^{k'}$, such that $\Delta_S(\mathcal{R}_n^{k'}) \leq \epsilon'$. By definition of ϵ' and k', we have $|S| = \mathrm{poly}(\log n, 1/\epsilon, (\lceil k/\log(1/\epsilon)\rceil)^{\log(1/\epsilon)})$. In the following analysis, we will assume that $k' < k/3$: if $k' \geq k/3$, we have $k = \mathrm{O}(\log(1/\epsilon))$, and hence we can define the constructed sample space S to be the construction S_1 of [18].

Our claim is that S also satisfies $\Delta_S(\mathcal{R}_n^k) \leq \epsilon$. To see this, let R, $\{X_i : i \in [k]\}$, $\vec{Y} = (Y_1, Y_2, \ldots, Y_n)$, and $\{Z_i : i \in [k]\}$ be as in Notation 5.1. We now prove (8). Lemmas 5.2 and 5.3 show that

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

Since we have assumed that $k' < k/3$,

$$\begin{aligned}\sum_{\ell=1}^{k'}\binom{k}{\ell}2^\ell &\leq 2\cdot\binom{k}{k'}\cdot 2^{k'} \\ &\leq 2\cdot\frac{k^{k'}}{\left(\frac{k'}{e}\right)^{k'}}\cdot 2^{k'} \\ &\leq \left(\frac{ke^2}{k'}\right)^{k'}.\end{aligned}$$

Thus,

$$\left|\ \Pr(\bigwedge_{i\in[k]}(Z_i=0)) - \Pr(\bigwedge_{i\in[k]}(X_i=0))\ \right| \leq 2^{-k'} + e\cdot e^{-k'/(2e)} + \epsilon'\left(\frac{ke^2}{k'}\right)^{k'}.$$

By our choice of the parameters ϵ' and k', the right-hand-side is at most ϵ. Thus the discrepancy of the set S is bounded by ϵ, and by Lemma 5.1 we can construct a (k, ϵ)-approximate sample space with the required bounds. ■

6. DISCUSSION

We have devised a new method to construct sample spaces whose Fourier coefficients are close to those of the distribution of n independent and unbiased random bits. Though the size of the sample space so constructed is larger than that 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 are motivated by their use as pseudorandom generators and in derandomization. For this problem, we have presented a construction that improves on the previously known constructions.

Furthermore, there is a natural related problem of *approximating combinatorial rectangles* [7, 9], defined as follows. Let m and n be positive integers. A *combinatorial rectangle* R is now a cross-product of the form $S_1 \times S_2 \times \cdots \times S_n$, where $S_i \subseteq \{0, 1, \ldots, m-1\}$ for each i; R is called *trivial in dimension* i iff $S_i = \{0, 1, \ldots, m-1\}$. What we require in this problem is a "small" and efficiently constructible multiset $X \subseteq \{0, 1, \ldots, m-1\}^n$ such that for $\vec{X} = (X_1, X_2, \ldots, X_n)$ sampled uniformly at random from X, we have $\left| \Pr(\vec{X} \in R) - (\prod_{i=1}^{n} |S_i|)/m^n \right| \leq \epsilon$ for all combinatorial rectangles R that are nontrivial in at most k dimensions. Some constructions of such X are now known [7, 9]; the work of [9] uses our construction of (k, ϵ)-approximations as a tool.

ACKNOWLEDGMENT

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. We also thank the referee for his/her detailed and valuable comments, which have substantially improved the presentation of the ideas in this paper.

REFERENCES

1. W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr, RSA/Rabin functions: certain parts are as hard as the whole, *SIAM J. Comput.*, 17:194–209, 1988. A preliminary version appeared as: RSA/Rabin bits are 1/2 + 1/poly(log N) secure, in *Proc. IEEE Symposium on Foundations of Computer Science*, pages 449–457, 1984.

2. 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.

3. 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.

4. 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.
5. N. Alon and M. Naor, Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions, *Algorithmica*, 16:434–449, 1996.
6. N. Alon and A. Srinivasan, Improved parallel approximation of a class of integer programming problems, *Algorithmica*, 17:449–462, 1997.
7. R. Armoni, M. Saks, A. Wigderson, and S. Zhou, Discrepancy sets and pseudorandom generators for combinatorial rectangles, in *Proc. IEEE Symposium on Foundations of Computer Science*, pages 412–421, 1996.
8. R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou, $SL \subseteq L^{4/3}$, in *Proc. ACM Symposium on Theory of Computing*, pages 230–239, 1997.
9. P. Auer, P. M. Long, and A. Srinivasan, Approximating hyper-rectangles: learning and pseudorandom sets, *Journal of Computer and System Sciences*, 57:376–388, 1998.
10. J. Beck and W. Chen, Irregularities of distribution, Cambridge University Press, 1987.
11. B. Berger, The fourth moment method, *SIAM J. Comput.*, 26:1188–1207, 1997.
12. B. Berger and J. Rompel, Simulating $(\log^c n)$-wise independence in NC, *Journal of the ACM*, 38:1026–1046, 1991.
13. B. Berger and P. W. Shor, Tight bounds for the maximum acyclic subgraph problem, *Journal of Algorithms*, 25:1–18, 1997.
14. M. Blum and S. Micali, How to generate cryptographically strong sequences of random bits. *SIAM J. Comput.*, 13:850–864, 1984.
15. B. Chazelle and J. Matoušek, On linear-time deterministic algorithms for optimization problems in fixed dimension, *Journal of Algorithms*, 21:579–597, 1996.
16. B. Chor, J. Friedman, O. Goldreich, J. Hastad, S. Rudich, and R. Smolensky. The bit extraction problem or t–resilient functions. in *Proc. IEEE Symposium on Foundations of Computer Science*, pages 396–407, 1985.
17. B. Chor and E. Goldreich. On the power of two–point based sampling, Journal of Complexity, Vol 5, pages 96–106, 1989.
18. 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. Journal version: G. Even, O. Goldreich, M. Luby, N. Nisan, and B. Veličković, Efficient approximations for product distributions, *Random Structures & Algorithms*, 13: 1–16, 1998.
19. 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.
20. Y. Han, A fast derandomization scheme and its applications, *SIAM J. Comput.*, 25:52–82, 1996.
21. T.-s. Hsu, *Graph augmentation and related problems: theory and practice*, PhD thesis, Department of Computer Sciences, University of Texas at Austin, October 1993.
22. 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.
23. A. Joffe, On a set of almost deterministic k–independent random variables, *The Annals of Probability*, 2(1):161–162, 1974.
24. H. J. Karloff and P. Raghavan, Randomized algorithms and pseudorandom numbers, *Journal of the ACM*, 40(3):454–476, 1993.
25. R. M. Karp and V. Ramachandran, A survey of parallel algorithms for shared memory machines, in *Handbook of Theoretical Computer Science* (J. Van Leeuwen Ed.), Chapter 17, MIT Press, pages 871–941, 1990.
26. R. M. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, *Journal of the ACM*, 32:762–773, 1985.
27. M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM J. Comput.*, 15(4):1036–1053, 1986.
28. M. Luby, Removing randomness in parallel computation without a processor penalty, In *Proc. IEEE Symposium on the Foundation of Computer Science*, pages 162–173, 1988. (This is a preliminary version of [29].)

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

30. S. Mahajan, E. A. Ramos, and K. V. Subrahmanyam, Solving some discrepancy problems in NC, In *Proc. Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 1346, Springer, pages 22–36, 1997.

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

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

33. 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.

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

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

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

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

38. J. H. Spencer, *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.

39. A. C. C. Yao, Theory and Application of Trapdoor Functions, in *Proc. IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.