New Developments in Iterated Rounding*

Nikhil Bansal

Eindhoven University of Technology
Eindhoven, The Netherlands
n.bansal@tue.nl

Abstract

Iterated rounding is a relatively recent technique in algorithm design, that despite its simplicity has led to several remarkable new results and also simpler proofs of many previous results. We will briefly survey some applications of the method, including some recent developments and giving a high level overview of the ideas.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Algorithms, Approximation, Rounding

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.1

Category Invited Talk

1 Introduction

A natural and very general approach to designing approximation algorithms for NP-Hard problems is the following: Write an exact integer programming formulation for the problem, and then relax the integer constraints to be continuous, giving rise to a convex optimization problem such as a linear program or a semidefinite program that can be solved efficient in polynomial time. The goal then is to design a good *rounding* procedure that converts this fractional solution back to an integral solution without much loss in the value of the objective.

In the last few decades various ingenious rounding techniques have been developed, with several surprising connections to probability, geometry and so on. For an overview of these methods and approximation algorithms in general, we refer the reader to [10]. In recent years, a powerful new approach for rounding, referred to as iterated rounding has emerged. Here the variables are rounded one by one, or in small steps over time, while crucially leveraging the information gained from previous steps. While this idea is not new by itself, many interesting and surprising applications have emerged recently. An excellent description of iterated rounding and its applications can be found in [5].

Here we give a brief introduction to the method. We start with some classic results to demonstrate its versatility and power, and then discuss some more recent developments and incarnations based on techniques such as discrepancy and Lovász Local Lemma.

2 The Basic Approach

Consider a linear programming relaxation of the form

$$\min c^T x \qquad \text{s.t.} \quad Ax \leq c \qquad \text{and} \quad x \in [0,1]^n,$$

* Supported by NWO grant 639.022.211 and ERC consolidator grant 617951.

34th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014).
Editors: Venkatesh Raman and S. P. Suresh; pp. 1–10

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with variables $x_1, \ldots, x_n$. Here x denotes the vector $(x_1, \ldots, x_n)$ and A is some $m \times n$ matrix, and c is a non-negative cost vector in R^n.

We will refer to the m constraints given by rows of $Ax \leq c$ as the non-trivial constraints, and the remaining constraints given by $x \in [0,1]^n$ as the trivial constraints.

Recall that for any linear program, there is always some optimum solution that lies at the vertex of the polytope formed by the constraints. Such a solution is referred to as a basic feasible solution. The key idea behind much of iterated rounding is the following easy observation.

▶ **Lemma 1.** *For any linear program of the form above with $m < n$ non-trivial constraints, there is an optimum solution with at least $n - m$ variables set to 0 or 1.*

Proof. Given a solution x, We say that a constraint is tight at x if it is satisfied by equality by at x. As the polytope in n-dimensional, every vertex x of the polytope is determined uniquely by some n linear independent constraints that are tight at x.

Consider some basic feasible optimal solution. As there are m non-trivial constraints, at least $n - m$ of the tight constraints at x must be trivial ones. If a trivial constraint involving x_i is tight, this means that $x_i = 0$ or $x_i = 1$. ◀

The algorithm proceeds as follows:

1. Start with the initial LP, and compute some basic feasible solution x^*.
2. Permanently fix the value of any variable that is set to 0 or 1, and then consider the residual LP (obtained by drop this fixed variable and updating the right hand of each constraint accordingly).
3. Find a linearly independent set of tight constraints at x^*, and choose one (or more) of these constraints in some suitable problem specific manner (this is where the ingenuity lies) and drop it from the residual LP. Recompute a basic feasible solution of this reduced LP and iterate the process until an integer solution is obtained.

The key observation is that dropping one of the constraints that determines x^* allows the LP to get *unstuck* at x^* and move to some another vertex solution. Note that since we drop at least one constraint at each iteration, the procedure will eventually terminate. Moreover, it is easy to verify that objective value of the LP can only go down during the various iterations of the algorithm (as permanently fixing a variable that is already 0 or 1 does not affect the objective value, and dropping a constraint can only reduce the objective).

We begin with a simple example.

2.1 Makespan Minimization on Unrelated Machines

The unrelated machine setting is the following. There are m machines and n jobs. Each job j must be processed on some machine, and it has arbitrary machine dependent processing time p_{ij} on machine i. The goal is to assign jobs to the machines to minimize the makespan (or the maximum load over all machines). In a classic result, Lenstra, Shmoys and Tardos [6] gave a 2-approximation for the problem, and also show that no $1.5 - \epsilon$ approximation exists unless P=NP. It is a major open question in approximation algorithms to improve the approximation ratio of 2. Here we give a simple iterated rounding based proof of their result, as described in [5].

2.1.1 Algorithm

By doing a binary search we can assume that we know the value T of the optimum makespan. We will write a LP with variables x_{ij} with the intended solution that $x_{ij} = 1$ if j is assigned to machine i and 0 otherwise.

We do an initial preprocessing step where we set $x_{ij} = 0$ if $p_{ij} > T$ (as j can never be assigned to i in a solution with makespan T). Let us assume that we are given a feasible solution to the following LP.

$$\sum_j p_{ij} x_{ij} \leq T \qquad \forall i \in [m] \tag{1}$$

$$\sum_i x_{ij} = 1 \qquad \forall j \in [n] \tag{2}$$

$$x_{ij} \geq 0 \qquad \forall i, j \tag{3}$$

Note that while the number of variables can be nm, the number of non-trivial constraints (1) and (2) is $n + m$. Also note that we do not impose the constraint $x_{ij} \leq 1$ as this is implied by (2).

The algorithm proceeds via the iterated rounding framework. Recall that all we need to do is to design a procedure that given a basic feasible solution as input, determines which constraint to drop in the current round.

Consider some basic feasible solution to the LP in the current round. We fix the variables that are already 0 or 1, and consider the residual solution on variables x_{ij} with $0 < x_{ij} < 1$. We say that j appears on i if $x_{ij} > 0$. We will show the following.

▶ **Lemma 2.** *There exists a machine i such that (i) exactly one job j appears on i, or (ii) exactly two jobs j and j' appear on i and satisfy $x_{ij} + x_{ij'} \geq 1$.*

Given Lemma 2, the dropping rule will be to drop the load constraint (1) corresponding to this machine i. Lemma 2 ensures that no matter how the variables are rounded in subsequent iterations the additional load assigned to machine i can be at most $p_{\max}$. Indeed, in case (i) either job j can be assigned to i, which can increase the load by at most $(1 - x_{ij})p_{ij} \leq p_{ij}$, and in case (ii) both j and j' can be assigned to i, and the load can increase by at most $p_{ij}(1 - x_{ij}) + p_{ij'}(1 - x_{ij'}) \leq p_{\max}(2 - x_{ij} - x_{ij'}) \leq p_{\max}$.

It remains to show Lemma 2. This is done by a counting argument (which is typical in most iterated rounding proofs).

Proof. Let p (resp. f) denote the number of variables x_{ij} with $0 < x_{ij}$ (resp. $0 < x_{ij} < 1$). As the number of non-trivial constraints is at most $n + m$ (and as the trivial constraints are of the form $x_{ij} \geq 0$), there is a basic feasible solution with $p \leq n + m$. Note that each job contributes 1 to p if it is assigned integrally to some machine, and least 2 if it is assigned fractionally to two or more machines. So we obtain that

$$n \leq (p - f) + (f/2) = p - f/2 \leq (n + m) - f/2. \tag{4}$$

The first inequality follows as $p - f$ is the number of variables with $x_{ij} = 1$ and the last inequality follows as $p \leq n + m$. This gives that $f \leq 2m$. If $f \leq 2m - 1$, then we are already done as there must exist some machine i with at most one fractional variable appearing on it.

On the other hand, if exactly two fractional variables appear on each machine, then $f = 2m$, which implies that equality must hold throughout in (4), and hence $p - f = n - m$. This implies that exactly m jobs are split fractionally, and each of them appears on exactly two machines. Thus, there must be some machine i with two jobs j and j' with $x_{ij} + x_{ij'} \geq 1$, as claimed. ◀

2.2 Degree-bounded Spanning Trees

Our next example is one where on first glance it would seem that iterated rounding should not work, as the number of non-trivial constraints m is substantially larger than the number of variables n, and the conditions of Lemma 1 do not seem to apply.

The minimum cost degree bounded spanning tree problem is defined as follows. Given a graph $G = (V, E)$ with non-negative edge costs c_e, and degree bounds b_v on the vertices, find a minimum cost spanning tree of G satisfying the degree bounds. Even though the minimum spanning tree problem is efficiently solvable, adding the degree bounds makes the problem NP-complete. In particular, if $b_v = 2$ for all v, the problem reduces to Hamiltonian Path.

We will show the following result of Singh and Lau [9] which gives essentially the best possible guarantee.

▶ **Theorem 3.** *There is an efficient algorithm that finds a spanning tree with degree bounds violated by at most +1, and with cost at most the cost of optimum spanning tree (that satisfies the degree bounds exactly).*

2.2.1 Algorithm

The starting point is the following natural LP relaxation for the problem with variables x_e that are supposed to indicate whether edge e is chosen or not.

$$\min c_e x_e \tag{5}$$

$$\text{s.t.} \sum_{e \in E[S]} x_e \leq |S| - 1 \qquad \forall S \subset V \tag{6}$$

$$\sum_{e} x_e = n - 1 \tag{7}$$

$$\sum_{e \in \delta(v)} x_e \leq b_v \qquad \forall v \in v \tag{8}$$

$$0 \leq x_e \leq 1 \qquad \forall e \in E \tag{9}$$

Here $E[S]$ is the set of edges in the subgraph induced by $S \subset V$, and $\delta(v)$ is the set of edges incident to v. Recall that the constraints (6), (7) and (9) completely characterize the spanning tree polytope.

Observe that the number of variables is $O(n^2)$ (one for each edge) while the number of constraints (6) are exponentially many. Still it turns out that the iterated framework can be applied very effectively.

The crucial observation is that even though exponentially many constraints (6) may be tight at a given vertex of the polytope, all these tight constraints are spanned by a small set of at most $n - 1$ linearly independently constraints. More precisely, the supermodularity of the function $\chi_{E[S]}$ implies that the tight constraints given by (6) can be uncrossed, and hence spanned by constraints corresponding to sets forming a laminar family. Thus the fractional part of the basic feasible solution is completely determined by these (at most) $n - 1$ constraints together with some other tight degree constraints. As the number of degree constraints can be at most n, essentially the number of relevant constraints (and hence the number of fractional variables) is at most $(n - 1) + n$. We remark later in later iterations, a slightly more careful argument is needed.

A counting argument now allows one to show that there is some vertex v such that $\sum_{e \in \delta(v): x_e > 0} (1 - x_e) < 2$. The details are quite simple, and we refer the reader to [5] for details.

This implies that if we drop the degree constraint on vertex v, then even if all the variables are rounded to 1 is subsequent iterations, the degree violation can be at most strictly less than 2. This specifies the dropping rule for the algorithm. The algorithm keeps iterating until it has found an integral solution, or until all degree bounds have been dropped in which case the LP reduces to the spanning tree LP which is integral. Finally, we observe that since the degrees and integral, a violation of strictly less than 2 implies a violation of at most 1.

2.3 Bin Packing

Our next example gives an application of iterated rounding where we drop a constant fraction of constraints at each iteration, and the algorithm terminates in a logarithmic number of rounds.

The classical bin packing problem is the following. Given a collection of items with sizes $s_1, \ldots, s_n$ where each $0 < s_i \leq 1$, pack these items feasibly into the fewest number of unit size bins. The problem is NP-Complete, as the Partition problem implies that it is hard to distinguish if the optimum packing requires 2 bins or 3 bins. In fact, this is the best known hardness for the problem, and it is a major open question to determine whether there exists a polynomial time algorithm achieving an $\mathrm{Opt} + O(1)$ or even $\mathrm{Opt} + 1$ guarantee.

In 1981, Karmarkar and Karp [4] gave a remarkable algorithm that achieves a guarantee of $\mathrm{Opt} + O(\log^2 \mathrm{Opt})$. This result is one of the first applications of iterated rounding.

2.3.1 Algorithm

The starting point is a very strong relaxation known as the configuration LP. Suppose we have an instance that contains n_i items of size s_i. Let k denote the number of distinct sizes. By simple arguments, we can ignore items of size $\leq 1/\mathrm{Opt}$ (as these items are easy to fill in later). A valid configuration C is any multiset of sizes in the collection $\{s_1, \ldots, s_k\}$ with total size at most 1. Let $\mathcal{C}$ denote the collection, possibly exponentially large, of all valid configurations.

Consider the following LP formulation with a variable x_C for each configuration $C \in \mathcal{C}$, that is supposed to indicate the number of bins packed using configuration C.

$$\min \sum_C x_C \qquad \text{s.t.} \quad \sum_C a_{i,C} x_C \geq n_i \quad \forall i = 1, \ldots, k, \quad \text{and} \quad x_C \geq 0 \qquad \forall C \in \mathcal{C}.$$

Here $a_{i,C}$ denotes the number of items of size s_i in C. Even though the number of variables is exponential, this LP can be solved to any desired accuracy by considering the dual (and moreover a basic feasible solution can be computed using standard techniques). As this LP has only $k \leq n$ constraints, at most k configurations C have non-zero x_C values.

The crucial observation of [4] was the following.

▶ **Lemma 4.** *Given a bin packing instance I with total size of items $s(I)$, there is a procedure to round up the size of each item sizes to obtain another instance $\tilde{I}$ with at most $s(I)/2$ distinct item sizes, and $Opt(\tilde{I}) \leq Opt(I) + O(\log s(I))$.*

The proof of Lemma 4 is quite simple and we refer the reader to [5] for a proof.

Let us see how this gives the claimed algorithm. Consider an instance with k item sizes. The algorithm solves the configuration LP to obtain a solution with at most k non-zero configurations. For each C with $x_C > 0$, pack $\lfloor x_C \rfloor$ bins with configuration C and remove these items. Consider the instance consisting of the remaining unpacked items. As these fit fractionally in at most k configurations, the total size of these items is at most k, and by

Lemma 4 we can round these to $k/2$ sizes, while losing at most $O(\log k)$ in the objective. We now iterate the algorithm on this rounded instance.

Observe that at each iteration the number of distinct sizes decreases by at least half, and thus there are logarithmically many iterations, each adding at most log(Opt) to the objective. This implies the claimed result.

2.4 Flow Time Minimization

Usually when designing a LP based approximation, one tries to add as many valid constraints as possible to make the relaxation tighter. However, sometimes it is beneficial to reduce the number of constraints so that iterated rounding can be used. We next give an example of a problem for which strong LP relaxations are known but they key is to consider a different (weaker) formulation with much fewer constraints.

The problem is that of minimizing the total flow time on unrelated machines. Given a collection on n jobs and m machines, where job j has size p_{ij} on machine i and release time r_j, find a feasible schedule to minimize the total flow time. Here the flow time of a job is the amount of time it spends in the system; i. e., its completion time minus its arrival time. We assume that the schedule is non-migratory and preemptive, i. e., a job can only be executed on a single machine and can be preempted and resumed later without any penalty.

Recently, Bansal and Kulkarni [2] gave the first poly-logarithmic approximation for the problem based on iterated rounding.

2.4.1 Standard LP formulation

We first describe the widely used standard time indexed LP relaxation for our problem. There is a variable x_{ijt} for each machine $i \in [m]$, each job $j \in [n]$ and each unit time slot $t \geq r_j$. The x_{ijt} variables indicate the amount to which a job j is processed on machine i during the time slot t. The first set of constraints (10) says that every job must be completely processed. The second set of constraints (11) says that a machine cannot process more than one unit of job during any time slot. The objective function is referred to as the fractional flow time and will be irrelevant to our discussion here.

$$\min \sum_{i,j,t} \left(\frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot x_{ijt}$$

$$\text{s. t.} \quad \sum_{i} \sum_{t \geq r_j} \frac{x_{ijt}}{p_{ij}} \geq 1 \qquad \forall j \tag{10}$$

$$\sum_{j \,:\, t \geq r_j} x_{ijt} \leq 1 \qquad \forall i, t \tag{11}$$

$$x_{ijt} \geq 0 \qquad \forall i, j, t \geq 0$$

2.4.2 New LP formulation

We now describe a new LP relaxation for the problem, where we do not enforce the capacity constraints (11) for each time slot, but instead only enforce these constraints over carefully chosen intervals of time.

Let $P = \max_{i,j} p_{ij} / \min_{i,j} p_{ij}$ and assume that $\min_{i,j} p_{ij} = 1$. For $k = 0, 1, \ldots, \log P$, we say that a job j belongs to class k on machine i if $p_{ij} \in (2^{k-1}, 2^k]$. Note that the class of a job depends on the machine.

There is a variable y_{ijt} (similar to x_{ijt} before) that denotes the total units of job j processed on machine i at time t. However, unlike the time indexed relaxation, y_{ijt} is allowed to take values greater than one.

For each class k and each machine i, we partition the time horizon $[0, T]$ into intervals of size $4 \cdot 2^k$. For $a = 1, 2, \ldots$, let $I(i, a, k) = ((4 \cdot 2^k)(a-1), (4 \cdot 2^k)a]$ denote the a-th interval of class k on machine i. The new relaxation is the following.

$$\sum_i \sum_{t \geq r_j} \sum_k \sum_{j \in (2^{k-1}, 2^k]} \left(\frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot y_{ijt}$$

$$\text{s.t.} \quad \sum_i \sum_{t \geq r_j} \frac{y_{ijt}}{p_{ij}} \geq 1 \qquad \forall j \tag{12}$$

$$\sum_{j\,:\,p_{ij} \leq 2^k} \sum_{t \in I(i,a,k)} y_{ijt} \leq \text{Size}(I(i,a,k)) \qquad \forall i, k, a \tag{13}$$

$$y_{ijt} \geq 0 \qquad \forall i, j, t \,:\, t \geq r_j$$

Here, $\text{Size}(I(i, a, k))$ denotes the size of the interval $I(i, a, k)$ which is $4 \cdot 2^k$ (but would change in later iterations of the LP when we apply iterated rounding). Observe that in (13) only jobs of class $\leq k$ contribute to the left hand side of constraints corresponding to intervals of class k.

The main idea why this LP is useful is the following. For simplicity assume that all jobs belong to a single class k. Some some basic feasible solution and suppose that h constraints given by (13) are tight. As there are n constraints 12, at most $n + h$ non-trivial constraints can be tight and hence at most $n + h$ variables are non-zero. If h constraints (13) are tight, this also means that the total size of jobs is at least $h \cdot (4 \cdot 2^k)$. As these are class k-jobs, this means that $n \geq 4h$, and hence the number of non-zero variables is at most $n + h \leq 5/4n$. So a constant fraction of the jobs j must be assigned integrally to a single machine. Thus after logarithmic iterations, this produces a reasonable *pseudo-schedule*, that can be converted to a proper schedule without much additional loss. We refer the reader to [2] for details.

3 A Generalization based on Discrepancy

Recently a very powerful generalization of iterated rounding was developed, based on discrepancy theory. The examples of iterated rounding that we have seen thus far are based on Lemma 1 which requires that $m < n$, and to maintain this invariant, in each the algorithm drops certain constraints in each iteration. Observe that when a constraint $a_j x \leq b_j$ is dropped, we have no control on how much this constraint could be violated in future iterations. In particular the violation could be as large as $\|a_j\|_1$ (e. g. if all the variables x_i are very close to 0 when the constraint was dropped, and eventually they all get rounded to 1).

Recently, Lovett and Meka [7] gave the following rounding result.

▶ **Theorem 5.** *Let $x \in [0, 1]^n$ be some fractional solution to a linear system $Ax = b$, where A is an $m \times n$ matrix. For $j = 1, \ldots, m$, let λ_j be such that*

$$\sum_j \exp(-\lambda_j^2/16) \leq n/16. \tag{14}$$

Then there is an efficient algorithm to find a solution $\tilde{x}$ with the following properties:

(i) *at most $n/2$ variables fractional (that is strictly between 0 and 1),*

(ii) *$|a_j \tilde{x} - a_j x| \leq \lambda_j \sqrt{\|a_j\|_2}$ for each $j = 1, \ldots, m$, where a_j denotes the j-th row of A.*

Let us parse what theorem 5 gives us. First observe that if $m \leq n/16$, then setting each $\lambda_j = 0$ for $j = 1, \ldots, m$ satisfies (14), and gives a solution $\tilde{x}$ that does not violate any constraint and has $n/2$ variables integral.

In this setting Lemma 1 would give a solution with $n - m = (15/16)n$ variables set integrally and no constraint violated. Thus ignoring constants (which can be modified if needed by the application at hand, see for example [1]) this can be viewed as an analog of Lemma 1.

However, theorem 5 also holds when $m \gg n$ provided the error parameters λ_j are chosen to satisfy condition (14). The fact that one has complete flexibility in how to choose λ_j (provided (14) holds) can make this variant extremely powerful. For example suppose $m = 10n$. Then, standard iterated does not give anything until $m - n = 9n$ more constraints are dropped, potentially incurring $\|a_j\|_1$ error on these dropped constraints . On the other hand one can set each λ_j to be $O(1)$ in theorem 5 to obtain error $O(\|a_j\|_2)$ for each row j. The crucial point is that the ℓ_2 norm $\|a_j\|_2$ of a constraint can be substantially smaller than its ℓ_1 norm $\|a_j\|_1$ (e. g. $\sqrt{n}$ vs n), and hence theorem 5 can give much less error. Moreover, by setting $\lambda'_j s$ non-uniformly one can enforce smaller error on more critical constraints.

3.1 Improvement for Bin Packing

Recently, based on these ideas, Rothvoss [8] improved the long-standing bound of Opt + $O(\log^2(\text{Opt}))$ for bin packing that we saw in Section 2.3 to Opt + $O(\log \text{Opt} \log\log \text{Opt})$. The main observation was that if most of the configurations are "well-spread", that is, they do not consist of only few item types that appear many times, then the ℓ_2 norm of the vector corresponding to such a configuration (i. e. the vector indicating how many times each item type appears) is much smaller than its ℓ_1 norm. However, there is no apriori reason why such configurations should be used by a basic feasible solution to the LP. To get around this problem, Rothvoss introduces the crucial idea of a creating new items by grouping together various small items of the same size that appear in a configuration. These ideas together with the framework of Karmarkar Karp then give the improved bound. The details are somewhat technical and we refer the reader to [8] for details.

4 Iterated rounding via the Lovász Local Lemma

Our final example illustrates how other powerful tools such as the Lovász Local Lemma can fit nicely into the iterative approach for rounding.

The problem we consider is that of minimizing makespan on unrelated machines in the multi-dimensional setting. There are m machines and n jobs. Each machine has d resources and each job needs some quantities of these resources. For example, the machines could correspond to computers and the $d = 2$ resources could be CPU and memory. In the unrelated machines setting, the load of job j on machine i is specified by an arbitrary non-negative number p_{ijk} for each $k \in [d]$. In typical scenarios d is a fixed small constant, and n and m are much larger.

As in Section 2.1, we can guess the optimum makespan T, and write an LP with variables x_{ij}. We set $x_{ij} = 0$ if $p_{ijk} > T$ for some k, and find a feasible solution to the following.

$$\sum_j p_{ijk} x_{ij} \leq T \qquad \forall i \in [m], k \in [d] \tag{15}$$

$$\sum_i x_{ij} \geq 1 \qquad \forall j \in [n] \tag{16}$$

Let us first observe what the natural approaches give. As the LP has $n + md$ constraints, each machine could potentially have $\Omega(d)$ jobs fractionally assigned to it in a basic feasible solution. So applying the approach in Section 2.1 only gives an $O(d)$ approximation.

On the other hand if we do randomized rounding (i. e. independently assign each job j to machine i with probability x_{ij}), then by a standard balls in bins argument, the makespan could be as high as $\Theta(T \log dm / \log\log dm)$, which has an undesirable dependence on m.

It turns out that one can show the following substantially stronger result.

▶ **Theorem 6** ([3]). *There is an $O(\log d / \log\log d)$ approximation for the problem.*

Before we sketch the idea behind theorem 6, we recall the Lovász Local Lemma.

▶ **Lemma 7.** *Let $B_1, B_2, \ldots, B_k$ be a collection of (bad) events such that each B_i occurs with probability at most p and is independent of all the other events except for at most d of them. If $epd < 1$, then there is a nonzero probability that none of the events occurs.*

The idea of the algorithm is to start with an arbitrary solution x, and gradually make the variables x_{ij} *closer* to integral in each iteration, without substantially deteriorating the quality of the solution. In particular, in iteration ℓ, the values x_{ij} will be integral multiples of ϵ_ℓ, where ϵ_ℓ increases exponentially with ℓ, until $\epsilon_\ell = \Omega(\log\log d / \log d)$. At this point that algorithm can arbitrarily assign a job j to any machine i where $x_{ij} > 0$.

Let $\epsilon_0 = 1/(\log dm)$. Given an initial solution x, we can assume that each x_{ij} is an integral multiple of ϵ_0 by rounding each x_{ij} independently to either $\epsilon_0 \lfloor x_{ij}/\epsilon_0 \rfloor$ or $\epsilon_0 \lceil x_{ij}/\epsilon_0 \rceil$, with the right probability such that its expectation remains the same. By standard Chernoff bounds the makespan remains $O(T)$ with high probability, and we can further scale the solution by an $O(1)$ factor to ensure that $\sum_i x_{ij} \geq 1$ for each job j. Let us also assume here (to avoid some technical details), that each job j is *large* on every machine i in the sense that the ℓ_1 norm of its load vector satisfies $\sum_k p_{ijk} \geq 1/d$.

Consider round ℓ, where we round the values of $x_{ij}^{\ell-1}$ (as previously) at the end of iteration $\ell - 1$, to multiples of some suitably chosen $\epsilon_\ell \gg \epsilon_{\ell-1}$. The following observation is crucial.

▶ **Lemma 8.** *Consider round ℓ and apply the rounding to $x_{ij}^{\ell-1}$ mentioned above to obtain x_{ij}^{ℓ}. Let B_i be the event that the load on machine i increases by more than T for some coordinate k, and let A_j denote the event that for a job $\sum_i x_{ij}^{\ell} \leq 1/2$. Then each of these events depends on at most poly$(d, \epsilon_{\ell-1})$ other events.*

Proof. Two events B_i and $B_{i'}$ are dependent if and only if $x_{ij}^{\ell-1} > 0$ and $x_{i'j}^{\ell-1} > 0$ for some job j. Similarly, two events A_j and B_i are dependent if and only if some $x_{ij}^{\ell-1} > 0$. As each $x_{ij}^{\ell-1}$ is an integral multiple of $\epsilon_{\ell-1}$ and each job is large, the total number of jobs assigned to a machine can be at most $O(d^2 \cdot (1/\epsilon_{\ell-1}))$. Moreover, for a job j at most $O(1/\epsilon_\ell)$ variables $x_{ij}^{\ell-1}$ can be non-zero. Together this implies the claim. ◀

Thus by Lemma 7, we can choose $\epsilon_\ell = \Theta(\log\log(d/\epsilon_{\ell-1}) / \log(d/\epsilon_\ell))$ and ensure that none of the bad event happens. Thus crucial observation is that as we iterate the algorithm, the dependence on m in ϵ_ℓ becomes like $\log^{(\ell)} m$ (i. e. log iterated ℓ times) and eventually disappears[1], while the dependence on d converges to $\Omega(\log\log d / \log d)$.

[1] Strictly speaking, this gives an $\exp(O(\log^* m))$ guarantee. But a more gradual rounding with a slightly more careful of parameters gives theorem 6. We refer to [3] for details.

References

1. Nikhil Bansal, Moses Charikar, Ravishankar Krishnaswamy, and Shi Li. Better algorithms and hardness for broadcast scheduling via a discrepancy approach. In *SODA*, pages 55–71, 2014.
2. Nikhil Bansal and Janardhan Kulkarni. Minimizing flow-time on unrelated machines. *CoRR*, abs/1401.7284, 2014.
3. David G. Harris and Aravind Srinivasan. The moser-tardos framework with partial resampling. In *FOCS*, pages 469–478, 2013.
4. Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *FOCS*, pages 312–320, 1982.
5. Lap-Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, 2011.
6. Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.*, 46:259–271, 1990.
7. Shachar Lovett and Raghu Meka. Constructive discrepancy minimization by walking on the edges. In *FOCS*, pages 61–67, 2012.
8. Thomas Rothvoss. Approximating bin packing within o(log OPT * log log OPT) bins. In *FOCS*, pages 20–29, 2013.
9. Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *STOC*, pages 661–670, 2007.
10. David Williamson and David Shmoys. *The design of Approximation Algorithms*. Cambridge University Press, 2011.