A Case Study of De-randomization Methods for Combinatorial Approximation Algorithms*

José D. P. Rolim† Luca Trevisan‡

October 30, 1997

Abstract

We study three different de-randomization methods that are often applied to approximate combinatorial optimization problems. We analyze the *conditional probabilities* method in connection with randomized rounding for routing, packing and covering integer linear programming problems. We show extensions of such methods for non-independent randomized rounding for the assignment problem. The second method, the so called *random walks* is exemplified with algorithms for dense instances of some NP problems. Another often used method is the *bounded independence* technique; we explicit this method for the sparsest cut and maximum concurrent flow problems.

1 Introduction

Randomized algorithms are often used to solve or to approximate optimization problems related to network design. Theoretical and practical concern about the effective availability of a source of truly random bits, as well as the desire of improved reliability, motivate the search for deterministic versions of such algorithms.

While there is no general recipe to come up with the de-randomization of a randomized algorithm, a small set of techniques seem to underlie most

*This research was supported by the Swiss National Science Foundation under grant no. 50003-034295.

†Centre Universitaire d'Informatique, University of Geneva, 24 rue General Dufour, 1204 Geneva, Switzerland. rolim@cui.unige.ch

‡MIT Laboratory for Computer Science, Room NE43-371, 545 Technology Square, Cambridge MA 02139, USA. luca@theory.lcs.mit.edu (Work done at the University of Geneva)

of the known results, such as the method of *conditional probabilities*, the use of *random walks* in expander graphs and *k-wise independency* techniques. In general, randomization is used to provide a combinatorial structure with certain properties, using the *probabilistic method*. In such cases, a deterministic construction of such an object can be a very hard task; a good rule of the thumb is to prove that an object with somewhat weaker properties suffices for the algorithm, and then develop a deterministic construction of an object of the latter type. The particular de-randomization method to be applied will depend on which properties are to be preserved in the deterministic version of the algorithm. In this paper we will see a number of examples of randomized approximation algorithms that can be de-randomized with one of the above mentioned techniques.

Several approximation algorithms for network design problems are somewhat related to linear programming (or generalizations of linear programming). Once an optimization problem is formulated as an integer mathematical programming problem, the relaxation over the reals can be used to develop approximation algorithms in one of the following way:

1. Primal/dual algorithms: the algorithm finds a feasible solution for the problem of interest and a feasible solution for the dual of its relaxation; the ratio between the costs of the two solutions is at most some constant r. Then the solution is r-approximate.

2. Rounding: an optimum (or near-optimum) solution is found for the relaxation, and it is rounded to yield an integer solution. The ratio between the cost of the rounded solution and the fractional one is at most r. Then the rounded solution is r-approximate.

In the first case, the linear programming formulation is used just as a conceptual tool to prove the approximation bound. In the second case, the relaxation is actually solved. Rounding a fractional solution is, in general, a difficult task. Randomization is a primary tool. The algorithms we shall consider in this paper use randomization to find a rounded solution. Despite the similarity in the structure, the algorithms are quite different, and their de-randomization requires different techniques, that are a good illustration of the different ways a randomized algorithm can be de-randomized.

In general, the way randomization is used in the algorithm calls for the proper de-randomization method. The method *of conditional probabilities* (and its generalizations) is suitable when we have a set of 0/1 values $x_1, \ldots, x_n$ that are chosen independently with probability $p_1, \ldots, p_n$, and the objective function is a linear combination of such values. Randomized rounding falls in this case. Another classical use of randomness is to *sample* a small set of points from some larger space. Several methods are known to sample using few random bits. In several cases, the number of random bits can be made logarithmic and thus a trivial exhaustive de-randomization is possible. In some cases, randomization is used to construct combinatorial structures with variables that are k-wise independent. The de-randomization technique then consists in finding a probability distribution which assigns a non-zero probability to only a polynomial number of solutions while preserving the k-wise independency. On this way an approximation solution is guaranteed to exist and it can be searched in polynomial time deterministically.

The above three scenarios that are themselves a random sample out of a somewhat larger variety and a combination of them are explained via some case studies.

Randomized rounding for routing, and packing and covering ILP. Randomized rounding was introduced in Raghavan and Thompson (1987), Raghavan (1988). It allows to approximate routing problems as well as problems that can be formulated as integer linear programs in packing or covering form. The approximation guarantee is proved using Chernoff bounds. De-randomization involves the use of *pessimistic estimators*, a generalization of the method of conditional expectation. Srinivasan (1996) has improved Raghavan's bound for set cover, and has provided pessimistic estimators to de-randomize his algorithm.

Non-independent randomized rounding for the assignment problem. Arora, Frieze and Kaplan (1996) develop a version of randomized rounding for assignment problems (where the solution is a one-one function between two sets of n objects). Applications include the linear arrangement problem in dense graphs. Randomization is used in two phases of their algorithm: an *oversampling* phase and a *merging* phase. Chernoff bounds are used in the proof of correctness. De-randomization uses (de)randomized rounding to simulate the sampling and merging.

Random walks for optimization problems. Dense instances of many NP-hard problems have been given polynomial time approximation algorithms by Arora, Karger and Karpinski (1995). Their approach is based on an exhaustive sampling and a subsequent phase of rounding. The exhaustive sampling is de-randomized via random walks on expanders while the second phase is de-randomized via conditional probabilities.

Min Cut. Linial, London and Rabinovich (1994) (and independently Aumann and Rabani (1994)) show that the sparsest cut and its dual the concurrent flow problem can be approximated within a factor $O(\log k)$. The algorithm uses a mapping of a certain metric (obtained by solving a linear programming relaxation of the problem) into a ℓ_1 metric. Randomization is used to generate the mapping. Linial, London and Rabinovich (1995) have successively found a deterministic way of constructing the mapping (even if it requires more dimensions than the original one). Garg (1995) has independently found a deterministic algorithm for the special case of sparsest cut.

Open questions. We remark that for some important network optimization problems, only randomized approximation algorithms are known. One example is the all-terminal network reliability problem (Karger (1995)), a $\#P$-hard counting problem. A more important one is the Euclidean TSP and Steiner Tree problems, for which Arora [Aro97] developed randomized quasi-linear time approximation schemes. Arora's algorithms are trivially de-randomizable, but the trivial de-randomization increases the running time. A quasi-linear time de-randomized algorithm would be a major result.

2 Random Rounding and Conditional Expectation

The framework is as follows. Say that we have a probability space $(\mathbf{Pr}, \{0,1\}^n)$ and a set of "good strings" $A \subseteq \{0,1\}^n$ such that $\Pr(A) \geq \epsilon$ for some $\epsilon > 0$; assume also that the algorithm that we want to to de-randomize uses randomness only to find a string $\mathbf{x} \in A$.

A deterministic algorithm can construct such a string in the following way. For $i = 0, 1, \ldots, n$ and $(b_1, \ldots, b_i) \in \{0,1\}^i$, let us call $P^i_{b_1,\ldots,b_i} = \Pr[\mathbf{x} \notin A | x_1 = b_1, \ldots, x_i = b_i]$. Of course we have $P^0 = 1 - \Pr[A] < 1$, moreover we have that, for any i and any $(b_1, \ldots, b_i)$, either $P^{i+1}_{b_1,\ldots,b_i,0} \leq P^i_{b_1,\ldots,b_i}$ or $P^{i+1}_{b_1,\ldots,b_i,1} \leq P^i_{b_1,\ldots,b_i}$. Observe that $P^n_{b_1,\ldots,b_n}$ is either equal to 0 or 1, and is equal to 1 if and only if $(b_1, \ldots, b_n) \notin A$.

algorithm `cond-prob`
for $i = 1$ **to** n **do**
  if $P^i_{b_1,\ldots,b_{i-1},0} \leq P^{i-1}_{b_1,\ldots,b_{i-1}}$ **then**
    $b_i := 0$
  else
    $b_i := 1$;
return $(b_1, \ldots, b_n)$;

The above algorithm maintains the invariant that, at any iteration of the **for** loop, b_i receives a value such that $P^i_{b_1,\ldots,b_i} \leq P^{i-1}_{b_1,\ldots,b_{i-1}}$. By induction, this implies that $P^n_{b_1,\ldots,b_n} \leq P^0 < 1$, and so $P^n_{b_1,\ldots,b_n} = 0$ and $(b_1,\ldots,b_n) \in A$.

This method could, in principle, be applied to almost any randomized algorithm. In practice, however, the computation of the conditional probabilities is an utterly complicated task.

A careful examination of the proof of correctness of the method reveals that it is not really necessary to exactly compute all the conditional probabilities. This is formalized below

Definition 2.1 *A* pessimistic estimator *for set A and probability distribution* $\Pr$ *is a set of values $\{U^i_{b_1,\ldots,b_i}\}^{i=0,\ldots,n}_{(b_1,\ldots,b_i)\in\{0,1\}^i}$ such that the following properties hold.*

1. *$U^0 < 1$;*
2. *$\forall i = 0,\ldots,n-1$, for all $(b_1,\ldots,b_i) \in \{0,1\}^i$, $U^i_{b_1,\ldots,b_i} \geq \min\{U^{i+1}_{b_1,\ldots,b_i,0}, U^{i+1}_{b_1,\ldots,b_i,1}\}$;*
3. *$\forall i = 0,\ldots,n$, for all $(b_1,\ldots,b_i) \in \{0,1\}^i$, $P^i_{b_1,\ldots,b_i} \leq U^i_{b_1,\ldots,b_i}$.*

If we have an algorithm that computes $U^i_{b_1,\ldots,b_i}$ in $\mathrm{poly}(n)$ time, then we are done: we can run algorithm `cond-prob` using $U^i_{b_1,\ldots,b_i}$ in place of $P^i_{b_1,\ldots,b_i}$. The proof of correctness is the same.

Attributions and Related Work. *The method of conditional probabilities is a standard way to obtain deterministic constructions out of an existence proof that involves the probabilistic method. A clear exposition is in [AS92]. Pessimistic estimators are defined in [Rag88].*

2.1 Rounding Integer Linear Programs

Randomized rounding is an algorithmic technique that is suitable of derandomization using conditional probabilities. The general framework is as follows: we have a problem that can be formulated as an integer linear program (ILP) with 0/1 variables. We relax the ILP to a linear program (LP) and we solve it to optimality. Then, we interpret the fractional solution obtained on this way as a probability distribution over the variables. The constraints of the LP are satisfied with high probability and the expected value of the objective function is close to the value of the relaxation.

Theorem 2.2 *Let* $\mathbf{x} = (x_1, \ldots, x_n)$ *be a vector satisfying* $\mathbf{a}^T \cdot \mathbf{x} = b$, $0 \leq x_i \leq 1$. *Define the random variables* $y_1, \ldots, y_n$ *such that* y_i *is equal to 1 with probability* x_i *and equal to 0 with probability* $(1 - x_i)$ *then, for any* $f > 0$, *the following holds with probability at least* $(1 - n^{-f})$*:*

$$b - a_{\max}\sqrt{fn \log n} \leq \mathbf{a}^T \cdot \mathbf{y} \leq b - a_{\max}\sqrt{fn \log n} \tag{1}$$

where $a_{\max} = \max_i |a_i|$.

The Theorem also holds for rounding inequalities. It is also clear that the Theorem can be extended to systems of m equations; in this case the error will be $a_{\max}\sqrt{fn \log mn}$. Let $\mathbf{x}'$ be a fractional solution to the following linear program.

$$\begin{array}{ll} \max & \mathbf{c}^T \cdot \mathbf{x} \\ \text{Subject to} & \\ & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \end{array}$$

Construct an integer solution $\mathbf{y}$ probabilistically by setting, independently for any j, $y_j = 1$ with probability x'_j and $y_j = 0$ with probability $1 - x'_j$. With high probability, the resulting solution has cost at least $(1 - o(1))\mathbf{c}^T \cdot \mathbf{x}'$ and satisfies

$$A\mathbf{y} \leq b + O(M_A\sqrt{nlogn}) \tag{2}$$

where M_A is the largest entry of A.

The above fundamental results have direct application to randomized approximation algorithms for routing problems and for problems expressible as integer linear programs in packing or covering forms (that generalize, respectively, hypergraph matching and set cover).

De-randomization uses a pessimistic estimator and yield the following result.

Theorem 2.3 *There exists a polynomial time algorithm that given a vector* $\mathbf{x} = (x_1, \ldots, x_n)$, $0 \leq x_i \leq 1$, *and a* $m \times n$, *matrix* A, *finds a vector* $\mathbf{y} = y_1, \ldots, y_n \in \{0, 1\}^n$ *such that*

$$A\mathbf{x} - O(A_{\max}\sqrt{n \log n}) \leq A\mathbf{y} \leq A\mathbf{x} + O(A_{\max}\sqrt{n \log mn}) \tag{3}$$

where $a_{\max} = \max_{i,j} |a_{i,j}|$. *In addition, for the rows of* A *with all non-negative entries, the stronger bound*

$$\sum_i a_{i,j} x_j - O(a_j^{\max} \log n) \leq \sum_j a_{i,j} y_j \leq \sum_i a_{i,j} x_j + O(a_j^{\max} \log n)$$

holds, where $a_j^{\max} = \max_i |a_{i,j}|$.

The very notion of pessimistic estimator has been introduced in order to prove Theorem 2.3 [Rag88].

ATTRIBUTIONS AND RELATED WORK. *Random rounding has been introduced by Raghavan and Thompson [RT87]. The de-randomized rounding using a pessimistic estimator is due to Raghavan [Rag88]. Both results are also presented in Raghavan's PhD Thesis [Rag86].*

Random rounding has been used to develop approximation algorithms for the Maximum Satisfiability problem [GW94] and the Constraint Satisfaction problem [Tre96]. In these algorithms, the probability distribution used to round the variables is not the solution of the relaxation, but rather a convex combination of the solution of the relaxation and of the uniform distribution. The de-randomization of these algorithms is easier since, basically, any 0/1 solution is feasible.

Theorem 2.2 can be improved in special cases, for example for resource-constrained scheduling problem [SS96] and for packing and covering integer linear programs [Sri95]. In both cases, the authors de-randomize their rounding schema using new pessimistic estimators.

3 Extensions of Random Rounding and Random Walks

Another useful de-randomization technique is based on random walks on graphs. Let $G(V, E)$ be an undirected graph and let $v_1, v_n \in V$ and $|v| = n$. A randomized naive algorithm for testing a path from v_1 to v_n would start at vertex v_1 and chose an edge leaving v_1 at random, follow this edge to a new vertex and repeat the procedure. This algorithm defines a *random walk* and has many implications in de-randomization. The *random walk* on a graph corresponds to a *Markov chain* with states represented by the vertices of the graphs. Let G be a connected non-bipartite undirected weighted graph; the probability associated with a transition from a vertex v_i to v_j is given by $p_{ij} = \frac{w_{ij}}{w_i}$ with w_{ij} the weight for the edge between v_i and v_j and

$w_i = \sum_{(i,j)} w_{ij}$ or 0 if v_i and v_j are not connected. Notice that the probability of being on a fixed state represented by a vertex is $\frac{1}{n}$ independently of the particular vertex. Let A be a subset of E and consider the time the random walk spends in A; this time for almost every trajectory converges to the limiting probability of A, $\pi(A)$. Notice that if $A = E$ then $\pi(E) = \frac{1}{n}$ the stationary probability.

Estimating the value of $\pi(A)$ is an essential tool for de-randomizing algorithms based on the sampling of exponentially large sets. The basic idea is to generate a number of random sample points in G and compute the fraction in A. This may be done by the use of the *rapid mixing property* of the random walk on G to generate a single nearly random sample point from π and repeating the procedure to generate enough independent samples points in order to apply an appropriate Chernoff bound. An alternative way, which gives better results, is first to generate a nearly random starting point and then to sample every point along a single trajectory of the random walk [Gil93]. It is shown there that the convergence to $\pi(A)$ has error probability exponentially small in the length of the random walk and the square of the size of the deviation from $\pi(A)$. For the uniform probability distribution $\pi(A) = \rho$, if $|A| = \rho n$.

The bound depends on the degree of expansion of a graph. An expander is a graph G with the property that any subset S of vertices of G containing at most $\frac{n}{2}$ vertices is connected to at least $O(|S|)$ vertices not in S. The better an expander G is, the more likely the random walk will visit A a fraction of time approximating $\pi(A)$. Also the uniformity of the graph improves the bound and if the random walk starts in the stationary distribution the bound does not depend on the size of the graph. This bound can be stated in terms of random walks as follows:

Theorem 3.1 ([Gil93]) *Let $G = (V, E)$ be a constant degree expander and let $A \subseteq V$ with $|A| = \rho n$ and $|V| = n$. Consider a random walk of length l on G with uniform probability and let t_l be the number of vertices of A in the andom walk. Then:*

$$Pr[|\frac{t_l}{l} - \rho| \geq \epsilon] \leq 2^{-l\epsilon^2 K}$$

where K is a constant depending on the graph G.

The application for sampling works basically as follows. In order to sample $O(m)$ points almost uniformly from a set S, we build a constant degree expander with vertex set S and set up a random walk of length m on S. The vertices on the random walk correspond to the m points to be sampled [Gil93]. Deterministically, we can go through all possible paths of length m and one of them will almost surely correspond to a set of m points uniformly sampled on set S.

3.1 Approximation of "Dense" Instances of NP-hard Problems

We present an application to an approximation scheme for dense instances of NP-hard problems. It applies to problems such as Max CUT and Max Bisection.

The general theorem is as follows

Definition 3.2 *For a constant c, a polynomial in n variables of total degree d is said to be c-smooth if the coefficient of the monomial of degree i is at most cn^{d-i}.*

For example a linear function $a_1 x_1 + \ldots + a_n x_n + b$ is c-smooth if $a_i \leq c$ for $i = 1, \ldots, n$ and $b \leq cn$.

Theorem 3.3 *There exists a randomized polynomial time algorithm that given a rational $\epsilon > 0$ and the following mathematical programming problem*

$$\begin{array}{ll} \max & p(x_1, \ldots, x_n) \\ \text{Subject to} & \\ & \mathbf{x} \in \{0,1\}^n \end{array}$$

where p is a c-smooth degree-d polynomial, finds a feasible solution $\mathbf{y}$ in time $O(n^{\mathrm{poly}(c,d,1/\epsilon)}$ whose value is within an additive factor ϵn^d from the optimum.

For $d = 1$ the theorem is obvious (indeed, it is trivial to find an optimum solution). The most interesting case is for $d = 2$, since the cases with higher degree can be reduced to the case of degree 2. Furthermore, the application to Max CUT and Min Bisection only require degree 2 formulations.

The algorithm has two phases. In the first phase a relaxation of the problem is considered over the reals, and an approximate solution is found. Then the approximate (fractional) solution is rounded to yield an integer solution. The first phase uses a reduction to linear programming; the reduction uses random sampling and can be de-randomized using random walks on expanders. The second phase uses the randomized rounding of the previous section, and is de-randomized, as already seen, with the method of the pessimistic estimator. We briefly outline the algorithm. For simplicity, we specialize the presentation to the case of the Max CUT problem.

Recall that in the Max CUT problem an undirected graph is given and the goal is to find a partition of the set of its vertices so as to maximize the number of edges having an endpoint in each side of the partition.

A graph is said to be δ-dense if it has n vertices and at least δn^2 edges. It is easily seen that in any graph with m nodes there always exists a cut of cost $\geq \lceil m/2 \rceil$. Since the algorithm of Arora, Karger amd Karpinski guarantees an additive error ϵn^2, it is $(1 + 2\epsilon/\delta)$-approximate in δ-dense graphs.

Given a graph $G = (V, E)$, $V = \{1, \ldots, n\}$ the Max CUT problem can be formalized with the following quadratic programming problem, having a variable x_i for any vertex i:

$$\begin{array}{ll} \max & \sum_{(i,j)\in E} x_i(1-x_j) + x_j(1-x_i) \\ \text{Subject to} & \\ & \mathbf{x} \in \{0,1\}^n \end{array} \tag{4}$$

Observe that the program above is 2-smooth. It is convenient to reformulate the objective function as $\sum_i x_i \sum_{(i,j)\in E}(1 - x_j)$.

Approximating a Fractional PIP

The intuition behind the algorithm is based on the fact that in the optimum solution each vertex has the majority of the vertices connected to it in the opposite subset or otherwise the solution would not be optimum. In order to decide in which subset a particular vertex should be, the algorithm takes a sample of $O(\log n)$ vertices of the graph and exhaustively tries all the $2^{O(\log n)}$ possibilities of placements of the vertices in the sample. For any such partial solution, the rest of the algorithm is executed. At the end, among the $2^{O(\log n)}$ solution that have been sequentially computed, the best one is selected. While considering the rest of the algorithm, the reader may assume, without loss of generality, to consider the case when the placements of the sampled vertices is done according to an optimum solution. Let $\mathbf{y}^*$ be such solution.

For the unsampled vertices a decision is taken based on the majority rule over the sampled neighbors. Notice that with high probability every vertex has some of its neighbors sampled. Since the sampling involves some error, it is difficult to decide where to put a vertex, whenever the sampled adjacent vertices are almost equally distributed in the two halves of the partition. For any vertex i, let r_i be the fraction of sampled vertices that are adjacent to i and have been placed in the right part of the partition. We expect $r_i n$ to be an estimation of $\sum_{(i,j)\in E}(1-y_j^*)$, within an additive error ϵn. These observations reduce the PLP (4) to the following LP:

$$\begin{array}{ll} \max & \sum_i x_i r_i \\ \text{Subject to} & \\ & r_i - \epsilon n \leq \sum_{(i,j)\in E}(1-x_j) \leq r_i n + \epsilon n \end{array} \tag{5}$$

It should come as no surprise that, with high probability (over the choice of the sample points), if the sample points are partitioned according to an optimal solution, (5) is feasible and an optimum solution for it has a cost that is at least the optimum of the relaxation of (4) minus ϵn^2.

In this first phase, randomization is only used to generate the sample points. The point have to satisfy $O(n^2)$ conditions; thus each condition has to be satisfied with probability at least $1-\delta$ where $\delta = 1/O(n^2)$. The additive error that is admitted in each condition is ϵ. Under such condition, a random walk of length $O(1/\epsilon^2 \log 1/\delta) = O(\frac{1}{\epsilon^2}\log n)$ on a constant-degree expander will give the required sample space. There only $n^{O(1/\epsilon^2)}$ such walks, so de-randomization can be done in polynomial time. An alternative way of de-randomizing this sampling phase is via a technique called *iterated sampling* [BR94] which is also an oblivious method using less random bits but more points than the method just described.

Rounding a Fractional PIP Solution

In the second phase, the fractional solution obtained by solving (5) is converted into a 0/1 solution by loosing another additive factor $O(\sqrt{n\log n})$.

This phase uses randomized rounding as in Theorem 2.2. An extension of such theorem is needed to handle the case of smooth polynomials rather than linear functions.

Theorem 3.4 *Let p be a c-smooth degree-d polynomial. Given fractional values $\mathbf{x} \in [0,1]^n$, suppose random rounding is performed as in Theorem 2.2 to yield a vector $\mathbf{y} \in \{0,1\}^n$. Then, with probability at least $1-n^{d-f}$, we have*

$$p(\mathbf{x}) - c\sqrt{f\log n}\, dn^{d-\frac{1}{2}} \leq p(\mathbf{y}) \leq p(\mathbf{x}) + c\sqrt{f\log n}\, dn^{d-\frac{1}{2}}$$

The theorem is proved by induction on d. The base case ($d=1$) is Theorem 2.2 and the inductive step uses Theorem 2.2 again. It follows that the de-randomization of Theorem 2.2 yields the de-randomization of the theorem above.

Remark 3.5 *When the Max CUT problem is formulated as a PIP, the objective function is a* multilinear *polynomial. For multilinear polynomials, a stronger version of Theorem 3.4 holds, namely, it is possible to find deterministically in polynomial time an integral solution* $\mathbf{x}$ *such that* $p(\mathbf{x}) \geq p(\mathbf{y})$*. Also, all the dense versions of Max SNP problems, when formulated as PIP, have a multilinear objective function.*

Attributions and Related Work. *Approximation schemes for dense problems are in [AKK95]. A more efficient randomized approximation scheme for dense Max CUT is in [dlV]. An alternative more efficient approximation scheme for dense Max CUT and Max* k*-CUT has been developed in [FK96] using an algorithmic version of Szemeredy's regularity lemma. A linear-time randomization approximation scheme for dense Max CUT appears in [GGR96]. An efficient de-randomization of the algorithm of [GGR96] is still an open question. Interestingly, the algorithm of [GGR96] runs in* constant time*, on the unit-cost RAM model of computation, if it is only required to output a* value *that approximates the Max CUT (rather than an approximately optimum* partition*).*

4 Advanced Use of Rounding: the Assignment Problem

Another example of algorithm using a combined de-randomization method is an approximation method developed by [AFK96] for assignment problems for dense graphs.

In an *assignment problem*, there are n tasks that have to be assigned to n people (so that each person is assigned to exactly one task), possibly under some additional constraint, and some objective function has to be minimized or maximized.

A feasible solution for an assignment problem can be seen as a permutation of the elements of $\{1, \ldots, n\}$, or as a *perfect matching* in a complete graph with n nodes. A mathematical programming formulation of an assignment problem usually involves n^2 variables $x_{i,j}$ and the *assignment constraints*.

$$\begin{array}{ll} \sum_i x_{i,j} = 1 & \forall j \in \{1,\dots,n\} \\ \sum_j x_{i,j} = 1 & \forall i \in \{1,\dots,n\} \\ x_i \in \{0,1\} & \forall i \in \{1,\dots,n\} \end{array} \tag{6}$$

A feasible solution $\mathbf{x}$ to the following set of constraints:

$$\begin{array}{ll} \sum_i x_{i,j} = 1 & \forall j \in \{1,\dots,n\} \\ \sum_j x_{i,j} = 1 & \forall i \in \{1,\dots,n\} \\ x_i \in \{0,1\} & \forall i \in \{1,\dots,n\} \end{array} \tag{7}$$

is usually called a *matching.* A fractional solution $\mathbf{x}$ with $0 \leq x_i \leq 1$ that satisfies the first two sets of constraints of 6 is called a *fractional perfect matching.* If we want to minimize a linear objective function of $\mathbf{x}$ with non-negative coefficients under the constraints 6, then we have the min-cost perfect matching problem, that can be solved in polynomial time. With a non-linear objective function and/or with additional constraints the problem becomes NP-hard, and approximation algorithms are of interest, even for special cases.

An approach to solve assignment problems could be to consider the relaxation over fractional matchings and then use randomized rounding to obtain a 0/1 solution. Unfortunately, after randomized rounding is performed, the obtained solution can be very far from a matching, and it does not seem possible to "patch" it in a reasonable way in order to obtain a matching of comparable cost.

We will now see a non-trivial method of rounding a fractional matching, introduced by Arora, Frieze and Kaplan. The fundamental result is the following.

Theorem 4.1 *There exists a randomized polynomial-time algorithm that, given a fractional perfect matching* $\mathbf{x}$*, and a vector* $\mathbf{a}$*, returns with high probability a matching* $\mathbf{y} \in \{0,1\}^{n^2}$ *such that*

$$\begin{array}{ll} \sum_{i,j} a_{i,j} x_{i,j} - a_{\max} O(\sqrt{n} \text{poly} \log(n)) & \leq \sum_{i,j} a_{i,j} x_{i,j} \\ & \leq \sum_{i,j} a_{i,j} x_{i,j} + a_{\max} O(\sqrt{n} \text{poly} \log(n)) \\ \sum_{i,j} y_{i,j} \geq n - o(n) & \end{array}$$

where $a_{\max} = \max_i |a_i|$.

When combined with the approximation algorithms of the previous section, the rounding procedure for the assignment problem yields the following result.

Theorem 4.2 *There exists a randomized algorithm that, given the polynomial integer problem*

$$\begin{array}{lll} \max & p(x_1,\ldots,x_n) & \\ \text{Subject to} & & \\ \sum_i x_{i,j} = 1 & & \forall j \in \{1,\ldots,n\} \\ \sum_j x_{i,j} = 1 & & \forall i \in \{1,\ldots,n\} \\ x_i \in \{0,1\} & & \forall i \in \{1,\ldots,n\} \end{array} \tag{8}$$

where p is a c-smooth degree-d, polynomial, and a real $\epsilon > 0$, finds a matching $\mathbf{y}$ such that:

1. $p(\mathbf{y}) \geq p(\mathbf{x}^*) - \epsilon n^d$ *where $\mathbf{x}^*$ is the optimum solution of (8);*
2. $\sum_{i,j} y_{i,j} \geq n - o(n)$.

The algorithm runs in time $n^{O(\mathrm{poly}(c,d,\frac{1}{\epsilon})\log n)}$.

We will only review the proof and the de-randomization of Theorem 4.1. The rounding has three phases: *oversampling*, *decomposition*, and *merging*. Randomness is used in the first and the third phase. Both such phases can be de-randomized using Theorem 2.3. In the following outline of the proof of Theorem 3.4 we denote by $\mathbf{x}$ the initial fractional perfect matching. We can also identify matchings (i.e. feasible solutions for (7)) as matchings in the complete bipartite graph $K_{n,n}$.

Oversampling

In the oversampling phase, n^2 integer variables $X_{i,j}$ are defined.

Properties: With high probability,

$$\begin{array}{l} \forall i \;\; L - \sqrt{L}\,\mathrm{poly}\log L \leq \sum_j X_{i,j} \leq L + \sqrt{L}\,\mathrm{poly}\log L \\ \forall j \;\; L - \sqrt{L}\,\mathrm{poly}\log L \leq \sum_i X_{i,j} \leq L + \sqrt{L}\,\mathrm{poly}\log L \\ \sum_{i,j} a_{i,j} X_{i,j} \geq L \sum_{i,j} x_i a_{i,j} - a_{\max}\sqrt{n}\,\mathrm{poly}\log n \end{array} \tag{9}$$

Procedure: Fix a value $L = \Theta(\log^2 n)$.

For $i = 1,\ldots,n$, $j = 1,\ldots,n$, make the following random experiment: toss L times a coin biased so that heads comes up with probability $x_{i,j}$ and tail with probability $1 - x_{i,j}$. Let $X_{i,j}$ be the number of heads.

In order to deterministically find values $X_{i,j}$ satisfying properties (9), we apply Theorem 2.3 to the vector $\mathbf{X}' = L\mathbf{x}$ that satisfies

$$\begin{array}{l} \forall j \ \sum_i X'_{i,j} = L \\ \forall i \ \sum_j X'_{i,j} = L \\ \sum_{i,j} X'_{i,j} a_{i,j} = L \sum_{i,j} x_i a_{i,j} \end{array}$$

The algorithm of Theorem 2.3 will find a vector $X_{i,j}$ that satisfies (9).

Decomposition

It is convenient to see $X_{i,j}$ as a bipartite multigraph with two components of size n. $X_{i,j}$ is the number of edges between the i-th left node and the j-th right node. Equations (9) can be restated as saying that any node of this graph has degree close to L. The graph can be made D-regular (for some $D = O(\log^2 n)$) by adding at most $\sqrt{L}\text{poly}\log L$ edges to any node. Overall, $O(n\sqrt{L}\text{poly}\log L)$ edges are added. Any D-regular bipartite multigraph can be decomposed into D perfect matchings, and this is what we do. After the decomposition, we ignore the edges that were added to make the graph regular. Let $M_1, \ldots M_d$ be the resulting matchings. Here we use the word matching in a graph-theoretic sense, i.e. a subset of edges of $K_{n,n}$ without common endpoints. To each such matching we can associate in a natural way a vector of $\{0,1\}^{n\times n}$ satisfying equations (7): it is just its characteristic vector. If M is a matching in $K_{n,n}$ and $\mathbf{y}$ is its vector representation, we write $a(M) = \sum_{i,j} a_{i,j} y_{i,j} = \sum_{(i,j)\in M} a_{i,j}$ its cost. From the properties of $X_{i,j}$, we have that

$$\mathbf{a}^T \cdot \mathbf{x} - \sqrt{n}\text{poly}\log(n) \le \frac{1}{D}\sum_i a(M_i) \le \mathbf{a}^T \cdot \mathbf{x} + \sqrt{n}\text{poly}\log(n)$$

4.1 Merging

Property: Given two matchings M and M' it finds a matching M'' that with high probability satisfies

$$\frac{a(M)+a(M')}{2} - n^{3/4}\text{poly}\log n \le a(M'') \le \frac{a(M)+a(M')}{2} + n^{3/4}\text{poly}\log n$$

$M \cup M'$ is a graph of maximum degree 2. Its connected components are paths and cycles. By possibly deleting $O(\sqrt{n})$ edges we make sure each connected component has size at most $O(\sqrt{n})$. We partition the connected components into $O(\sqrt{n})$ groups each of size $O(\sqrt{n})$. For each group independently we do the following: with probability $1/2$ we decide to put in M'' the edges of the i-th group belonging to M; with probability $1/2$ we put in M'' the edges of the i-th group belonging to M'.

The proof of correctness uses a variation of Chernoff bounds and is relatively simple. For the de-randomization, let g be the number of groups. For any $i = 1, \ldots, g$ let C_i (resp. C_i') be the total cost of the edges of M (resp. of M') in the i-th group. We introduce variables $z_1, \ldots, z_g$ with the intended meaning that if $z_i = 1$ (resp. $z_i = 0$) then we take the elements of M (resp. of M') out of the i-th group. With this interpretation, the cost of a matching M'' produced according to the variables z_i is

$$\sum_i z_i C_i + (1 - z_i) C_i'$$

Under the fractional assignment $z_i = 1/2$, the expression above evaluates to $(a(M) + a(M'))/2$. We can thus invoke Theorem 2.3.

Attributions and Related Work. *All the results of this section are due to Arora, Frieze and Kaplan [AFK96].*

5 Bounded Independence

In this section we consider a case where randomization is used to construct a combinatorial object, and de-randomization involves an alternative explicit construction of such an object. In this particular instance, the explicit construction makes use of small sample spaces with *bounded independence*, that constitute, together with conditional probabilities and random walks, a major de-randomization tool.

We consider the Sparsest Cut and the Maximum Concurrent Flow problems. In such problem, the instance is an undirected graph $G = (V, E)$, a set of k pairs (s_i, t_i), k *demands* d_i. and *capacities* $c(e) > 0$ for each edge $e \in E$.

In the Maximum Concurrent Flow problem, we are asked to find the largest α such there is an assignment of flow values $f(P)$ to paths P between the pairs (s_i, t_i), without exceeding the capacity of each edge, such that the flow from s_i to t_i is at least αf_i $1 \leq i \leq k$.

The problem can be formulated as a linear program. To this aim we introduce the following notation: for any $i = 1, \ldots, k$ we denote by P_i the set of paths in G from s_i to t_i. The formulation is as follows.

$$
\begin{array}{lll}
\max & \alpha & \\
\text{Subject to} & & \\
& \sum_{P \in \mathsf{P}_i} f(P) \geq \alpha d_i & \forall i \in \{1, \ldots, k\} \\
& \sum_{i=1}^{k} \sum_{P \in \mathsf{P}_i} f(P) \leq c(e) & \forall e \in E \\
& f(P) \geq 0 & \forall P \in \mathsf{P}\ \forall i \in \{1, \ldots, k\}
\end{array}
\tag{10}
$$

Even if the above problem has an exponential number of variables, it can be shown that it can be reformulated in an equivalent way in polynomial size, and thus can be optimally solved in polynomial time. Even without changing the formulation, it can be solved in polynomial time with the ellipsoid method, since it is a convex optimization problem for which a polynomial-time *separation oracle* exists.

In the Sparsest Cut problem we want to find a partition $(S, V - S)$ of the vertices that induces a "small" cut and such that "most" pairs (s_i, t_i) are separated. For a set $S \subseteq V$, we define $\mathsf{cut}(S) = \{(u, v) \in E : u \in S \wedge v \notin S\}$ and we let $\mathsf{sep}(S)$ be the set of indices i such that s_i and t_i are in opposite sides of the partition $(S, V - S)$ (formally, $\mathsf{sep}(S) = \{i : |S \cap \{s_i, t_i\}| = 1\}$). Then, we want to find a set S with a large value of $\sum_{i \in \mathsf{sep}(S)} d_i$ and a small value of $\sum_{e \in \mathsf{cut}(S)} c(e)$. We will actually minimize the ration of the two values. The problem is thus as follows

$$
\begin{array}{ll}
\min & \frac{\sum_{e \in \mathsf{cut}(S)} c(e)}{\sum_{i \in \mathsf{sep}(S)} d_i} \\
\text{Subject to} & \\
& S \subseteq V
\end{array}
$$

When $k = 1$, the Sparsest Cut problem reduces to the standard Min CUT problem, and is solvable in polynomial time. For larger values of k, it is a standard NP-complete problem. With simple manipulation, we can restate in the following equivalent way, where we have a variable x_e for any edge and a variable y_i for any pair (s_i, t_i).

$$
\begin{array}{lll}
\min & \frac{\sum_{e \in E} c(e) x_e}{\sum_{i=1}^{k} d_i y_i} & \\
\text{Subject to} & & \\
& \sum_{e \in P} x_e \geq y_i & \forall P \in \mathsf{P}_i \ , \ \forall i = 1, \ldots, k \\
& x_e \in \{0, 1\} & \forall e \in E \\
& y_i \in \{0, 1\} & \forall i = 1, \ldots, k
\end{array}
$$

If we consider the relaxation where the variables x_i and y_i range over the non-negative reals, we can normalize the objective function by adding the constraint $\sum_i d_i y_i = 1$. We obtain a linear programming relaxation

$$
\begin{array}{lll}
\min & \frac{\sum_{e \in E} c(e) x_e}{\sum_{i=1}^{k} d_i y_i} & \\
\text{Subject to} & & \\
& \sum_{e \in P} x_e \geq y_i & \forall P \in \mathsf{P}_i \ , \ \forall i = 1, \ldots, k \\
& x_i \geq 0 & \forall i = 1, \ldots, k \\
& y_i \geq 0 & \forall e \in E\}
\end{array}
\tag{11}
$$

Which is the dual of the formulation (10) of the Concurrent Flow problem. In particular the optima of the relaxed Sparsest Cut problem and of the Concurrent Flow problem are the same.

An $O(\log k)$-approximate algorithm for the Sparsest Cut problem can be obtained by rounding an optimum solution of the relaxation. The rounding procedure increases the cost of the rounded solution by at most a factor $O(\log k)$.

The rounding scheme is based on two results: that a certain embedding of the graph in a ℓ_1 metric can be converted into a feasible solution for the Sparsest Cut problem, and that any solution to the relaxation (11) yields an embedding of the graph into a ℓ_1 metric. Before presenting such results, we need some preliminary definitions.

Definition 5.1 *For any integer n and real $p > 1$, the ℓ_p norm of a vector $\mathbf{a} \in \mathbf{R}^n$ is defined as*

$$
||\mathbf{a}||_p = \left(\sum_{i=1}^{n} |a_i|^p \right)^{1/p} .
$$

The ℓ_p norm induces a metric space in $\mathbf{R}^n$ under the distance function $d_p(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}||_p$. We call ℓ_p^n the metric space $(\mathbf{R}^n, d_p)$. Observe that d_2 is the standard *Euclidean distance*, while d_1 has been called *rectilinear distance*.

Given two metric spaces (X, d) and (Y, d'), and *embedding* of the former in the latter is simply a function $\phi : X \to Y$. An *isometry* is an embedding $\phi : X \to Y$ such that, for any $\mathbf{a}, \mathbf{b} \in X$, $d(\mathbf{a}, \mathbf{b}) = d(\phi(\mathbf{a}), \phi(\mathbf{b}))$. An embedding has *distorsion* $c \geq 1$ if, for any $\mathbf{a}, \mathbf{b} \in X$, $d(\phi(\mathbf{a}), \phi(\mathbf{b})) \leq d(\mathbf{a}, \mathbf{b}) \leq cd(\phi(\mathbf{a}), \phi(\mathbf{b}))$. In particular, an isometry is an embedding with distorsion 1.

A feasible solution $(\mathbf{x}, \mathbf{y})$ for (11) can always be seen as defining a metric space $(V, d_{\mathbf{x}})$; the solution assigns "weights" x_e to any edge, and the distance $d_{\mathbf{x}}(u, v)$ between two vertices u and v can be defined as the shortest path distance in the resulting weighted graph.

Theorem 5.2 *Given a feasible solution* $\mathbf{x}, \mathbf{y}$ *of (11), and an embedding* ϕ *of* $(V, d_{\mathbf{x}})$ *into an* ℓ_1 *space such that the following holds:*

1. *For all* $u, v \in V$, $d_{\mathbf{x}}(u, v) \geq ||\phi(u) - \phi(v))||_1$;
2. *For all* $i = 1, \ldots, k$, $d_{\mathbf{x}}(s_i, t_i) \leq c||\phi(s_i) - \phi(t_i)||_1$.

it is possible to find in polynomial time a feasible solution for the Sparsest Cut problem of cost at most c *times the cost of* $\mathbf{x}, \mathbf{y}$.

Note that the embedding is required to have distorsion c on the points s_i, t_i, and is only required not to increase the other distances. For convenience, we define a generalization of the kind of embeddings we are interested in.

Definition 5.3 *Let* (X, d) *be a metric space and* $Y \subseteq X$. *An embedding* ϕ *of* (X, d) *into another metric space* (M, d') *is said to be* c*-good for* $((X, d), y)$ *if*

1. *For all* $\mathbf{a}, \mathbf{b} \in X$, $d(\mathbf{a}, \mathbf{b}) \geq d'(\phi(\mathbf{a}) - \phi(\mathbf{b}))$;
2. *For all* $\mathbf{a}, \mathbf{b} \in Y$, $d(\mathbf{a}, \mathbf{b}) \leq cd'(\phi(\mathbf{a}), \phi(\mathbf{b}))1$.

Theorem 5.4 *For any metric space* (X, d) *and subset* $Y \subseteq X$, *there exists a* $O(log|Y|)$*-good embedding for* $((X, d), y)$ *in a* ℓ_1 *space. Such an embedding can be found in random polynomial time.*

¿From the fact the (11) can be optimally solved in polynomial time and from the above two theorems it follows that the Sparsest Cut problem admits a $O(\log k)$-approximate algorithm.

Randomization is only used in the proof of Theorem 5.4. We will see how to obtain a deterministic version.

Theorem 5.5 *For any metric space (X, d) and subset $Y \subseteq X$, there exists a $O(log|Y|)$-good embedding in a ℓ_2 space. Such an embedding can be found in random polynomial time.*

The existence proof uses the probabilistic method. Once the existence is established, an actual embedding can be found as a standard application of Semidefinite Programming. It is worth stressing that semidefinite programming only works for ℓ_2 metrics.

Theorem 5.6 *Any set X of points in a ℓ_2 space can be embedded into an ℓ_1 space with constant distorsion. The embedding is computable in polynomial time.*

This theorem is the hearth of the de-randomization.
PROOF: [Rough Sketch] Let m be the dimension of the space containing V. Let $S \subseteq \{-1, 1\}^m$ be a collection of 4-wise independent vectors, $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_{|S|}\}$. It is possible to construct such a collection of size $O(m^2)$ in poly(m) time. The mapping associates to an element $\mathbf{a} \in V$ the element $\phi(\mathbf{a}) \in \mathbf{R}^{(|S|)}$ defined as follows

$$\phi(\mathbf{a}) = \left(\frac{1}{|S|}\mathbf{a} \cdot \mathbf{s}_1, \ldots, \frac{1}{|S|}\mathbf{a} \cdot \mathbf{s}_m\right)$$

Such a mapping introduces a distorsion at most $\sqrt{3}$. □

Combining the proofs of Theorems 5.5 and 5.6 it is possible to get the final result.

Theorem 5.7 *For any metric space (X, d) and subset $Y \subseteq X$, there exists a $O(log|Y|)$-good embedding in a ℓ_1 space. Such an embedding can be found in polynomial time.*

The *deterministic* $O(\log k)$-approximate algorithm for the Sparsest Cut problem works as follows:

1. an optimum solution $(\mathbf{x}^*, \mathbf{y}^*)$ for (11) is found. The cost of such solution is lower bound to the optimum of the Sparsest Cut problem;

2. Using Theorem 5.7 we find an $O(\log k)$-good embedding for $((V, d_{\mathbf{x}^*}), \{s_i, t_i : i = 1, \ldots, k\}$;

3. Using Theorem 5.2 we find a feasible solution for the Sparsest Cut problem of cost at most $O(\log k)$ times the cost of $(\mathbf{x}^*, \mathbf{y}^*)$. Such a solution is $O(\log k)$-approximate.

ATTRIBUTIONS AND RELATED WORK. *There has been a lot of research on the approximability of the Sparsest Cut problem and the related Multi Cut problem. Shmoys' survey [Shm96] presents a very complete account on the known results and their history. Regarding the results mentioned in this section, the relation between embeddings in ℓ_1 metrics and the Sparsest Cut problems was first noted in [AD91]. Theorem 5.2 was independently discovered by Linial, London and Rabinovich [LLR95] and by Aumann and Rabani[AR]. The randomized embedding in ℓ_1 metrics appeared in a preliminary version [LLR94] of the paper of Linial, London and Rabinovich. The deterministic embedding was found later, by the same authors, and appears in [LLR95]. The deterministic mapping of ℓ_2 metrics into ℓ_1 metrics using 4-wise independence (Theorem 5.6) is attributed to [Ber97] in [LLR95]. A deterministic $O(\log k)$-approximate algorithm for Sparsest Cut was also discovered by Garg [Gar95].*

Acknowledgements

We are grateful to Sanjeev Arora and Oded Goldreich for prompt answers to some questions of ours.

References

[AD91] D. Avis and M. Deza. The cut cone, l^1 embeddability, complexity, and multicommodity flows. *Networks*, 21:595–617, 1991.

[AFK96] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 21–30, 1996.

[AKK95] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Pro-*

ceedings of the 27th ACM Symposium on Theory of Computing, pages 284–293, 1995.

[AR] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*. To appear.

[Aro97] S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. Manuscript, 1997.

[AS92] N. Alon and J. Spencer. *The Probabilistic Method*. Wiley Interscience, 1992.

[Ber97] B. Berger. The fourth moment method. *SIAM Journal on Computing*, 26(4):1188–1207, 1997.

[BR94] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 276–287, 1994.

[dlV] W.F. de la Vega. MAXCUT has a randomized approximation scheme in dense graphs. *Random Structures and Algorithms*. To appear.

[FK96] A. Frieze and R. Kannan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, 1996.

[Gar95] N. Garg. A deterministic $O(\log k)$-approximate algorithm for the sparsest cut problem. Manuscript cited in [Shm96], 1995.

[GGR96] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection with learning and approximation. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, 1996.

[Gil93] D. Gillman. A chernoff bound for random walks on expander graphs. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 680–691, 1993.

[GW94] M. Goemans and D. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994. Preliminary version in *Proc. of IPCO'93*.

[LLR94] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 577–591, 1994.

[LLR95] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[Rag86] P. Raghavan. *Randomized Rounding and Discrete Ham-Sandwich Theorems: Provably Good Algorithms for Routing and Packing Problems*. PhD thesis, University of California at Berkeley, 1986.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37:130–143, 1988.

[RT87] P. Raghavan and C.D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[Shm96] D. Shmoys. Cut problems and their applications to divide-and-conquer. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 192–235. PWS Publishing, 1996.

[Sri95] A. Srinivasan. Improved approximations of packing and covering problems. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 268–276, 1995.

[SS96] A. Srivastav and P. Stangier. Algorithmic chernoff-hoeffding inequalities in integer programming. *Random Structures and Algorithms*, 1996. Preliminary version in *Proc. of ISAAC'94*.

[Tre96] L. Trevisan. Positive linear programming, parallel approximation, and PCP's. In *Proceedings of the 4th European Symposium on Algorithms*, pages 62–75. LNCS 1136, Springer-Verlag, 1996.