Lattice-based Algorithms for Number Partitioning in the Hard Phase

Bala Krishnamoorthy, William Webb, and Nathan Moyer
Department of Mathematics, Washington State University, Pullman WA
{bkrishna, webb, nmoyer}@math.wsu.edu

Abstract

The number partitioning problem (NPP) is to divide n numbers $a_1, \ldots, a_n$ into two disjoint subsets such that the difference between the two subset sums – the *discrepancy*, Δ, is minimized. In the balanced version of NPP (BalNPP), the subsets must have the same cardinality. With a_js chosen uniformly from $[1, R]$, $R > 2^n$ gives the *hard* phase, when there are no equal partitions (i.e., $\Delta = 0$) with high probability (whp). In this phase, the minimum partition is also unique whp. Most current methods struggle in the hard phase, as they often perform exhaustive enumeration of all partitions to find the optimum.

We propose reductions of NPP and BalNPP in the hard phase to the closest vector problem (CVP). We can solve the original problems by making polynomial numbers of calls to a CVP oracle. In practice, we implement a heuristic which applies basis reduction (BR) to several CVP instances (less than $2n$ in most cases). This method finds near-optimal solutions without proof of optimality to NPP problems with reasonably large dimensions – up to $n = 75$. second, we propose a *truncated* NPP algorithm, which finds approximate minimum discrepancies for instances on which the BR approach is not effective. In place of the original instance, we solve a modified instance with $\bar{a}_j = \lfloor a_j/T \rceil$ for some $T \leq R$. We show that the expected optimal discrepancy of the original problem given by the truncated solution, $E(\Delta_T^*)$, is not much different from the expected optimal discrepancy: $E(\Delta_T^*) \leq E(\Delta^*) + nT/2$. This algorithm can be used to find good quality partitions within a short time for problems of sizes up to $n = 100$. Third, we propose a direct mixed integer programming (MIP) model for NPP and BalNPP. We then solve a lattice-based reformulation of the original MIP using standard branch-and-cut methods. Assuming it terminates, the MIP model is guaranteed to find the optimum partition.

1 Introduction

The number partitioning problem (NPP) is to divide n numbers $a_1, \ldots, a_n$ into two disjoint subsets such that the sums of the subsets are as close as possible to each other. In the *constrained* NPP, the difference of the cardinalities of the subsets is upper bounded by a small number [1]. The most common version of the constrained NPP is the balanced NPP (BalNPP), where the two subsets must have the same cardinality (or differ by 1, when n is odd). The absolute value of the difference between the subset sums is called the *discrepancy* of a given partition, and is denoted by Δ. A partition is *perfect* if $\Delta = 0$ (or $\Delta = 1$ when $\sum_j a_j$ is odd). NPP finds several practical and theoretical applications – multiprocessor scheduling, VLSI circuit design, cryptography, and choosing balanced sides for a ball game are a few of them [2].

NPP is known to be NP-complete [3]. Further, it is known to have *poor* heuristics [4, 5]. The best known polynomial time heuristic is the Karmarkar-Karp differencing method [6] (also known as KK heuristic), which runs in $O(n \log n)$ time. The bottleneck step is maintaining a sorted list

of the numbers as the algorithm continues to run. When the a_i's are random real numbers that are independent and identically distributed (i.i.d.) in the interval $[0, 1]$, Yakir [7] proved that the expected value of the discrepancy achieved by the KK heuristic is $O(n^{-0.72 \ln n})$. Mertens [8] has recently reported this bound to be exact. The median value of the optimum discrepancy was shown to be much smaller by Karmarkar et al. [9] – $O(\sqrt{n}\, 2^{-n})$. Lueker [10] derived the same bound for the expected optimal discrepancy. When the a_js are i.i.d. integers drawn from $[1, R]$, the expected optimal discrepancy is $E(\Delta^*) = O(\sqrt{n}\, 2^{-n} R)$ for NPP, and $E(\Delta^*) = O(n\, 2^{-n} R)$ for BalNPP [11, 5]. Korf [12] proposed a *complete* version of the KK heuristic (complete-KK, or CKK for short), which starts with the partition given by the KK heuristic, and produces better solutions as it continues to run. Mertens [5] proposed a similar complete differencing heuristic for BalNPP. Currently, the complete versions of differencing appear to be the best methods in practice.

NPP and BalNPPexhibits a phase transition when the size of a_js change. This phase transition has been fully characterized mathematically [13, 1]. When $R < 2^n$, there are many perfect partitions with high probability (whp) as $n \to \infty$, giving the *easy* phase. Since any perfect partition is optimal by definition, finding one will solve the problem. $R > 2^n$ gives the *hard* phase, as there are no perfect partitions as n increases in this range whp, and the optimum partition is also unique whp. Most known heuristics struggle in the hard phase, as the optimality of non-perfect partition(s) is not proven until examining all possibilities. Thus, running time of CKK is typically exponential in n for instances in the hard phase. Further, its rate of convergence is slow – $O(1/N^\theta)$ with $\theta < 1$, where N is the number of partitions examined so far. In fact, Mertens [5] argues that these heuristics cannot be significantly better than random search. He also suggests that a *better* description of NPP, distinct from the one provided in the context of differencing strategies, may be the key to achieve improved performances.

1.1 Our Contributions

We establish correspondences between number partitioning in the hard phase and lattice problems, using which we propose algorithms for NPP that are fundamentally distinct from differencing methods (we treat NPP and BalNPP in a unified way). First, we present an efficient reduction of NPP in the hard phase to an instance of the closest vector problem (CVP) in a lattice (see Section 3). We can solve NPP instances using a polynomial number of calls to a CVP oracle, assuming we have access to one. In practice, we propose a discrete search algorithm that uses basis reduction (BR) to tackle the CVP instances. Using Kannan's homogenization technique [14], we cast the instance of CVP as an instance of the shortest vector problem (SVP) in a related lattice. We apply BR to several (up to $2n$ in most cases) such SVP instances, which are generated using the estimates of the expected optimal discrepancy ($E(\Delta^*)$). Using routines for block Korkine-Zolotarev reduction [15], this BR heuristic finds optimal or near-optimal discrepancies for instances with n up to 75 in the hard phase within an hour of computation on a typical personal computer.

The discrete search algorithm does not find good partitions for larger problems within a few hours of computation. Hence we propose a *truncated* NPP heuristic to obtain approximate minimum discrepancies for larger NPP instances (see Section 4). In place of the original instance, we solve a modified NPP instance with numbers $\bar{a}_j = \lfloor a_j / T \rceil$ for some $T \leq R$. We show that the expected optimal discrepancy for the original problem corresponding to the solution of the truncated problem is not much different from the expected optimal discrepancy: $E(\Delta_T^*) \leq E(\Delta^*) + nT/2$. A natural choice is to set $T = 10^t$ for $t \geq 1$, which amounts to truncating the last t digits from each a_j. For large values of n on which the discrete search is not efficient, we choose T such that the

truncated instances are well into the easy phase, and hence are solved quickly by a lattice-based approach to solve standard knapsack problems [16]. We are able to obtain good quality solutions to problems of sizes up to $n = 100$ within a few minutes of computation using this approach.

The discrete search and truncation algorithms find partitions without proof of their optimality or quality. In order to get a certificate of optimality of the solution obtained using the lattice based approaches, we propose a direct mixed integer programming (MIP) model for NPP and BalNPP (see Section 5). Using a framework similar to that used for reducing NPP to CVP, we solve a BR-based reformulation of the original MIP using a commercial solver employing branch-and-cut methods. The original MIP formulation cannot be handled efficiently by current solvers, mainly due to the huge numbers involved in the hard phase of NPP. The MIP model is guaranteed to find the optimal discrepancy, as long as the solution process is completed. Instances with up to $n = 50$ in the hard phase are solved efficiently within several minutes.

1.2 Related Work

NPP is related to the subset sum problem (or 0-1 knapsack problem), which is used in the study of lattice problems such as CVP and SVP [17, Chap. 3,4]. The correspondence is immediate for an NPP instance with at least one perfect partition (i.e., in the easy phase), but has not been studied previously for instances in the hard phase. In this context, Howgrave-Graham and Joux have recently proposed some generic algorithms to solve hard knapsack problems, i.e., with density very close to 1 [18]. Their algorithms are motivated by the algorithm for knapsack problems proposed by Schroeppel and Shamir thirty years ago [19]. These approaches could be adapted to tackle an instance of NPP in the hard phase by solving a series of related knapsack problems, the approach being termed the approximate knapsack problem. Our lattice-based approaches to solve NPP and BalNPP instances appears to do better than the above approaches to solve them as approximate knapsacks for dimensions up to $n = 75$ (see Subsection 3.2.2).

BR-based reformulations for integer programs (IPs) have been studied by several researchers [20, 21, 22, 23]. Our reformulation of the MIP model for NPP is a generalization of the preconditioning method for IPs termed column basis reduction, proposed earlier by the first author and Pataki [23], to the case of MIPs. Lattice-based approaches specifically for NPP in the hard phase are new, and are fundamentally different from previous approaches, most of which are based on differencing strategies. CVP is NP-hard, and so is SVP under randomized reductions [17]. At the same time, current algorithms can solve instances of CVP and SVP for reasonably large dimensions efficiently [24, 25], thus making our methods competitive in practice for number partitioning in the hard phase.

2 Definitions, and Notation

In the context of an NPP with n positive numbers $a_1, \ldots, a_n$, a *partition* defines two sets of indices $S_1 \subset \{1, \ldots, n\}$ and $S_2 = \{1, \ldots, n\} \setminus S_1$. A balanced partition must satisfy $| \, |S_1| - |S_2| \, | \leq 1$. The discrepancy associated with a partition is $\Delta = |\sum_{j \in S_1} a_j - \sum_{j \in S_2} a_j|$. We assume the a_js are i.i.d. among the integers in the interval $[1, R]$ for some $R \in \mathbb{Z}_{>0}$. Without loss of generality, we assume that $\alpha = \sum_j a_j$ is even. If the sum is indeed odd, we can reduce any one a_j by one, and equivalently solve the resulting instance. Note that when α is even, Δ for any partition will also be even.

We will use bold lower case letters to denote vectors, e.g., $\mathbf{a} = (a_1, \ldots, a_n)$. Whether a vector is a row or a column vector is evident from the context. We use upper case letters to denote matrices. We let $E(\Delta)$ denote the (theoretical) expected value of the random variable Δ, and use $\langle\Delta\rangle$ to denote the average value of Δ as observed in computations.

Connection to 0-1 knapsack problem We let $\beta = \alpha/2$. Each partition corresponds to the solutions of two related 0-1 knapsack problems:

$$\sum_{j=1}^{n} a_j x_j = \beta \pm \Delta/2, \quad x_j \in \{0,1\}\, \forall j.$$

If $x_j = 1$ for all $j \in S_1$ (and zero otherwise) solves the knapsack with right-hand side (rhs) equal to $\beta - \Delta/2$, then $x'_j = 1$ for all $j \in S_2$ solves the knapsack with the rhs of $\beta + \Delta/2$ (and vice-versa). For instances of NPP in the easy phase ($\Delta = 0$), we could solve the binary knapsack problem with rhs of β to recover a perfect partition. On the other hand, this correspondence cannot be utilized directly for NPP instances in the hard phase, as $\Delta > 0$ is unknown to start with. Instead, we use the connection to define and study a decision (or promise) version of NPP, which we subsequently reduce to the closest vector problem. We denote this problem as Decision NPP (DNPP for short) indexed by a parameter d (we do not include the a_js in this notation for the sake of brevity).

Definition 1. (DNPP_d) *Given numbers $a_1, \ldots, a_n$ and an even number $2d$, decide if a partition exists with $\Delta \leq 2d$. Equivalently, find a solution $x_j \in \{0,1\}\, \forall j$ to the knapsack $\sum_j a_j x_j = \beta - \delta$ for some $\delta \leq d$, if one exists.*

The original (search) version of NPP is not substantially harder than DNPP. More precisely, given an oracle to solve DNPP, we can solve the search version of NPP by making a polynomial number of calls to the oracle, while performing a binary search on the value of the optimal discrepancy (see Section 3.1).

We define a similar decision version for BalNPP, which we term DBalNPP.

Definition 2. (DBalNPP_d) *Given numbers $a_1, \ldots, a_n$ and an even number $2d > 0$, decide if a balanced partition exists with $\Delta \leq 2d$. Equivalently, find a solution $x_j \in \{0,1\}\, \forall j$ that satisfies $\sum_j x_j = \lfloor n/2 \rfloor$ to either the knapsack $\sum_j a_j x_j = \beta - \delta$ or to the knapsack $\sum_j a_j x_j = \beta + \delta$, for some $\delta \leq d$, if one exists.*

Notice that we are considering both knapsacks in order to account for the case when n is odd. In this case, the subset with the lower (by one) cardinality can have either the lower or the higher sum. We also point out that the definitions of DNPP_d and DBalNPP_d are applicable with $d = 0$, in which case they are equivalent to the original knapsack (with rhs β).

Background on lattices Let $\boldsymbol{b_1}, \ldots, \boldsymbol{b_n}$ be a set of n linearly independent vectors in $\mathbb{R}^m$ with $m \geq n$. The lattice generated by these vectors is the set of all integer linear combinations of them:

$$\mathbb{L}(\boldsymbol{b_1}, \ldots, \boldsymbol{b_n}) = \{\, \textstyle\sum_j \boldsymbol{b_j} x_j : x_j \in \mathbb{Z} \,\}.$$

$\boldsymbol{b_1}, \ldots, \boldsymbol{b_n}$ forms a *basis* for the lattice $\mathbb{L}$. Putting these vectors into an $m \times n$ basis matrix B, vectors in $\mathbb{L}$ are also described by $B\mathbf{x}$ for some $\mathbf{x} \in \mathbb{Z}^n$. We use the Euclidean norm for vectors, unless mentioned otherwise. Intuitively, a *reduced* basis for a lattice consists of vectors that are

"short" and "nearly orthogonal", and can be viewed as an extension to lattices of the concept of a Gram-Schmidt orthogonalized basis of a real space. Several notions of reduced basis have been proposed [26, 27], and several classes of algorithms are also available to perform basis reduction [27, 28, 29, 15].

Two fundamental problems related to a lattice are the shortest vector problem and the closest vector problem. Given a lattice basis $B \in \mathbb{Z}^{m\times n}$, the shortest vector problem (SVP) is to find a non-zero vector in the lattice $B\mathbf{x}$ with $\mathbf{x} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ such that $\| B\mathbf{x} \| \leq \| B\mathbf{y} \|$ for all $\mathbf{y} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$. Given a lattice basis $B \in \mathbb{Z}^{m\times n}$ and a target vector $\mathbf{u}$, the closest vector problem (CVP) is to find a lattice vector $B\mathbf{x}$ with $\mathbf{x} \in \mathbb{Z}^n$ such that $\| B\mathbf{x} - \mathbf{u} \| \leq \| B\mathbf{y} - \mathbf{u} \|$ for any $\mathbf{y} \in \mathbb{Z}^n$. These are the search (default) versions of SVP and CVP. We study the decision version of CVP, termed DCVP, to which we will reduce DNPP.

Definition 3. (DCVP) *Given a lattice basis $B \in \mathbb{Z}^{m\times n}$, a target vector $\mathbf{u}$, and a rational $\gamma > 0$, decide if $\| B\mathbf{x} - \mathbf{u} \| \leq \gamma$ for some $\mathbf{x} \in \mathbb{Z}^n$, or if $\| B\mathbf{x} - \mathbf{u} \| > \gamma$ for all $\mathbf{x} \in \mathbb{Z}^n$.*

We denote such an instance of DCVP by $(B, \mathbf{u}, \gamma)$. A similar decision version can be defined for SVP as well. The search and promise versions of SVP and CVP are equivalent [17]. Basis reduction is one of the staple methods used for solving instances of SVP and CVP [25]. We now present the reductions of decision versions of NPP and BalNPP to decision versions of CVP.

3 Reducing NPP to CVP

Our reductions of NPP and BalNPP to CVP are motivated by the surprisingly simple reduction of the subset sum problem to CVP presented originally by Micciancio [30] (also presented in [17, Chap 3]). We believe it is non-trivial to obtain a similarly simple reduction of number partitioning in the *hard phase* to CVP, especially for the balanced version. Further, our reductions subsume the reduction of subset sum to CVP as a special case.

Notice that the *existence* of a reduction from NPP to CVP is not a surprising fact, given that both problems are NP-complete. The purpose of our reductions is not just to prove the equivalence of the two problems, or the NP-completeness of NPP. The sole purpose of these reductions is to motivate the creation of a new family of lattice-based algorithms to NPP and BalNPP problems specifically in the hard phase. These lattice-based approaches and their computational evaluation constitute the main contribution of this paper.

Theorem 4. *For any integer $d > 0$,* DNPP_d *is reducible to* DCVP.

Proof Given the vector of numbers $\mathbf{a} = (a_1, \ldots, a_n)$ associated with the DNPP_d, we create an $(n+1) \times n$ basis matrix B and a target vector $\mathbf{u}$ as follows:

$$B = \begin{bmatrix} 2d\, I \\ \mathbf{a} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} d\,\mathbf{1} \\ \beta \end{bmatrix}. \tag{3.1}$$

I is the $n \times n$ identity matrix and $\mathbf{1}$ is the n-vector of all ones. The output of the reduction is the instance of DCVP specified as $(B, \mathbf{u}, d\sqrt{n+1})$. To be exact, we need to specify a rational number in place of the bound $d\sqrt{n+1}$. We can use any rational number γ such that $d\sqrt{n+1} \leq \gamma < \sqrt{d^2(n+1)+1}$. For instance, we can choose $\gamma = \sqrt{\left(d^2(n+1) + \frac{1}{4}\right)} = \sqrt{(4d^2(n+1)+1)}/2$.

We can then evaluate this square root to a constant precision – two digits past the decimal place will do. Recall that we can find square roots, and do multiplication and division in time and space polynomial in the input size [31]. Dividing the approximation to $\sqrt{(4d^2(n+1)+1)}$ by 2, we can indeed construct a suitable γ that has size polynomially bounded in d and n, in polynomial time as well.

To show this reduction is correct, we show that if DNPP$_d$ is a YES instance, then $(B, \mathbf{u}, d\sqrt{n+1})$ is a YES instance of DCVP; while if DNPP$_d$ is a NO instance, then $(B, \mathbf{u}, d\sqrt{n+1})$ is a NO DCVP instance.

($\Rightarrow$) Assume DNPP$_d$ is a YES instance. Let $\mathbf{x}$ be the 0-1 solution satisfying $\mathbf{ax} = \beta - \delta$ for some $\delta \leq d$. Consider the distance between the target vector $\mathbf{u}$ and the lattice vector $B\mathbf{x}$:

$$\left\| \begin{bmatrix} d(2\mathbf{x} - \mathbf{1}) \\ \mathbf{ax} - \beta \end{bmatrix} \right\|^2 = d^2 \sum_j (2x_j - 1)^2 + (\mathbf{ax} - \beta)^2 = d^2 n + \delta^2 \leq d^2(n+1). \tag{3.2}$$

The second equality uses the fact that $x_j \in \{0, 1\}$, and hence $2x_j - 1 = \pm 1$ for all j. Thus, $\|B\mathbf{x} - \mathbf{u}\| \leq d\sqrt{n+1}$ showing that $(B, \mathbf{u}, d\sqrt{n+1})$ is a YES instance of DCVP.

($\Leftarrow$) We show that if $(B, \mathbf{u}, d\sqrt{n+1})$ is a YES instance of DCVP, then DNPP$_d$ is a YES instance. Let $\mathbf{x} \in \mathbb{Z}^n$ (not necessarily binary) be such that $\| B\mathbf{x} - \mathbf{u} \| \leq d\sqrt{n+1}$. The upper bound given in Equation (3.2) holds in this case as well (without involving δ). We note that each $2x_j - 1$ is an odd number, and hence contributes at least 1 to the summation term. Thus $\sum_j (2x_j - 1)^2 \geq n$. Hence, for the above upper bound to hold, it must be the case that $\sum_j (2x_j - 1)^2 = n$, and thus each $x_j \in \{0, 1\}$. Indeed, if one of the terms $|2x_j - 1|$ is larger than 1, then it must be at least 3, and hence the sum will be at least $(n + 8)$. It immediately follows that the contribution of the second term in the squared norm has to be at most d^2, thus showing that DNPP$_d$ is a YES instance. □

For the case of BalNPP, we add the cardinality constraint that forces the partition to be balanced to the basis matrix and the target vector of the DCVP instance. It is critical to choose the weight for this constraint appropriately for this reduction to work – we choose the weight as $d + 1$.

Theorem 5. *For any integer $d > 0$,* DBalNPP$_d$ *is reducible to* DCVP.

Proof Given the vector of numbers $\mathbf{a} = (a_1, \ldots, a_n)$ associated with DBalNPP$_d$, we create an $(n + 2) \times n$ basis matrix B' and a target vector $\mathbf{u}'$ as follows:

$$B' = \begin{bmatrix} 2d\, I \\ (d+1)\mathbf{1}^T \\ \mathbf{a} \end{bmatrix}, \quad \mathbf{u}' = \begin{bmatrix} d\,\mathbf{1} \\ (d+1)\lfloor n/2 \rfloor \\ \beta \end{bmatrix}. \tag{3.3}$$

The output of the reduction is the instance of DCVP specified as $(B', \mathbf{u}', d\sqrt{n+1})$. We can replace the distance bound $d\sqrt{n+1}$ with an appropriate rational number γ, and prove the correctness of this reduction, similar to the case of DNPP.

($\Rightarrow$) Assume DBalNPP$_d$ is a YES instance. Let $\mathbf{x}$ be the 0-1 solution satisfying one of the two knapsacks $\mathbf{ax} = \beta \pm \delta$ for some $\delta \leq d$, and $\sum_j x_j = \lfloor n/2 \rfloor$. Consider the distance between the target vector $\mathbf{u}'$ and the lattice vector $B'\mathbf{x}$:

$$\begin{aligned} \left\| \begin{bmatrix} d(2\mathbf{x} - \mathbf{1}) \\ (d+1)(\sum_j x_j - \lfloor n/2 \rfloor) \\ \mathbf{ax} - \beta \end{bmatrix} \right\|^2 &= d^2 \sum_j (2x_j - 1)^2 + (d+1)^2 (\sum_j x_j - \lfloor n/2 \rfloor)^2 + (\mathbf{ax} - \beta)^2 \\ &= d^2 n + \delta^2 \;\leq\; d^2(n+1). \end{aligned} \tag{3.4}$$

Thus we get $\|B'\mathbf{x} - \mathbf{u}'\| \leq d\sqrt{n+1}$ showing that $(B', \mathbf{u}', d\sqrt{n+1})$ is a YES instance of DCVP.

($\Leftarrow$) We show that if $(B', \mathbf{u}', d\sqrt{n+1})$ is a YES instance of DCVP, then DBALNPP$_d$ is a YES instance. Let $\mathbf{x} \in \mathbb{Z}^n$ (not necessarily binary) be such that $\| B'\mathbf{x} - \mathbf{u}' \| \leq d\sqrt{n+1}$. The upper bound given in Equation (3.4) holds in this case as well (without involving δ). Again, since $2x_j - 1$ is odd, $\sum_j (2x_j - 1)^2 \geq n$. Hence, for the upper bound to hold, it must be the case that $\sum_j (2x_j - 1)^2 = n$, and thus each $x_j \in \{0, 1\}$. Similarly, $\sum_j x_j = \lfloor n/2 \rfloor$, else the upper bound will not hold due to the weight of $(d+1)^2$. Hence, we must have $|(\mathbf{ax} - \beta)| \leq d$, thus showing that DBALNPP$_d$ is a YES instance. □

We make the following observations about the reductions.

- These reductions subsume the cases of perfect partition ($d = 0$) in the following sense. We just need to modify the outputs of the reductions slightly – use $d = 1$ in $B, \mathbf{u}$ or $B', \mathbf{u}'$. The bound on the distance is given as $\sqrt{n}$, again by setting $d = 1$, and decreasing the factor $n+1$ to n, as we desire $\mathbf{ax} = \beta$.
- The reductions apply without modification to the cases of partitioning numbers into unequal sets. For instance, if we want to assign 30% of the total sum to one subset (instead of 50%), we just set $\beta = \lfloor 0.3\alpha \rfloor$.
- The reductions apply without modification to the cases of constrained partitions that are not balanced. We just replace $\lfloor n/2 \rfloor$ with the appropriate number in this case.

3.1 A binary search algorithm for NPP

The reductions of the decision problems presented in the last section suggest an algorithm to solve the search versions of NPP and BALNPP. Assuming we have an oracle that solves DCVP, we perform a binary search on the discrepancy values, calling the oracle a polynomial number of times (see Fig. 1). We reduce the size of the interval for the optimal discrepancy value by at least a factor of half in each step, until we collapse it to a single number. The number of calls to the oracle is $O(\log \beta)$, which is in fact a very conservative estimate. The expected value of the optimum discrepancy is much smaller than R, and hence the number of calls needed is also much smaller in expectation. Micciancio and Voulgaris [32] have recently proposed a deterministic algorithm for most lattice problems including CVP, which runs in $\tilde{O}(2^{2n+o(n)})$ time. This algorithm is the most efficient one known currently for SVP and CVP. Using this algorithm to solve each DCVP instance, the binary search algorithm will have a running time of $\tilde{O}(2^{2n+o(n)} \log \beta)$.

While this scheme appears promising, we have two main concerns going against it. First, there is no such oracle available for DCVP. There are several algorithms that come close [15, 25], at least in practice; but they are not *guaranteed* to solve every instance. Many of these methods try to solve a corresponding instance of the shortest vector problem, often using basis reduction. Given an instance of DCVP $(B, \mathbf{u}, d\sqrt{n+1})$ as specified in Equation (3.1), we create the matrix

$$D = \begin{bmatrix} B & \mathbf{u} \\ \mathbf{0} & M \end{bmatrix} = \begin{bmatrix} 2d\, I & d\mathbf{1} \\ \mathbf{a} & \beta \\ \mathbf{0} & M \end{bmatrix}, \quad \text{where } M \text{ is a suitably large number.} \tag{3.5}$$

We then try to solve the decision SVP (DSVP) instance for the lattice $\mathbb{L}(D)$ by adapting the methods of Lagarias and Odlyzko [33], and that of Coster et al. [16] to our case. If $\mathbf{x}$ is a 0-1

```
BINARY SEARCH FOR NPP

Δ_l := 0, Δ_u := β;
while Δ_u − Δ_l > 1
      2d := (Δ_u + Δ_l)/2;
      solve DNPP_d by solving DCVP_d;
      if DNPP_d ≡ YES
         Δ_u := min{ Δ(DNPP_d), 2d };
      else
         Δ_l := 2d;
      end_if
end_while
```

Figure 1: A binary search algorithm to solve NPP by solving several instances of DCVP. The quantity $\Delta(\text{DNPP}_d)$ represents the minimum discrepancy obtained when solving the YES instance of DNPP_d.

vector such that $\| B\mathbf{x} - \mathbf{u} \| \leq d\sqrt{n+1}$, thus making the DCVP a YES instance, then the vector $\mathbf{z} = [d(2\mathbf{x} - \mathbf{1})\ (\mathbf{a}\mathbf{x} - \beta)\ - M]^T$ is a short(est) vector in $\mathbb{L}(D)$ for appropriately chosen values of M. Equivalently, the vector $\tilde{\mathbf{z}} = [d(2\mathbf{x} - \mathbf{1})\ (\mathbf{a}\mathbf{x} - \beta)]^T$ is a short vector in $\mathbb{L}(\tilde{D})$, where $\tilde{D} = [B\ \mathbf{u}]$, with $\| \tilde{\mathbf{z}} \| \leq d\sqrt{n+1}$. We apply basis reduction on D, and search for vectors of the form $\mathbf{z}$ in the reduced D matrix. This reduction of DCVP to DSVP is essentially the homogenization technique proposed by Kannan [14], which is not exact [17]. Note that the argument for this reduction works for the case of balanced partition as well by using $B', \mathbf{u}'$ in place of $B, \mathbf{u}$.

Once we have identified a vector $\tilde{\mathbf{z}}$ of this form, a straightforward calculation will yield the corresponding $\mathbf{x}$ vector, and hence the partition. On the other hand, if no such $\tilde{\mathbf{z}}$ vector is obtained, we conclude that the DSVP instance is a NO instance. Unfortunately, the latter option does not always lead to accurate conclusions – the method may not identify any vectors $\tilde{\mathbf{z}}$ even when there exists a corresponding vector $\mathbf{x}$. We implemented this method, and it is remarkably efficient for smaller problems ($n \leq 40$). When solving DNPP_d, we often find a partition with Δ much smaller than $2d$, which results in the binary search converging faster. But for larger n, the method often fails to identify *any* partition after all the runs, i.e., after we get $\Delta_u - \Delta_l \leq 1$.

The second drawback of this algorithm is that it does not utilize the estimates available for the expected optimal discrepancies ($E(\Delta^*)$). We start the calculations with $\Delta_u = \beta$, while the actual discrepancy is typically much smaller. Taking all these observations into account, we propose the following rather straightforward algorithm, which works quite efficiently in practice even though we do not provide a proof of convergence.

3.2 A discrete search algorithm for NPP

We propose a very simple heuristic that uses the estimates on $E(\Delta^*)$ to generate a *small* number of DCVP instances, and tackles them using BR as described in Equation (3.5). We set $d = cE(\Delta^*) = c\sqrt{2\pi/3}\sqrt{n}\,2^{-n}\,R$ for NPP, and use $c\sqrt{\pi/3}\,n\,2^{-n}R$ for BALNPP for a parameter c. We then vary the value of c over a pre-determined range by choosing several ($O(n)$) values from $[2/n, 2n+2]$. We use the block Korkine-Zolotarev (BKZ) reduction routines [15] available in the Number Theory Library [34]. We are not guaranteed to find the $\mathbf{z}$ vectors for every D. We first try n values of the

form $2i/n$ for $i = 1, \ldots, n$ for c. If no 0-1 solutions are found, we try n more values of the form $2i$ for $i = 2, \ldots, n$. We stop the sequence of DCVP instances once we find a partition with good quality, i.e., with $\Delta \leq 2nE\,(\Delta^*)$. The treatment of DBALNPP is identical to that of DNPP.

The results of our tests are presented in Figure 2. Near optimal discrepancies are consistently obtained for $n \leq 75$ without proof of optimality. All computations were performed on an Intel Pentium 4 PC with 3.2 GHz CPU and 1 MB RAM. For each n, the value of R is chosen such that the instances generated are in the hard phase with high probability – typically, one or two orders of magnitude larger than $E\,(\Delta^*)$. The details are given in Table 1 in the Appendix. The average best discrepancies over 100 trials are plotted.

For each of the $2n$ values of c we try, we apply basis reduction to the original D matrix described in Equation 3.5. If no good partition is found even after the $2n$ runs, we repeat the procedure on five different random permutations of each D matrix considered in the previous run. As the dimension grows, we will have to try more number of such randomly permuted starting bases until we get a good solution. We do not have a bound on the number of such bases we have to consider for each n, but it seems to grow with increasing n. For $n = 80$ with three random permutations considered for each D in the original sequence, we are able to find a good partition in 9712 seconds, but the success percentage drops to 74% for $n = 85$ with only three passes each. In other words, no 0-1 short vector is found as a solution by the discrete search heuristic for 26% of the problem instances. For these instances, we have to try several more random permuted starting bases, resulting in large increases of computational time.

3.2.1 Comparison to Complete Differencing

For comparison purposes, we have implemented the complete KK (CKK) differencing algorithm for NPP [12] and the complete balanced largest differencing method (CBLDM) [35] for BALNPP. In the original papers, these authors report the number of nodes examined by the complete versions of the differencing methods as a measure of the hardness of solving the instances. In our case, we run these heuristics for the same time limit as taken by the basis reduction heuristic, and record the best discrepancy obtained by that time. To be conservative, we actually ran CKK and CBLDM for 110% of the average time taken by the BR heuristic on instances of the same size (n and R). BR heuristic consistently finds smaller discrepancies (better by orders of magnitude) than CKK and the CBLDM heuristic. The improvements appear to be increasing with n, although the computational time is also increasing quite fast. We have also recorded the average discrepancies found by the KK and the BLDM heuristics, which are the values corresponding to the first partition found by these differencing strategies.

While both Korf [12] and Mertens [35] mention the steps to recover the actual partition from their differencing methods, they do not give the details of this step for the complete versions of their algorithms. The partition is recovered by two-coloring a graph that gets built as the differencing algorithm runs. We start with one node representing each number (thus a total of n nodes). When we replace two numbers in the list by their difference (and commit to place them in different subsets), we add an edge between the corresponding nodes. The partition is obtained by two-coloring the resulting graph, which will be a tree. In the complete versions of differencing, we also replace two numbers by their sum (and commit to place them in the same subset). We add an extra node to the graph, and add two edges connecting the extra node and the nodes corresponding to the two numbers. The partition is recovered by two-coloring the resulting graph, which will again be a tree. Since the BR heuristic automatically provides the partition (and not just the discrepancy)

we find the partitions in the case of CKK and CBLDM algorithms, for the sake of fair comparison. The previous authors may well have been locating the partitions, but their papers [12, 35] do not mention the details. In our implementation, the times taken by the two-coloring components of the complete algorithms were not negligible.

We must add a note about the slight up-and-down nature of the curves of discrepancy values for increasing n values in Figure 2. These variations reflect the choices of R for each n (see Table 1 in the Appendix). For example, the estimate for the optimal discrepancy of an NPP instance with $n = 35$ and $R = 10^{12}$ is

$$\Delta^* = \sqrt{\frac{2\pi}{3}}\,\sqrt{n}\,2^{-n}\,R = 249.18,$$

while the same estimate for $n = 40$ and $R = 10^{13}$ is 83.25. We made the choices of R such that these estimates exceed zero. To make sure, we also checked whether the partition found happens to be perfect, and did not include such partitions in the calculations. As expected, perfect partitions were found only rarely given the choices of R values.

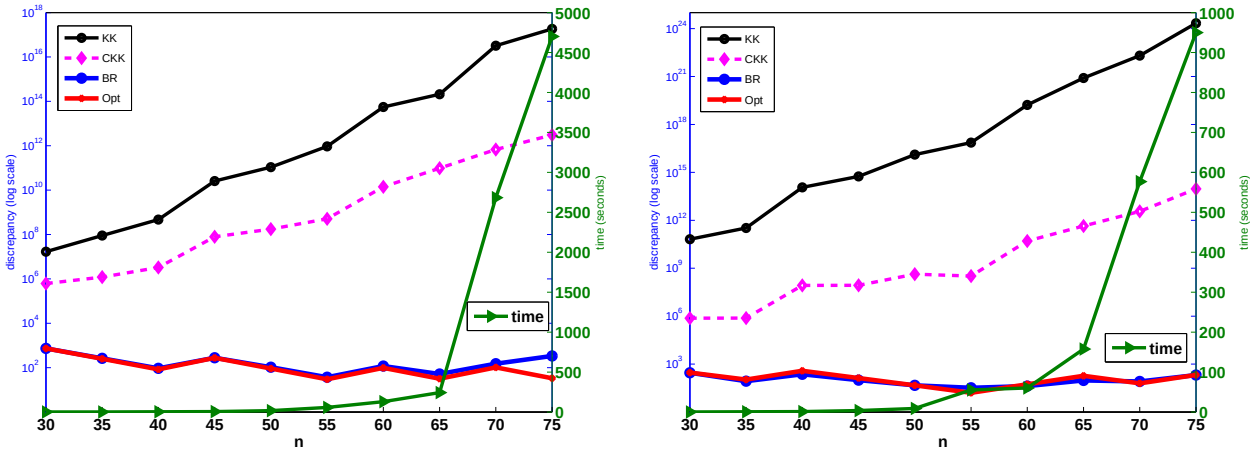

Figure 2: Comparison of discrepancies given by the BR heuristic, KK, and CKK heuristics for NPP instances (on left) and BalNPP instances (on right). The expected optimum discrepancies are also plotted. Discrepancies are plotted on log scale. Average run times are plotted on the right axis.

3.2.2 Comparison to Schroeppel-Shamir algorithm

Schroeppel and Shamir (SS) proposed a $O(2^{n/2})$ time and $O(2^{n/4})$ space algorithm to solve the knapsack problem [19]. It was claimed in this paper that knapsacks with sizes up to $n = 100$ could be solved in practice at that time using this algorithm, even though no detailed computational evidence was provided. In the context of number partitioning, this algorithm solves BalNPP in the easy phase. The algorithm could be easily modified to solve the unbalanced case, i.e., NPP, in the easy phase with the same time and space complexities. This work did not explicitly address the case of NPP and BalNPP in the hard phase. Howgrave-Graham and Joux (HJ) have recently proposed a new family of generic algorithms for the knapsack problem [18], which improve the running time bound to $O(2^{0.3113n})$. A natural way to use these algorithms for knapsacks to solve NPP and BalNPPin the hard phase is to solve a series of knapsack problems with varying target

(right-hand side) values to find the optimal choice. Termed the approximate knapsack problem [18], the number of times HJ or SS algorithm gets applied depends on the size of the bound we have on the target value. In the worst case, we have to solve $O(\beta)$ individual knapsacks as part of this approach. Hence the time complexity of the approximate knapsack problem to solve NPP and BALNPP is $O(2^{0.3113n}\,\beta)$. For instances that are well into the hard phase, we could have $\beta \geq 2^{2n}$ – see details for $n = 60$ instances discussed below. Recall that the running time of the CVP-based binary search algorithm (Figure 1) depends only on $\log \beta$.

Howgrave-Graham and Joux have given estimates for running times of SS and HJ algorithms on an $n = 96$ instance of knapsack problem with n bit length numbers [18], i.e., with density of 1. They estimate their implementation of SS algorithm could take $1,500$ *days* on a single computer with typical configuration, when a good decomposition into four balanced quarters is not readily available for the instance. In comparison, the HJ algorithm could take only up to 5 days for such an instance. Of course, the SS and HJ algorithms are guaranteed to find the optimal solution for the knapsack problem when they terminate. On the other hand, a heuristic version of the HJ algorithm found a solution in less than 10 hours. Notice that these times are for solving instances of the knapsack problem, even though they have high density. This calculation needs to be repeated many times to solve instances of number partitioning in the hard phase. At this point, our discrete search heuristic is not as effective as the HJ algorithm for $n = 95$ to find near optimal solutions, even though we find good quality solutions within 63 seconds using the truncation heuristic – see Section 4.

In a personal communication, Joux mentioned that their implementation of the SS algorithm on a knapsack instance with $n = 60$ numbers each of bit length 70 took around 11 minutes to solve as an approximate knapsack problem. It was also reported that this value of n was not large enough for the HJ algorithm to perform better than SS. We tried our discrete search algorithm on $n = 60$ for $R = 10^{21}$, and obtained an average discrepancy of 15,798 in an average time of 116.2 seconds (averages over 100 random instances). Note that the average time for this set of instances is slightly smaller than, but is comparable to, the average time we observed for $n = 60$ instances with $R = 10^{19}$ (Table 1). The expected optimal discrepancy for this choice of parameters is $E(\Delta^*) = 9723$. We also tried the discrete search method for $n = 60$ instances with $R = 10^{36}$, which corresponds to a bit length of around 120 ($\log_2(10^{36}) = 119.6$). We obtained an average discrepancy of 9.534×10^{18}, in the average time of 158.1 seconds (over 100 instances). The expected optimal discrepancy is $E(\Delta^*) = 9.723 \times 10^{18}$. With the bit size of $2n$, the SS algorithm would be expected to take a much longer time to solve the approximate knapsack problem, while the running time of the discrete search algorithm is still comparable to the previous cases with smaller bit lengths.

4 Truncated NPP

The discrete search algorithm is currently not efficient for hard phase NPP instances of size $n = 80$ or higher. When n is fixed, the expected optimal discrepancy value depends linearly on the size parameter R. Using this relationship as the key, we propose the *truncated* NPP heuristic, whose goal is to find a good quality partition *quickly* when n is large. We solve a truncated instance in place of the original instance, denoted in short as TRUNCNPP.

Definition 6. (TRUNCNPP) *Given an* NPP *instance $a_1, \ldots, a_n$ and a real number $T > 0$, the truncation parameter, solve a new NPP instance with numbers $\bar{a}_j = \lfloor a_j/T \rceil$. Let $\bar{\beta} = \sum_j \bar{a}_j/2$, and let $\bar{\mathbf{x}}$ be an optimal solution to the new problem. Using $\bar{\mathbf{x}}$ as a solution to the original* NPP*, we*

get a discrepancy $\Delta_T = 2\,|\sum_j a_j\bar{x}_j - \beta|$. *The smallest such discrepancy obtained by truncation is denoted by* Δ_T^*.

A natural choice for T is 10^t for some integer $t \geq 0$. In this case, we cut off the last t digits from each number a_j to obtain the truncated problem. We now bound the quality of the expected optimal discrepancy obtained using TruncNPP. The expectation is taken over the numbers (a_js), as described by Borgs et al. [13].

Theorem 7. *The expected optimal discrepancy obtained from* TruncNPP *satisfies* $E(\Delta_T^*) \leq E(\Delta^*) + nT/2$, *where* $E(\Delta^*)$ *is the expected optimal discrepancy and* T *is the truncation parameter.*

Proof Let $\bar{a}_j = \lfloor a_j/T \rceil = a_j/T + \epsilon_j$ for $\epsilon_j \in (-1/2,\, 1/2]$. With the optimal solution $\bar{\mathbf{x}}$ for TruncNPP, the optimal discrepancy of the truncated problem is given as

$$\begin{aligned}
\bar{\Delta} &= 2\left|\textstyle\sum_j \bar{a}_j\bar{x}_j - \bar{\beta}\right| = 2\left|\sum_j (a_j/T)\bar{x}_j + \sum_j \epsilon_j\bar{x}_j - 1/2\sum_j (a_j/T + \epsilon_j)\right| \\
&= 2\left|1/T\left(\textstyle\sum_j a_j\bar{x}_j - \beta\right) + \left(\sum_j \epsilon_j\bar{x}_j - 1/2\sum_j \epsilon_j\right)\right| \\
\Rightarrow\quad \bar{\Delta} &\geq (1/T)\Delta_T - 2\left|\textstyle\sum_j \epsilon_j\bar{x}_j - 1/2\sum_j \epsilon_j\right| \\
\Rightarrow\quad \Delta_T &\leq T\bar{\Delta} + T\epsilon, \quad \text{where } \epsilon = 2\left|\textstyle\sum_j \epsilon_j\bar{x}_j - 1/2\sum_j \epsilon_j\right| = \left|\sum_j \epsilon_j(2\bar{x}_j - 1)\right|. \\
\text{Hence, } E(\Delta_T^*) &\leq TE\left(\bar{\Delta}^*\right) + nT/2,
\end{aligned}$$

as $\epsilon \leq n/2$, with the upper bound attained when $\epsilon_j = 1/2$ for all j with $x_j = 1$, and all remaining $\epsilon_j = -1/2$. The size parameter for $\bar{a}_j$'s is R/T. Hence the expected value of the optimal discrepancy for the truncated problem satisfies $E\left(\bar{\Delta}^*\right) = O(\sqrt{n}\,2^{-n}\,R/T)$. Hence $TE\left(\bar{\Delta}\right) = O(\sqrt{n}\,2^{-n}\,R) = E(\Delta^*)$, the expected optimal discrepancy for the original NPP instance. □

Notice that if T is large enough such that the truncated NPP has $E\left(\bar{\Delta}^*\right) = 0$, then the upper bound of $nT/2$ holds for $E(\Delta_T^*)$. In other words, if we truncate the problem to the easy phase, the discrepancy thus obtained will have the size of the truncation parameter times n on average. The upper bound on ϵ is rather conservative. Since a_j are i.i.d. integers, it is reasonable to expect the ϵ_j also to be i.i.d. in $(-1/2,\, 1/2]$. Under some more simplifying assumptions (independence of x_j distribution from ϵ_j) the expected value of ϵ will be much smaller than $nT/2$.

We illustrate the effectiveness of TruncNPP to find good quality solutions for large instances of NPP and BalNPP within a short time in Figure 3. Each point on the graphs represents the average best discrepancies over 100 trials. For n in the range 80 to 100, we choose R for the original NPP instance such that it is in the hard phase whp. Since the discrete search heuristic is not efficient, we choose the truncation parameter T such that the truncated instance is well into the easy phase, i.e., it is a 0-1 knapsack problem. We solve the subset sum problem using the BR approach of Coster et al. [16]. We start with an initial truncation parameter $T_0 = T_s$, and apply BKZ reduction to five different randomly permuted bases corresponding to the knapsack problem lattice. If a 0-1 solution is not found, we increase T by a factor of 10, i.e., we set $T_j = 10T_{j-1}$, and repeat the BR process. For $n = 100$, we try 10 different starting bases. We stop the runs once a 0-1 solution corresponding to a partition with discrepancy not more than the bound specified in Theorem 7. The results indicate that this truncation heuristic finds better quality partitions than the KK and CKK heuristics. Similar to the previous case, we run the CKK algorithm for 1.1× time taken by the truncation heuristic.

Similar results were obtained for BalNPP as well. The expected optimal discrepancies for these instances are much smaller than the $\langle\Delta_T\rangle$ values obtained. For example, $E(\Delta^*)$ for the $n = 100$

case is only 1.142×10^3. The goal of the truncation heuristic is only to find a good quality partition in short time, and not to search for near optimal partitions. We could obtain smaller discrepancies through truncation by starting with smaller values of T_s, but the running time will also increase in this case. One could also consider truncating an instance well into the hard phase such that the truncated instance is also in the hard phase. The value of Δ_T thus obtained in this case will be much closer to Δ^*. For values of n at which the discrete search heuristic is finds a near optimal partition efficiently, increasing the value of R does not appear to blow up the running times much – see the discussion of the case for $n = 60$ in Section 3.2.2. Hence, we may as well solve the original instance directly rather than truncating it to the easy phase. On the other hand, we are not able to handle the truncated hard phase instances for $n \geq 80$ efficiently, and hence we truncate these instances to the easy phase to obtain a good solution quickly.

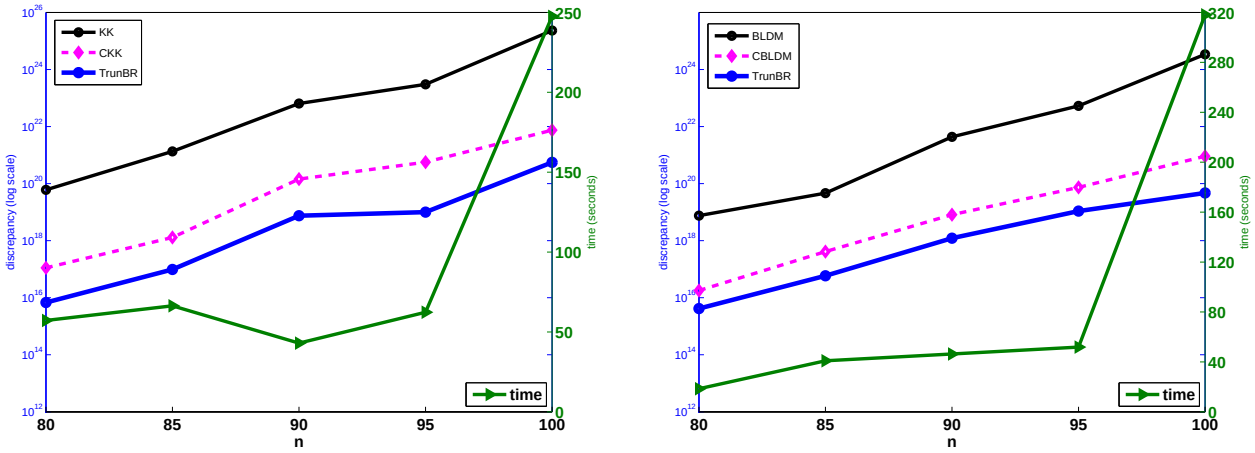

Figure 3: Comparison of TruncNPP method and KK,CKK heuristics for NPP (on left) and for BalNPP instances (on right). Discrepancies are plotted on a log scale, but notice the difference in ranges of values plotted on the left y-axis, as compared to Figures 2 and 4.

5 A Mixed Integer Program (MIP) for NPP

The discrete search and truncation algorithms do not provide any guarantees of optimality or quality of the solutions found. We try to address the question of how to provide a certificate of optimality for the solutions provided by the lattice-based approaches. A key shortcoming of the BR approach for both NPP in the hard phase and for subset sum problems is that it cannot *impose* bounds on variables. For the instances on which it fails, i.e., does not find 0-1 vectors, it typically finds relatively short vectors that correspond to solutions with a few $x_j = -1, 2$ etc. We can force such bounds through explicit optimization models. NPP is naturally modeled as a mixed integer program (MIP), as shown below.

$$\begin{array}{lrcl} \min & \Delta = 2w & & \\ \text{s.t.} & w & \geq & \sum_j a_j x_j - \beta \\ & w & \geq & -(\sum_j a_j x_j - \beta) \\ & x_j & \in & \{0,1\} \quad \forall j. \end{array} \tag{5.6}$$

The first two constraints model $w \geq |\sum_j a_j x_j - \beta|$, which implies $w \geq 0$. Given the objective function to minimize $2w$, the above inequality will be tight at the optimal solution. Integer programming can be solved in polynomial time in fixed dimensions by Lenstra's algorithm [36]. Basis reduction is one of the key steps in this algorithm. At the same time, no practical implementations of this algorithm are known. The standard techniques to solve MIPs are branch-and-bound and cutting planes [37], often referred to as branch-and-cut in conjunction. Still, we are not able to submit the NPP MIP to a state-of-the-art MIP solver such as CPLEX, due to the huge numbers involved. As far as we know, there are no arbitrary precision MIP solvers available. Hence we propose a BR-based reformulation of the NPP MIP, which can be handled by CPLEX. We write the MIP (5.6) as

$$\min \{2w \,|\, A\mathbf{x} + \mathbf{f}w \leq \mathbf{b},\ (\mathbf{x}, w) \in \mathbb{Z}^n \times \mathbb{R}\},$$

where

$$A = \begin{bmatrix} \mathbf{a} \\ -\mathbf{a} \\ -I \\ I \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} -1 \\ -1 \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \beta \\ -\beta \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix}.$$

We then form $D = \begin{bmatrix} A & \mathbf{b} \\ \mathbf{0} & M \end{bmatrix}$, with M being a large number ($> \beta$), and apply BR on D to obtain $\tilde{D}$. We must have $\tilde{D} = DV$ for some unimodular matrix V (i.e., $\det(V) = \pm 1$) , which has the form

$$V = \begin{bmatrix} U & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix}. \text{ Hence, } \tilde{D} = \begin{bmatrix} \tilde{A} & \tilde{\mathbf{b}} \\ \mathbf{0} & M \end{bmatrix}, \text{ where } \tilde{A} = AU, \quad \text{and } \tilde{\mathbf{b}} = \mathbf{b} + A\mathbf{v}.$$

Note that U is also unimodular. We then obtain a reformulation of the MIP given as

$$\min \{2w \,|\, \tilde{A}\mathbf{y} + \mathbf{f}w \leq \tilde{\mathbf{b}},\ (\mathbf{y}, w) \in \mathbb{Z}^n \times \mathbb{R}\}.$$

The equivalence of the original MIP and the reformulated MIP is shown by the one-to-one correspondence of the integer variables $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{y} = U^{-1}(\mathbf{x} + \mathbf{v}), \quad \mathbf{x} = U\mathbf{y} - \mathbf{v}.$$

For BalNPP, we add the constraint $\sum_j x_j = \lfloor n/2 \rfloor$ to the original MIP, and also add a row of ones to A and the entry $\lfloor n/2 \rfloor$ to $\mathbf{b}$. The rest of the analysis remains same.

This reformulation technique is an extension of a similar technique for integer programs termed column basis reduction (CBR) [23] to the case of MIPs. CBR generalizes previous reformulation techniques proposed by Aardal et al. for equality constrained IPs [20, 21]. In our case, the variable w will have an integer value in any optimal solution. As such, we could have treated it as the $(n+1)$-st integer variable (along with $\mathbf{x}$), and the analysis of CBR would have applied. The reformulation presented here is more general, and applies for general MIPs where we could have a set of continuous variables $\mathbf{w}$ in place of the single variable w.

Intuitively, we reduce the original MIP to an instance of *extended* CVP (lattice is $\mathbb{L}(A)$ extended by the addition of $\mathbf{f}$, and the target vector is $\mathbf{b}$). As we did before, we reduce the CVP to an instance of extended SVP using Kannan's homogenization technique, and then try to solve the latter using direct BR application to its lattice. We now go one step further, as compared to the BR heuristic. We convert the unsolved (by BR) version of SVP to an instance of MIP and solve it using branch-and-cut methods. We present results of MIP runs on NPP and BalNPP in Figure 4. Once again,

each point on the graphs represents the average best discrepancies and time over 100 trials. The version of CPLEX used is 9.0.

Notice that the MIP model is guaranteed to find the optimal discrepancy, as long as the solver finishes the run. Hence, we selected instances in the hard phase without having to choose R too large (if an instance had a perfect partition, we discarded the same). The improvements over the complete versions of differencing methods is marked, and seems to increase with larger values of n. Still, similar to the case of the BR algorithm, the run times also increase rapidly with n. For problems with $n \geq 60$, the method takes several hours to terminate, and hence is not efficient.

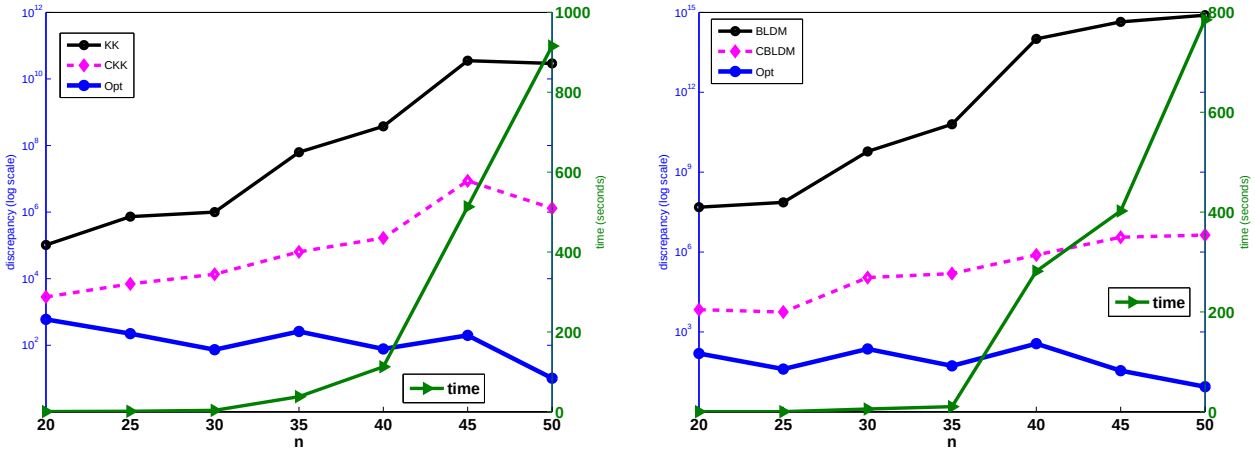

Figure 4: Comparison of MIP model and KK,CKK heuristics, for NPP (on left), and BalNPP (on right). "opt" represents the average *optimal* discrepancy, and not the expected optimal discrepancy. Discrepancies are plotted on log scale. Average run times are plotted on the right axis. We point out that the slight decrease in the discrepancies from $n = 45$ to $n = 50$ is not a *trend* – it so happens that we chose the R value for $n = 50$ slightly smaller than that used for the $n = 50$ instances reported in the BR algorithm runs (Figure 2). We still considered only instances in the hard phase (even with the smaller R value).

6 Discussion

Unlike most previous methods, the BR-based discrete search algorithm does not start with a feasible solution (corresponding to some partition), and improve on it. As such, the algorithm needs to run till the good partition is found. The The idea of truncation is very general, and can be used in conjunction with any other method that can solve, or find good approximate solutions to, the truncated instance. Similar to the BR algorithm, the truncation heuristic also adapts to more general partition problems such as unequal division and general constrained partitions. Although, we should mention that an unequal partition problem can be reduced to a standard NPP instance by adding one more number that is chosen based on the extent of inequality of shares.

At the heart of both the BR heuristic and the MIP model is a routine for basis reduction. While BKZ reduction is adequate for moderate dimensions, the efficiency of the reduction is not good as n becomes large. The lack of a true CVP oracle is the only reason why we could not develop the binary search procedure to solve instances of NPP by solving a limited number of DNPP instances.

Gama and Nguyen [24] have recently reported the state of the art of basis reduction algorithms. They are able to solve specific SVP instances for dimensions higher than 100. An adequate BR algorithm could extend the applicability of our methods to NPP instances with much larger n. Until then, the KK heuristic may still be the best option to get rough estimates of the discrepancies for NPPs with hundreds of numbers.

All methods presented in this paper are naturally extended to the case of multi-way partitions (i.e., three or more subsets). We will use a binary variable for each a_j-subset pair, and the relationships can be expressed as linear equations and inequalities. Limits on the expected values of optimal discrepancies are known in this case as well [38], and could be used just as in the discrete search algorithm.

7 Acknowledgments

We would like to thank Antonie Joux for communicating the limits of the SS and HJ algorithms for $n = 60$ instances, as well as for related insights into the behavior of these algorithms.

References

[1] C. Borgs, J. T. Chayes, S. Mertens, B. Pittel, Phase diagram for the constrained integer partitioning problem, Random Structures and Algorithms 24 (3) (2004) 315–380. doi:http://dx.doi.org/10.1002/rsa.v24:3.

[2] S. Mertens, The easiest hard problem: number partitioning, in: A. Percus, G. Istrate, C. Moore (Eds.), Computational Complexity and Statistical Physics, Oxford University Press, 2006, pp. 125–139.

[3] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W.H. Freeman and Company, 1979.

[4] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning, Operations Research 39 (3) (1991) 378–406.

[5] S. Mertens, A physicist's approach to number partitioning, Theoretical Computer Science 265 (1-2) (2001) 79–108.

[6] N. Karmarkar, R. M. Karp, The differencing method of set partitioning, Tech. Rep. 113, Computer Science Division, University of California, Berkeley (1982).

[7] B. Yakir, The differencing algorithm ldm for partitioning: a proof of a conjecture of Karmarkar and karp, Mathematics of Operations Research 21 (1) (1996) 85–99.

[8] S. Boettcher, S. Mertens, Analysis of the Karmarkar-Karp differencing algorithm, to appear in European Physics Journal B, http://arxiv.org/abs/0802.4040 (2008).

[9] N. Karmarkar, R. M. Karp, G. S. Lueker, A. M. Odlyzko, Probabilistic analysis of optimum partitioning, Journal of Applied Probability 23 (3) (1986) 626–645.

[10] G. S. Lueker, Exponentially small bounds on the expected optimum of the partition and subset sum problems, Random Structures and Algorithms 12 (1) (1998) 51–62.

[11] S. Mertens, Random costs in combinatorial optimization, Physical Review Letters 84 (6) (2000) 1347–1350.

[12] R. E. Korf, A complete anytime algorithm for number partitioning, Artificial Intelligence 106 (2) (1998) 181–203. doi:10.1016/S0004-3702(98)00086-1.

[13] C. Borgs, J. Chayes, B. Pittel, Phase transition and finite-size scaling for the integer partitioning problem, in: 33rd ACM Symposim on the Theory of Computing (STOC), Lecture notes in Computer Science, Springer-Verlag, 2001, pp. 330–336.

[14] R. Kannan, Minkowski's convex body theorem and integer programming, Mathematics of Operations Research 12 (3) (1987) 415–440.

[15] C.-P. Schnorr, H. H. Hörner, Attacking the Chor-Rivest cryptosystem by improved lattice reduction, in: Advances in Cryptology – EUROCRYPT 95, Vol. 921 of Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 1–12.

[16] M. J. Coster, A. Joux, B. A. LaMacchia, A. M. Odlyzko, C.-P. Schnorr, J. Stern, Improved low-density subset sum algorithms, Computational Complexity 2 (2) (1992) 111–128. doi:10.1007/BF01201999.

[17] D. Micciancio, S. Goldwasser, Complexity of lattice problems: A cryptographic perspective, Kluwer Academic Publishers, 2002.

[18] N. Howgrave-Graham, A. Joux, New generic algorithm for hard knapsacks, in: EUROCRYPT 2010, Lecture Notes in Computer Science, Springer-Verlag, 2010, full version available at http://eprint.iacr.org/2010/189.

[19] R. Schroeppel, A. Shamir, A $T = O(2^{n/2}), S = O(2^{n/4})$ algorithm for certain NP-complete problems, SIAM Journal on Computing 10 (3) (1981) 456–464.

[20] K. Aardal, C. A. J. Hurkens, A. K. Lenstra, Solving a system of linear Diophantine equations with lower and upper bounds on the variables, Mathematics of Operations Research 25 (3) (2000) 427–442.

[21] K. Aardal, A. K. Lenstra, Hard equality constrained integer knapsacks, Mathematics of Operations Research 29 (3) (2004) 724–738.

[22] S. Mehrotra, Z. Li, On generalized branching methods for mixed integer programming, Optimization Online http://www.optimization-online.org/DB_HTML/2005/01/1035.html.

[23] B. Krishnamoorthy, G. Pataki, Column basis reduction and decomposable knapsack problems, Discrete Optimization 6 (3) (2009) 242–270, pre-print at http://www.optimization-online.org/DB_HTML/2007/06/1701.html. doi:10.1016/j.disopt.2009.01.003.

[24] N. Gama, P. Q. Nguyen, Predicting lattice reduction, in: Advances in Cryptology – EUROCRYPT 2008, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 31–51.

[25] P. Q. Nguyen, T. Vidick, Sieve algorithms for the shortest vector problem are practical, Journal of Mathematical CryptologyTo appear.

[26] A. K. Lenstra, H. W. Lenstra, Jr., L. Lovász, Factoring polynomials with rational coefficients, Mathematische Annalen 261 (1982) 515–534.

[27] C.-P. Schnorr, A hierarchy of polynomial time lattice basis reduction algorithms, Theoretical Computer Science 53 (1987) 201–225.

[28] B. A. LaMacchia, Basis reduction algorithms and subset sum problems, sM Thesis, Dept. of Elec. Engg. and Comp. Sci., Massachusetts Institute of Technology. (1991).

[29] C.-P. Schnorr, M. Euchner, Lattice basis reduction: improved practical algorithms and solving subset sum problems, Mathematical Programming 66 (2) (1994) 181–199.

[30] D. Micciancio, The hardness of the closest vector problem with preprocessing, IEEE Transactions on Information Theory 47 (3) (2001) 1212–1215. doi:10.1109/18.915688.

[31] D. E. Knuth, The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[32] D. Micciancio, P. Voulgaris, A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations, in: 42nd ACM Symposim on the Theory of Computing (STOC), 2010.

[33] J. C. Lagarias, A. M. Odlyzko, Solving low-density subset sum problems, Journal of ACM 32 (1985) 229–246.

[34] V. Shoup, NTL: A Number Theory Library, http://www.shoup.net (1990).

[35] S. Mertens, A complete anytime algorithm for balanced number partitioning, http://arXiv.org/abs/cs.DS/9903011 (1999).

[36] H. W. Lenstra, Jr., Integer programming with a fixed number of variables, Mathematics of Operations Research 8 (1983) 538–548.

[37] L. A. Wolsey, Integer Programming, Wiley-Interscience, 1998.

[38] H. Bauke, S. Mertens, A. Engel, Phase transition in multiprocessor scheduling, Physical Review Letters 90 (15) (2003) 158701. doi:10.1103/PhysRevLett.90.158701.

Appendix: Details of Computation

We present details of the calculations represented by the graphs in Figures 2, 3, 4 in the following tables. Each point on the plots corresponds to the average value of 100 trials. The reduction parameter $1/4 < \nu < 1$ in the BKZ reduction [29] routine is fixed at 0.999 for all our calculations.

For the discrete search heuristic presented in Figure 2 and in Tables 1 and 2, the BlockSize parameter in BKZ reduction was set as half of the number of columns, i.e., as $\lfloor (n+1)/2 \rfloor$ for instances of size n. For values of n up to 75, the choice of half blocksize performed well. While increasing the blocksize further only increased the computational time by up to a factor of 2, it did not improve the solution quality by much. For the truncated problems with $80 \leq n \leq 100$, increasing the blocksize beyond half did help in improving the quality of solutions obtained. The choices of blocksize for each n are listed in Tables 3 and 4.

Table 1: Details of the data plotted for NPP in Figure 2 (left figure). R is given in $\log_{10}$ units – the R value for $n = 45$ is 10^{15} for instance. $\langle \#\text{BR} \rangle$ gives the average number of BR runs, i.e., the number of values tried for the parameter c in the discrete search algorithm. Time is in seconds.

n	R	$\langle \#\text{BR} \rangle$	$\langle \text{Time} \rangle$	$\langle \Delta_{KK} \rangle$	$\langle \Delta_{CKK} \rangle$	$\langle \Delta \rangle$	$E(\Delta^*)$
30	11	4.6	0.27	1.674e+07	6.186e+05	737.22	738.23
35	12	5.7	0.76	8.940e+07	1.204e+06	261.72	249.18
40	13	8.3	2.37	4.699e+08	3.268e+06	93.24	83.25
45	15	7.9	5.49	2.555e+10	7.923e+07	279.14	275.92
55	17	15.3	57.31	1.078e+11	1.754e+08	37.11	29.79
50	16	11.5	17.62	9.328e+11	5.063e+08	104.60	90.89
60	19	15.9	129.3	5.580e+13	1.415e+10	117.60	97.23
65	20	18.1	243.02	2.081e+14	9.701e+10	51.50	31.63
70	22	23.6	2683.67	3.227e+16	6.736e+11	150.47	102.56
75	23	65.0	4701.93	1.850e+17	3.062e+12	334.40	33.17

Table 2: Details of the data plotted for BalNPP in Figure 2 (right figure). R is given in $\log_{10}$ units. $\langle \#\text{BR} \rangle$ gives the average number of BR runs, and Time is in seconds.

n	R	$\langle \#\text{BR} \rangle$	$\langle \text{Time} \rangle$	$\langle \Delta_{BLDM} \rangle$	$\langle \Delta_{CBLDM} \rangle$	$\langle \Delta \rangle$	$E(\Delta^*)$
30	10	3.6	0.15	6.533e+10	7.452e+05	284.99	52.20
35	11	5.1	0.53	3.289e+11	7.562e+05	84.68	17.62
40	13	4.5	0.98	1.142e+14	8.442e+07	231	58.86
45	14	7.4	3.75	5.594e+14	8.572e+07	99.04	19.51
50	15	7.8	8.76	1.291e+16	4.272e+08	46.76	6.43
55	16	19.7	55.31	7.299e+16	3.217e+08	33.05	2.11
60	18	7.8	60.08	1.655e+19	5.022e+10	42.2	6.88
65	20	11.2	157.71	8.024e+20	4.366e+11	95.2	22.36
70	21	21.9	576.86	2.039e+22	3.638e+12	82.15	7.25
75	23	17.2	949.96	2.172e+24	9.221e+13	206.4	23.46

Table 3: Details of the data plotted for TruncNPP for NPP instances in Figure 3 (plot on the left). Blk gives the BlockSize parameter used in the Block Korkine-Zolotarev (BKZ) [34] BR routine for each n. t_s gives the starting truncation level in $\log_{10}$ units, i.e., $T_{\text{start}} = 10^{t_s}$. $\langle t \rangle$ gives the average truncation parameter over all trials in $\log_{10}$ units.

n	R	Blk	t_s	$\langle t \rangle$	$\langle \#\text{BR} \rangle$	$\langle \text{Time} \rangle$	$\langle \Delta_{KK} \rangle$	$\langle \Delta_{CKK} \rangle$	$\langle \Delta \rangle$
80	26	70	13	14.9	11.8	57.17	6.051e+19	1.111e+17	6.767e+15
85	27	70	15	16.0	12.1	66.38	1.347e+21	1.295e+18	9.804e+16
90	29	80	17	17.9	13.6	43.01	6.425e+22	1.439e+20	7.488e+18
95	30	80	17	18.3	8.8	62.34	3.020e+23	5.661e+20	1.001e+19
100	32	90	18	19.7	20.1	247.54	2.327e+25	7.454e+21	5.568e+20

Table 4: Details of the data plotted for TruncNPP for BalNPP instances in Figure 3 (plot on the right). Blk gives the BlockSize parameter used in the Block Korkine-Zolotarev (BKZ) [34] BR routine for each n. t_s gives the starting truncation level in $\log_{10}$ units, i.e., $T_{\text{start}} = 10^{t_s}$. $\langle t \rangle$ gives the average truncation parameter over all trials in $\log_{10}$ units.

n	R	Blk	t_s	$\langle t \rangle$	$\langle$#BR$\rangle$	$\langle$Time$\rangle$	$\langle \Delta_{BLDM} \rangle$	$\langle \Delta_{CBLDM} \rangle$	$\langle \Delta \rangle$
80	25	70	14	14.9	6.15	18.52	7.525e+18	1.786e+16	4.177e+15
85	26	70	14	15.5	9.55	40.91	4.627e+19	4.118e+17	5.883e+16
90	28	80	16	17.2	8.3	46.34	4.364e+21	8.163e+18	1.222e+18
95	29	80	17	18.5	9.05	51.87	5.300e+22	7.387e+19	1.085e+19
100	31	90	17	18.6	23.75	318.07	3.412e+24	9.105e+20	4.716e+19

Table 5: Details of the data plotted for the MIP approach to NPP in Figure 4 (left figure). R is given in $\log_{10}$ units. $\langle$#BB$\rangle$ gives the average number of branch-and-bound nodes evaluated by CPLEX 9.0 to solve each instance. Time is in seconds. Since the MIPs were solved to optimality, we report the average optimal discrepancy under $\langle \Delta^* \rangle$.

n	R	$\langle$#BB$\rangle$	$\langle$Time$\rangle$	$\langle \Delta_{KK} \rangle$	$\langle \Delta_{CKK} \rangle$	$\langle \Delta^* \rangle$
20	8	3305	0.61	1.032e+05	2.822e+03	599.64
25	9	5940	1.36	7.317e+05	6.990e+03	222.18
30	10	13340	3.55	1.001e+06	1.358e+04	73.38
35	12	115905	37.95	6.296e+07	6.402e+04	258.76
40	13	289124	112.72	3.759e+08	1.664e+05	77.26
45	15	2291299	513.20	3.548e+10	8.715e+06	198.25
50	15	3423102	915.60	2.929e+10	1.310e+06	10.20

Table 6: Details of the data plotted for the MIP approach to BalNPP in Figure 4 (right figure). R is given in $\log_{10}$ units. $\langle$#BB$\rangle$ gives the average number of branch-and-bound nodes evaluated by CPLEX 9.0 to solve each instance. Time is in seconds. Since the MIPs were solved to optimality, we report the average optimal discrepancy under $\langle \Delta^* \rangle$.

n	R	$\langle$#BB$\rangle$	$\langle$Time$\rangle$	$\langle \Delta_{BLDM} \rangle$	$\langle \Delta_{CBLDM} \rangle$	$\langle \Delta^* \rangle$
20	7	1070	0.20	4.816e+07	6.870e+03	155.43
25	8	1379	0.33	7.317e+07	5.583e+03	40.31
30	10	20847	5.56	5.969e+09	1.105e+05	229.69
35	11	32214	10.23	6.296e+10	1.554e+05	53.15
40	13	790583	281.66	1.008e+14	7.751e+05	362.32
45	14	465825	402.20	4.402e+14	3.595e+06	35.14
50	14	1104529	785.21	7.871e+14	4.335e+06	8.75