A Relaxed FPTAS for Chance-Constrained Knapsack

Galia Shabtai
School of Electrical Engineering, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel
galiashabtai@gmail.com

Danny Raz
Faculty of Computer Science, The Technion, Haifa 32000, Israel
danny@cs.technion.ac.il

Yuval Shavitt
School of Electrical Engineering, Tel Aviv University, Ramat Aviv, Tel Aviv 69978, Israel
shavitt@eng.tau.ac.il

Abstract

The stochastic knapsack problem is a stochastic version of the well known deterministic knapsack problem, in which some of the input values are random variables. There are several variants of the stochastic problem. In this paper we concentrate on the *chance-constrained* variant, where item values are deterministic and item sizes are stochastic. The goal is to find a maximum value allocation subject to the constraint that the overflow probability is at most a given value. Previous work showed a PTAS for the problem for various distributions (Poisson, Exponential, Bernoulli and Normal). Some strictly respect the constraint and some relax the constraint by a factor of $(1+\epsilon)$. All algorithms use $\Omega(n^{1/\epsilon})$ time. A very recent work showed a "*almost* FPTAS" algorithm for Bernoulli distributions with $O(poly(n) \cdot quasipoly(1/\epsilon))$ time.

In this paper we present a FPTAS for normal distributions with a solution that satisfies the chance constraint in a relaxed sense. The normal distribution is particularly important, because by the Berry-Esseen theorem, an algorithm solving the normal distribution also solves, under mild conditions, arbitrary independent distributions. To the best of our knowledge, this is the first (relaxed or non-relaxed) FPTAS for the problem. In fact, our algorithm runs in $poly(\frac{n}{\epsilon})$ time. We achieve the FPTAS by a delicate combination of previous techniques plus a new alternative solution to the non-heavy elements that is based on a non-convex program with a simple structure and an $O(n^2 \log \frac{n}{\epsilon})$ running time. We believe this part is also interesting on its own right.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis, Theory of computation → Stochastic approximation, Theory of computation → Discrete optimization, Theory of computation → Nonconvex optimization

Keywords and phrases Stochastic knapsack, Chance constraint, Approximation algorithms, Combinatorial optimization

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.72

1 Introduction

Stochastic optimization has been studied by a large community since the 1950s. In a stochastic problem the input contains information about *distributions* rather than concrete values, and the goal is to provide a solution that works well on instances drawn according to the input distributions. For example, one might want to optimize the expected value of certain objective function for inputs drawn from the given input distributions.

An important special case of stochastic optimization is *chance-constrained* optimization where we want to optimize a target function under the restriction that the probability we violate the constraints is at most some given threshold p. For example, Kleinberg, Rabani and Tardos [6], who were among the first to study approximation algorithms for stochastic problems, studied a chance-constrained version of the stochastic knapsack problem, `CCKnapsack`, in the context of bursty connections. In their case the input is information about n items. Item i has value val_i and its size is Bernoulli distributed, i.e., with probability q_i it has size s_i and with probability $1 - q_i$ it has size 0. The distributions of the n items are independent. The input also includes a value p and the knapsack capacity c. The goal is to choose a subset of the n items that maximizes the total value subject to the constraint that the overflow probability is at most p. Kleinberg et. al. provide a close to linear time $O(\log \frac{1}{p})$-approximation algorithm, by showing a simple reduction to the deterministic case.

Goel and Indyk [4] studied `CCKnapsack` for several other distributions: Poisson, Exponential and Bernoulli.

- For Poisson they gave a PTAS (Polynomial Time Approximation Scheme). More precisely, given $\epsilon > 0$ the algorithm runs in time $n^{O(1/\epsilon)}$ time and outputs a feasible solution (i.e., a solution where the overflow probability is at most p) with value at least $(1 - \epsilon)P^*$, where P^* is the optimal feasible value.
- For the Exponential distributions they obtained a *relaxed* PTAS, namely, they output an objective value that is no worse than the optimum, but the solution violates the knapsack size and the overflow probability by a factor of $(1 + \epsilon)$.
- For Bernoulli distribution the situation is even worse and they obtain a *relaxed* QPTAS (Quasi-Polynomial Time Approximation Scheme) algorithm which relaxes the constrains by a factor of $(1 + \epsilon)$, and for a given constant ϵ runs in quasi-polynomial time in n.

Goyal and Ravi [5] present a PTAS for `CCKnapsack` when item sizes are normally distributed. Their algorithm does not relax the overflow probability constraint nor the capacity constraint. However, it does not give a FPTAS, as the running time of the algorithm is $\Omega(n^{1/\epsilon})$. Later, Bhalgat, Goel and Khanna [1] obtained a PTAS which relaxes both the overflow probability constraint and the capacity constraint and works for any random variable. In a recent work, De [3] showed a "(nearly) FPTAS" for the `CCKnapsack` with Bernoulli distributions and quasi-FPTAS for k-supported random variables, i.e. when all item sizes are supported on a common set of constant size. De [3] also showed a PTAS for hypercontractive random variables, i.e. random variables whose second and fourth moments are within constant factors of each other. Poisson, Gaussian and Exponential random variables are hypercontractive random variables. All three algorithms presented by De [3] relax the overflow probability by an additive ϵ. Table 1 summarizes the above mentioned previous work results.

Goyal and Ravi [5] study the normal distribution case. The normal distribution is particularly interesting since by the central limit theorem the sum of n independent distributions converges to a normal distribution and the Berry Essen theorem gives a concrete bound on the rate of convergence as a function of the first three moments. It is shown in [9] and [8], in a slightly different setting, that an algorithm that solves a chance constrained stochastic problem also works for any n independent distributions, as long as the input distributions respect some mild conditions (e.g., their third moments are reasonable), and this is also true for `CCKnapsack`.

The special case where there are no *heavy* items, i.e., items whose value is more than ϵ fraction of the optimal value P^*, is particularly interesting, because this is the usual setting for many cloud problems, where there are many services and no single service alone dominates resource demand. In this special case Goyal and Ravi's algorithm is much faster and runs in $poly(n)$ time (with no dependence on ϵ).

Table 1 Known Results for `CCKnapsack`.

Reference	Distribution	Relaxed Overflow Probability	Relaxed Knapsack Capacity	Approximation Scheme
Kleinberg et al [6]	Bernoulli	no	no	$O(log\frac{1}{p})P^*$ polynomial time
Goel and Indyk [4]	Poisson	no	no	PTAS
	Exponential	yes	yes	PTAS
	Bernoulli	yes	yes	QPTAS
Goyal and Ravi [5]	Normal	no	no	PTAS
Bhalgat et al [1]	any	yes	yes	PTAS
De [3]	Bernoulli	yes	no	(nearly) FPTAS
	k-supported	yes	no	quasi-FPTAS
	hypercontractive (Poisson,Gaussian, Exponential, ...)	yes	no	PTAS
Current work	**Normal**	**no**	**yes**	**FPTAS**

We mention that other variants of the problem were studied, e.g., the dynamic knapsack problem [7, 2] where decisions are adaptive and each size is revealed (or realized) only after the decision maker attempts to insert it.

1.1 Previous techniques

Kleinberg et al. [6] show a simple reduction to the deterministic case, by calculating an *effective bandwidth* value for each item, and then running a greedy algorithm on these *deterministic* values.

Goel and Indyk [4] proceeded by separating the items to big and small based on whether their value is "large" (which should be appropriately defined) or "small". On small variables a greedy fractional algorithm is used, and it is easy to see that it has at most one non-integral value. This value is then dropped, and the loss in value is small, because the dropped item is "small". For "large" items, all candidate sets of large items (and there are at least $2^{1/\epsilon}$ such candidate sets) are checked.

Goyal and Ravi [5] replaced the greedy algorithm with a parametric LP program and showed that the resulting fractional solution has at most *two* non-integral values. They exploit that for a rounding algorithm that essentially solves correctly (with a small approximation error) the non-heavy elements. For the heavy elements, Goyal and Ravi again try all $n^{1/\epsilon}$ subsets of heavy elements (as there are at most $1/\epsilon$ elements with value ϵP^*).

Of course, there are more technical issues to be handled. For example, to do the partition to large and small items correctly one needs to know (approximately) the optimal value P^*. The solution is to try all approximations to P^* from the set $\{P_{min}, , \ldots, P_{min}(1+\epsilon)^i, \ldots\}$. There must be one P_i in the set such that $P_i \approx P^*$, and the number of such P_i is linear in the input length.

1.2 Our contribution

The results in this paper are two-fold:

- First, we simplify the solution of the light elements (items whose value is at most ϵ fraction of the optimal value P^*) and show a simple non-convex program that has an efficient almost integral solution, and,
- Second, we use it to give a relaxed FPTAS (i.e., an algorithm with running time $poly(\frac{n}{\epsilon})$ rather than $n^{1/\epsilon}$). We note that this is the first FPTAS algorithm for `CCKnapsack`, relaxed or not.

We now explain more about these two contributions. First, using a technique from Nikolova [8] we translate the problem on the light elements, to a concrete non-linear (and non-convex) program on $\mathbf{R}^2$. The program is, in fact, a quasi-concave minimization problem, and is minimized on one of the vertices of the polygon of possible solutions. We study this polygon and prove that:

- The polygon has at most n^2 vertices,
- These vertices can be easily enumerated, and,
- Each vertex represents an almost integral solution, with at most one non-integral item.

While the base approach is taken from Nikolova [8] the situation here is very different. In [8] the polygon has many (super polynomial) vertices and also it is NP hard to enumerate all the vertices of the polygon. Accordingly, we deviate from the approach taken in [8] and in this paper we use geometric intuition that completely unravel the nature of the polygon in our case. We use the above three properties to construct an efficient algorithm solving `CCKnapsack` when all the items are not heavy. Specifically, we show:

▶ **Theorem 1.** *There exists an algorithm for `CCKnapsack` over normal distributions such that if the value of each element is at most ϵP^*, where P^* is the optimal feasible value, then the algorithm outputs a feasible integral solution with value at least $(1-\epsilon)P^*$. The running time of the algorithm is $O(n^2 \log \frac{n}{\epsilon})$.*

We remark that Goyal and Ravi's algorithm [5] also gives a $poly(n)$ algorithm for the case in which all items are not ϵ-heavy, but our solution is simpler and faster, and we hope that it can also be used in practice .

We now move to our second contribution. As explained above, Goel and Indyk [4] showed a relaxed PTAS for `CCKnapsack` over several distributions and Goyal and Ravi [5] showed a strict PTAS for the normal distribution. The above algorithms have running time $\Omega(n^{1/\epsilon})$ which indeed allows a PTAS, but is prohibitively large. It is a natural intriguing problem to improve this situation and find a FPTAS whose running time is $poly(n, f(\epsilon))$ for some function f. We show such a result with running time $poly(\frac{n}{\epsilon})$. We prove:

▶ **Theorem 2.** *There exists an algorithm solving `CCKnapsack` over normal distributions that ϵ approximates the optimum in the relaxed sense, i.e., given an input, it finds a solution such that the overflow probability with a slightly larger capacity $(1+\epsilon)C$ is at most the specified overflow probability. The running time of the algorithm is $poly(\frac{n}{\epsilon})$.*

The idea is quite natural. We have two basic algorithms for `CCKnapsack`:

- The algorithm of Theorem 1 that approximates the optimal *integral* solution for non-heavy elements.
- An exact dynamic programming algorithm that finds an integral solution for `Knapsack` in time polynomial in the number of partial sums. This algorithm can be easily extended to `CCKnapsack`.

Suppose we know the optimal integral value P. If we divide all items to *light* and *heavy* according to whether their value is smaller than ϵP or higher than it, and if we want an ϵ approximation of the optimal value, we are allowed to lose any constant number of light items, but we are not allowed to lose any heavy item. This leads to the following strategy: On the light items we run the algorithm of Theorem 1. On the Heavy items we cannot miss any single item but we do not mind taking an ϵ multiplicative approximation. Hence, we round the input values of the heavy items, and run the exact algorithm (that does not lose even a single item) on the rounded heavy items. As there are only few possibilities, the number of partial sums is small, and that part can be efficiently implemented.

There are many technical challenges in implementing the above idea, and our solution is a delicate combination of previous techniques: the multiplicative incremental guessing of parameters, truncation of heavy elements, the dynamic programming for `Knapsack` and its extension to `CCKnapsack` and our new reduction to the non-convex problem and its simple structure.

The paper is organized as follows. In Section 2 we show a simple *fractional* solution to `CCKnapsack`. In Section 3 we study the non-convex program and the polygon of possible solutions. In [10] we generalize the dynamic programming algorithm of `Knapsack` to `CCKnapsack`, and In Section 4 we present our FPTAS with relaxed constraints.

2 The Fractional Chance Constrained Knapsack Problem

The Chance Constrained Knapsack Problem for Normal distributions is defined as follows:

`CCKnapsack`: Chance Constrained Knapsack

- **Input:** The input to the problem consists of:
  - C specifying the knapsack capacity,
  - n specifying the number of items available for inclusion in the knapsack,
  - ζ specifying a bound on overflow probability,
  - ϵ - accuracy parameter.
  - The size of item i, denoted by $X^{(i)}$, is normally distributed with mean $\mu^{(i)}$, and variance $V^{(i)}$ that are given as input. Set $Q^{(i)} = (\mu^{(i)}, V^{(i)})$. The distributions $X^{(i)}$ are independent.
  - Also, for each item i we are given the value $p^{(i)} > 0$ of that item.

  A solution $\alpha = (\alpha_i, \ldots, \alpha_n) \in \{0,1\}^n$ is *feasible* if $Pr[\sum_{i=1}^{n} \alpha_i X^{(i)} > C] \leq \zeta$.
- **Output:** The output is a vector $\alpha_{out} = (\alpha_i, \ldots, \alpha_n)$ such that:
  - Integrality constraint: $\alpha_i \in \{0,1\}$, $\alpha_i = 1$ if item i is selected to be included in the knapsack and $\alpha_i = 0$ otherwise.
  - The solution is feasible, and
  - Let $P_{out} = \sum_i \alpha_i p^{(i)}$, $P_{OPT} = \max\left\{\sum_{i=1}^{n} \alpha_i p^{(i)} \mid \alpha_i \in \{0,1\}, \alpha \text{ is feasible}\right\}$. We require that $|P_{out} - P_{OPT}| \leq \epsilon$.

`CCKnapsack` is not linear as the overflow probability is not linear. Moreover, the exact problem is clearly NP-hard since its deterministic version, in which each item $X^{(i)}$ takes a single value with probability 1, is the knapsack problem. Therefore, we only ask for an efficient approximation to the problem.

A *fractional* solution is a feasible vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ with $\alpha_i \in [0,1]$, dropping the integrality constraint.

▶ **Theorem 3.** *There exists an algorithm that ϵ-approximates the* `CCKnapsack` *problem with running time $O(n^2 \log \frac{n}{\epsilon})$, where n is the number of elements. Furthermore, in the fractional solution found, there is at most one fractional item.*

We now explain our approach for solving the problem. In the `CCKnapsack` problem the item sizes are independent and normally distributed. Suppose α is a fractional solution. The size of the solution α is a random variable $X_\alpha = \sum \alpha_i X^{(i)}$ and is normally distributed with mean $\mu_\alpha = \sum_{i=1}^n \alpha_i \mu^{(i)}$ and variance $V_\alpha = \sum_{i=1}^n \alpha_i V^{(i)}$. Also, μ_α and V_α determine the overflow probability (because the distribution is Normal and is determined by the mean and variance). Hence, we can represent each fractional solution α by the point $(\mu_\alpha, V_\alpha) \in \mathbf{R}^2$ and define the following polygon $\Lambda \subseteq \mathbf{R}^2$ of fractional solutions: $\Lambda = \left\{Q = \sum_{i=1}^n \alpha_i Q^{(i)} \in \mathbf{R}^2 \mid 0 \le \alpha_i \le 1\right\}$. The algorithm performs a binary search to find an approximate maximum value with overflow probability at most ζ. Each step in the binary search determines whether there exists a feasible solution with a given value P. This translates to the question whether the polygon

$$\Lambda_P \quad = \quad \left\{ Q = \sum_{i=1}^n \alpha_i Q^{(i)} \in \mathbf{R}^2 \mid 0 \le \alpha_i \le 1,\ \sum_{i=1}^n \alpha_i p^{(i)} = P \right\}. \tag{1}$$

contains a point with overflow probability at most ζ. Equivalently, the problem is whether the minimal overflow probability over the points in the polygon Λ_P is at most ζ.

We will see (in Section 3) that the problem of minimizing the overflow probability over the polygon is a quasi-concave minimization problem over a convex body and is seemingly hard. The novelty of the algorithm lies in efficiently solving this problem. This is done by showing (in Section 3) that:

▶ **Lemma 4.** *Fix P and look at the polygon Λ_P.*

1. *The minimum overflow probability over points in Λ_P is obtained at a vertex of Λ_P.*
2. *The polygon Λ_P has at most $n^2 - n$ vertices.*
3. *There exists an algorithm FPBoundary that outputs all vertices in time $O(n^2 \log n)$.*

Having that, a simple binary search (over the possible values of P) gives Theorem 3 and we give it (along with the correctness proof) in [10].

3 The boundary of the polygon Λ_P.

The overflow probability of a solution $\alpha = (\alpha_1, \ldots, \alpha_n)$, denoted $OFP(\alpha)$, is,

$$OFP(\alpha) = Pr[\sum_{i=1}^n \alpha_i X^{(i)} > C] = \frac{1}{\sqrt{2\pi V_\alpha}} \int_C^\infty e^{-\frac{(x-\mu_\alpha)^2}{2V_\alpha}} dx = 1 - \Phi(\frac{C-\mu_\alpha}{\sqrt{V_\alpha}}),$$

where Φ is the cumulative distribution function of the standard Normal distribution. Nikolova shows in [8] that when $C - \mu_\alpha > 0$, OFP is a quasi-concave function (on an n dimensional space $(\alpha_1, \ldots, \alpha_n)$). Also, Nikoova noticed that as OFP depends only on the total mean and variance, when we project the problem to two dimensions (as we did in the previous section) OFP remains quasi-concave. Hence, OFP gets a minimum value over Λ_P on a vertex of Λ_P.

In [10] we prove:

▶ **Theorem 5.** *For every $I_1 = \sum_{i=1}^n \alpha_i Q^{(i)}$ and $I_2 = \sum_{i=1}^n \beta_i Q^{(i)}$ that are adjacent vertices of the polygon Λ_P, the tuples $\alpha = (\alpha_1, \ldots, \alpha_n)$ and $\beta = (\beta_1, \ldots, \beta_n)$ differ in exactly two elements. Let k and ℓ be the indices of these two elements. We call the items k, ℓ the* active *items in vertex I_1.*

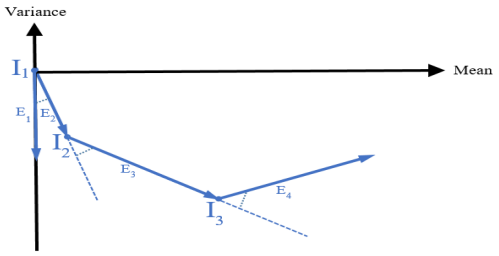

Figure 1 I_1, I_2 and I_3 are vertices of the polygon. The angle the edge (I_2, I_3) has with (I_1, I_2) is the smallest among all edges (I_2, X) for X in the polygon.

3.1 Enumerating the polygon vertices

In this section we first show an algorithm PBoundary running in $O(n^4)$ time, and then we introduce a faster algorithm FPBoundary solving the problem in $(n^2 \log n)$ time.

Algorithm PBoundary

1. The algorithm first calculates the leftmost vertex, I_1, of the polygon Λ_P on the mean-variance plane. The point I_1 has the minimum mean value among all points in Λ_P and therefore it is the leftmost point.
   To find I_1, sort the input in increasing order of mean to value ratio and re-index it, such that $\frac{\mu^{(1)}}{p^{(1)}} \leq \ldots \leq \frac{\mu^{(n)}}{p^{(n)}}$. Denote $I_1 = \sum_{i=1}^{n} \alpha_i Q^{(i)}$. Find the smallest k such that $\sum_{i=1}^{k} p^{(i)} \geq P$ and set $\alpha_i = 1$ for all $i < k$ and $\alpha_i = 0$ for all $i > k$. Set α_k such that $\sum_{i=1}^{n} \alpha_i p^{(i)} = P$. Also, set $E_1 = (0, -1)$.
2. Suppose we have calculated all points $I_1, I_2, \ldots, I_t$ and all vectors $E_1, E_2, \ldots, E_t$. We now calculate the point I_{t+1} and the vector E_{t+1}. Denote $I_t = \sum_{i=1}^{n} \alpha_i Q^{(i)}$ and $I_{t+1} = \sum_{i=1}^{n} \beta_i Q^{(i)}$.
   a. **Find direction:** Find a pair of items $(k, \ell) \in [n] \times [n]$, $k \neq \ell$, such that swapping items k and ℓ creates the smallest angle with E_t. Namely, $\alpha_k > 0$, $\alpha_\ell < 1$ (meaning that we can take more of item ℓ and less of item k) and the angle between E_t and $Q^{(\ell)} - Q^{(k)}$ is smallest. Also, set $E_{t+1} = Q^{(\ell)} - Q^{(k)}$.
   b. **Edge length:** Set $\beta_i = \alpha_i$, $\forall i \notin \{k, \ell\}$. If $\alpha_k p^{(k)} \leq (1 - \alpha_\ell) p^{(\ell)}$, let $\beta_k = 0$ and set β_ℓ such that $(\beta_\ell - \alpha_\ell) p^{(\ell)} = \alpha_k p^{(k)}$. Otherwise, let $\beta_\ell = 1$ and set β_k such that $(\alpha_k - \beta_k) p^{(k)} = (1 - \alpha_\ell) p^{(\ell)}$.
   The algorithm stops when $I_{t+1} = I_1$.

▶ **Claim 6.** *The list of points $I_1, I_2, \ldots$ found by PBoundary is the list of all polygon vertices.*

Proof. Step (1) finds the leftmost point I_1 in Λ_P which, in particular, is a vertex of Λ_P. We now want to find the next adjacent vertex, going counterclockwise. According to Theorem 5 two adjacent vertices differ in exactly two items. Among all possible pairs of different items, Step (2a) chooses the pair of indices $(k, \ell) \in [n] \times [n]$ such that inserting item ℓ and removing item k creates the smallest angle with the vector $E_1 = (0, -1)$ which is parallel to the variance axis. This sets a direction that equals the direction of the edge leaving I_1 in Λ_P. Step (2b) sets $(\beta_1, \ldots, \beta_n) \in [0, 1]^n$ such that we go in this direction as far as we can while staying in Λ_P: it either sets β_k to 0 or β_ℓ to 1, which ensure that we go on the chosen direction as far as we can. We therefore stop on the next vertex I_2. We then set $E_2 = I_2 - I_1$. Notice that E_2 is the *direction* of the edge (I_1, I_2). See Figure 1.

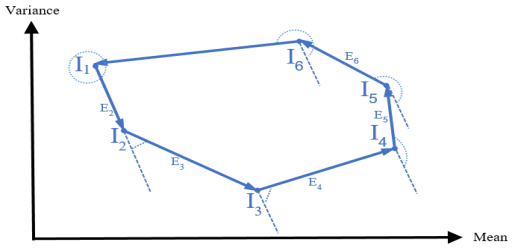

Figure 2 The angle between (I_5, I_6) and (I_1, I_2) is larger than angle between (I_4, I_5) and (I_1, I_2).

Similarly, suppose we have found the first t vertices $I_1, \ldots, I_t$ for $t > 1$, and (I_{t-1}, I_t) is the last edge found on the boundary so far with direction $E_t = I_t - I_{t-1}$. Again, by Theorem 5 two adjacent vertices differ in exactly two items. Step (2a) chooses a pair of items $(k, \ell) \in [n] \times [n]$ such that inserting ℓ and removing k creates the smallest angle with E_t. This sets a direction that equals the direction of the edge leaving I_t in Λ_P. Again, step (2b) sets $(\beta_1, \ldots, \beta_n) \in [0, 1]^n$ such that we go on the chosen direction as far as we can while staying in Λ_P and therefore stops on a vertex. See Figure 1. The algorithm stops when it reaches back to the initial vertex I_1. ◀

▶ **Corollary 7.** *The polygon Λ_P has at most $n(n-1)$ vertices.*

Proof. Let $I_1, \ldots, I_m$ be the vertices in the order found by algorithm PBoundary. Notice that the direction of (I_j, I_{j+1}) is $Q^{(\ell)} - Q^{(k)}$ where (k, ℓ) are the active items at vertex I_j, and does not depend on the edge length. Thus, there are at most $n(n-1)$ possible edge directions. As all edges of a convex body in $\mathbf{R}^2$ must have different (directed) directions, we see that the number of vertices is at most $n(n-1)$. ◀

Algorithm PBoundary takes $O(n^4)$ time, because we have seen that the number of vertices of Λ_P is at most n^2 and if we have found I_t, to find the next vertex we go over n^2 possible directions. Altogether, the running time is $O(n^4)$. Algorithm PBoundary goes over all n^2 possible directions at each step t. However, in a convex body that is contained in $\mathbf{R}^2$ the edges have a natural ordering. Suppose the polygon $\Lambda_P \subseteq \mathbf{R}^2$ has m vertices. Make any vertex a distinguished one and call it I_1. Suppose the other vertices of the polygon are ordered by $I_2, \ldots, I_m$, i.e., $(I_j, I_{j+1 \mod m})$ is an edge of the Λ_P. Let $angle_t$ be the angle between the vectors $I_{t+1 \mod m} - I_t$ and $I_2 - I_1$. Then, the angles are monotonically increasing in t until we complete a whole circle and get the zero angle again. Notice that the orientation of a vector is important and the angle between a vector v and $-v$ is π. See Figure 2.

With that it becomes clear that we do not need to consider all the n^2 directions at each time step t. Instead we first do a pre-processing step in which we calculate all the n^2 directions and sort them by the angle they make with the vector $(0, -1)$ in increasing order. Then, at step t we never look at a direction that we already passed (because the angles are monotonically increasing) and we start the search right after the last entry we have reached in the table. Altogether, our total running time is the table size, reducing the running time from n^4 to about n^2. We give algorithm FPBoundary and its analysis in [10].

3.2 The vertices of polygon Λ_P are almost integral

▶ **Theorem 8.** *Let I be a vertex of Λ_P. Then there is a way to write $I = \sum \alpha_i Q^{(i)}$ with $\alpha = (\alpha_1, \ldots, \alpha_n) \in [0, 1]^n$ such that there is at most one element $k \in [n]$ in which $\alpha_k \notin \{0, 1\}$.*

Proof. Assume that there are at least two elements, k and l, in which $\alpha_k, \alpha_l \notin \{0,1\}$. Let $\delta_k = \frac{1}{2}min\{\alpha_k, 1-\alpha_k\}$ and $\delta_l = \frac{1}{2}min\{\alpha_l, 1-\alpha_l\}$. If $\delta_k p^{(k)} < \delta_l p^{(\ell)}$, decrease δ_l to get $\delta_k p^{(k)} = \delta_l p^{(\ell)}$, otherwise, decrease δ_k to reach this equality. It is easy to see that both points $I - \delta_k Q^{(k)} + \delta_l Q^{(\ell)}$ and $I + \delta_k Q^{(k)} - \delta_l Q^{(\ell)}$ are in Λ_P, and that the point I is the mid point of the line connecting them. This contradicts the fact the that I is a vertex. Hence, there is at most one element $k \in [n]$ in which $\alpha_k \notin \{0,1\}$. ◀

▶ **Corollary 9.** *There exists an algorithm for* `CCKnapsack` *such that if the value of each element is at most ϵP^*, where P^* is the optimal feasible value, then the algorithm outputs a feasible integral solution I' with $p(I') \geq (1-\epsilon)P^*$. The running time is $O(n^2 \log \frac{n}{\epsilon})$.*

Proof. Run the algorithm of Theorem 3 to find the optimal fractional solution. We know that the optimal fractional value is obtained on a vertex of the corresponding polygon, and that the solution contains at most one non-integral element. Drop that element. Clearly, the new solution is feasible, and its value is smaller than the optimal value by at most ϵP^*. ◀

3.3 Every fractional point in the polygon is dominated by some integral point with almost the same value

The partial lexicographic order on $\mathbb{R}^2$ is $(a,b) \leq (a',b')$ iff $a \leq a'$ and $b \leq b'$. We claim:

▶ **Lemma 10.** *For every point $X \in \Lambda_P$ there exists a vertex I of Λ_P and an integral point $A \notin \Lambda_P$, such that $A \leq X$, $A \leq I$ and $I - A = \gamma Q^{(i)} + \delta Q^{(j)}$ where $i,j \in [n]$ are the active items at vertex I and $0 \leq \gamma, \delta \leq 1$. I.e., for every point X in the polygon Λ_P there exists an integral point A such that $A \leq X$ in the lexicographic partial order, and A is almost a vertex of the polygon differing from some vertex in at most two elements.*

Proof. We start at the point X, and *vertically* go *down* till we reach the boundary of Λ_P at some point Y. Obviously, $\mu(Y) = \mu(X)$ and $V(Y) \leq V(X)$, and hence $Y \leq X$. Let I and J be the two vertices of Λ_P to the left and right of Y, respectively (if Y is a vertex $I = Y$). Notice that $\mu(I) \leq \mu(Y) = \mu(X)$ and at least one of $V(I) \leq V(Y)$ or $V(J) \leq V(Y)$ holds.

According to Theorem (8), I has a representation $I = \sum \alpha_i Q^{(i)}$ that has at most one element $k \in [n]$ in which $\alpha_k \notin \{0,1\}$. Let i and j be the active items at vertex I.

▶ **Claim 11.** $k \in \{i,j\}$.

Proof. Suppose not, i.e. $\alpha_k \notin \{0,1\}$ and yet we do a replacement (i,j) in which k does not participate. Since I is a vertex we must have $\alpha_j = 0$ and $\alpha_i = 1$. Now we notice that:

- $\alpha_k < 1$ and $\alpha_i = 1$, hence k could have replaced i, and,
- $\alpha_k > 0$ and $\alpha_j = 0$, hence j could have replaced k.

The direction of k replacing i is $E_{k,i} = Q^{(k)} - Q^{(i)}$. The direction of j replacing k is $E_{j,k} = Q^{(j)} - Q^{(k)}$. Notice that $E_{j,i} = E_{j,k} + E_{k,i}$. Hence, $E_{j,i}$ is inside the parallelogram defined by $E_{k,i}$ and $E_{j,k}$, and therefore the algorithm cannot choose the replacement (i,j) since it doesn't give the smallest angle with the vector $(0,-1)$. A contradiction. ◀

Let A be the *integral* point that is received by throwing elements i and j from the vertex I (if they participate in I), i.e. $A = \sum_{\ell \neq i,j} \alpha_\ell Q^{(\ell)}$. Therefore, $A \leq I$. Also, since vertex J is received from vertex I by decreasing element i and increasing element j, $J \geq \sum_{\ell \neq i} \alpha_\ell Q^{(\ell)}$. Hence $A \leq J$. Thus $A \leq I$ and $A \leq J$. In particular, $\mu(A) \leq \mu(I) \leq \mu(Y)$ and also $V(A) \leq V(I)$ and $V(A) \leq V(J)$. Therefore, $V(A) \leq V(Y)$ and hence $A \leq Y \leq X$. ◀

4 The Relaxed FPTAS for CCKnapsack

In this section we present algorithm *IntegralRelaxed*, which is a relaxed FPTAS for `CCKnapsack`. The input to IntegralRelaxed is the same as that of `CCKnapsack`, defined in Section 2. Suppose the optimal solution outputs a set of items I^* with total value P^* such that the probability the total size of the items in I^* exceeds C is at most ζ (C and ζ are inputs to the problem). The output of IntegralRelaxed is a set of items I' with total value at least $(1-5\epsilon)P^*$ and the probability the total size of items in I' exceeds $(1+\epsilon)C$ is at most ζ. Before we explain the algorithm we need a few pre-requisites:

- The algorithm uses as a subroutine a dynamic programming algorithm (described in tails in [10]) that does the following. The input to the algorithm is a set of triplets $\left\{(a^{(i)}, b^{(i)}, c^{(i)}) \mid i \in S\right\}$. Define the set of partial sums in each coordinate by $PS_a = \left\{\sum_{i\in I} a^{(i)} \mid I \subseteq S\right\}$, and similarly for PS_b and PS_c. The algorithm finds all triplets $\left\{\sum_{i\in I}(a^{(i)}, b^{(i)}, c^{(i)}) \mid I \subseteq S\right\}$, i.e., triplets in $PS_a \times PS_b \times PS_c$ that can be obtained as the partial sum of the same set $I \subseteq S$. The algorithm does that by keeping a table whose k'th row records all combinations that can be obtained by the first k triplets in S. The procedure $OUT((s_1, s_2, s_3), |S|)$ of the algorithm checks whether (s_1, s_2, s_3) is a feasible triplet, and if so returns a subset $I \subseteq S$ that obtains it. The running time of the algorithm is polynomial in the number of partial sums.
- We need a truncation operator to truncate heavy elements so that the dynamic programming algorithm runs in polynomial time. We define $\lfloor x \rfloor_S = \lfloor \frac{x}{S} \rfloor \cdot S$. If $x \in [0, KS)$, then $\lfloor x \rfloor_S$ also belongs to $[0, KS)$ but may belong only to a set of K points which are the first points in consecutive intervals of length S partitioning $[0, KS)$.
- Define $OFP_C(\mu, V) = 1 - \Phi(\frac{C-\mu}{\sqrt{V}})$, and also for $I = (\mu, v)$ write $OFP_C(I) = OFP_C(\mu, v)$. We use the following fact, when proving that we get a relaxed constraint: For every $B > 0$, $OFP_{B\cdot C}(B\mu, B^2V) = 1 - \Phi(\frac{BC-B\mu}{\sqrt{B^2V}}) = 1 - \Phi(\frac{C-\mu}{\sqrt{V}}) = OFP_C(\mu, V)$.
- Finally, when we have a sequence $\left\{p^{(i)}\right\}_{i=1}^n$, we think of it as a function $p(i) = p^{(i)}$ and extend it to sets $A \subseteq [n]$ by letting $p(A) = \sum_{a\in A} p(a)$. We do the same for $\mu, V, \overline{p}$, etc.

With that IntegralRelaxed does the following. First it guesses four values:

- $\widehat{p}$: the guessed total value of the optimal solution.
- $\widehat{p}_h$: the guessed value of the heavy elements in the optimal solution.
- $\widehat{\mu}_h$: the guessed mean value of the heavy elements in the optimal solution.
- $\widehat{V}_h$: the guessed variance of the heavy elements in the optimal solution.

For each such guessed quadruplets, IntegralRelaxed splits the input elements to *heavy* and *light*, such that the value of an heavy item is at least $\epsilon\widehat{p}$. Then, it truncates the value, the mean and the variance of each *heavy* element, i.e., $\overline{p}^{(i)} = \lfloor p^{(i)} \rfloor_{\widehat{p}_h/T_1}$, $\overline{\mu}^{(i)} = \lfloor \mu^{(i)} \rfloor_{\widehat{\mu}_h/T_2}$, $\overline{V}^{(i)} = \lfloor V^{(i)} \rfloor_{\widehat{V}_h/T_2}$, where $T_1 = \frac{1}{\epsilon^2}$ and $T_2 = \frac{1+\epsilon}{\epsilon}n$. The truncated values of the heavy elements are passed as an input to the dynamic programming algorithm, the light elements are solved using Algorithm FPBoundry. More specifically, the algorithm does the following:

IntegralRelaxed(ϵ)

- Go over all $(\widehat{p}, \widehat{p}_h, \widehat{\mu}_h, \widehat{V}_h)$ tuples with $\widehat{p}_h \le \widehat{p}$ that are in:
  - $\widehat{p} \in \left\{P_{min}(1+\epsilon)^i \mid i \ge 0\ ,\ P_{min}(1+\epsilon)^i \le (1+\epsilon)P_{total}\right\}$,
  - $\widehat{p}_h \in \left\{P_{min}(1+\epsilon)^i \mid i \ge 0\ ,\ P_{min}(1+\epsilon)^i \le (1+\epsilon)P_{total}\right\}$,
  - $\widehat{\mu}_h \in \left\{\mu_{min}(1+\epsilon)^i \mid i \ge 0\ ,\ \mu_{min}(1+\epsilon)^i \le (1+\epsilon)\mu_{total}\right\}$,
  - $\widehat{V}_h \in \left\{V_{min}(1+\epsilon)^i \mid i \ge 0\ ,\ V_{min}(1+\epsilon)^i \le (1+\epsilon)V_{total}\right\}$,

where $P_{min} = \min\left\{p^{(i)} | i \in [n]\right\}$ and $P_{total} = \sum_{i\in[n]} P^{(i)}$. Similarly for μ_{min}, μ_{total}, V_{min}, V_{total}.

• For each such $(\widehat{p},\widehat{p}_h,\widehat{\mu}_h,\widehat{V}_h)$ divide $[n]$ to heavy and light items, $H = \left\{i \mid p^{(i)} \geq \epsilon\widehat{p}\right\}$ and $L = \left\{i \mid p^{(i)} < \epsilon\widehat{p}\right\}$.
• Fix $T_1 = \frac{1}{\epsilon^2}$ and $T_2 = \frac{1+\epsilon}{\epsilon}n$. Run the dynamic programming algorithm presented in [10] on the input $\left\{(\overline{p}^{(i)} = \lfloor p^{(i)}\rfloor_{\widehat{p}_h/T_1}, \overline{\mu}^{(i)} = \lfloor \mu^{(i)}\rfloor_{\widehat{\mu}_h/T_2}, \overline{V}^{(i)} = \lfloor V^{(i)}\rfloor_{\widehat{V}_h/T_2})\right\}_{i\in H}$ and compute all partial sums. Now we do the following two checks:

1. (Check that $(\widehat{p}_h,\widehat{\mu}_h,\widehat{V}_h)$ is heavy-feasible). Check that there exists some partial sum $(\overline{p},\overline{\mu},\overline{V})$ such that $(1-2\epsilon)\widehat{p}_h \leq \overline{p} \leq \widehat{p}_h$, $\overline{\mu} \leq \widehat{\mu}_h$, $\overline{V} \leq \widehat{V}_h$. If there is no such $(\overline{p},\overline{\mu},\overline{V})$ the test for $(\widehat{p},\widehat{p}_h,\widehat{\mu}_h,\widehat{V}_h)$ failed. If there is any such $(\overline{p},\overline{\mu},\overline{V})$ let $H' = OUT((\overline{p},\overline{\mu},\overline{V}),|H|)$
2. (Check that the light elements can complete a good solution): Let $\Lambda_{\frac{1}{1+\epsilon}\widehat{p}-\widehat{p}_h}$ be the polygon defined in Equation (1) using the light items, L, and the target value $\frac{1}{1+\epsilon}\widehat{p}-\widehat{p}_h$. Use Algorithm FPBoundry to enumerate the (at most) n^2 vertices of $\Lambda_{\frac{1}{1+\epsilon}\widehat{p}-\widehat{p}_h}$. For each vertex I, throw away the two active elements, i and j, to get a set $L' \subseteq L$ of integral light items (if the polygon is empty, $L' = \emptyset$). This step passes if
$$OFP_C(\mu(L') + \frac{\mu(H')}{1+\epsilon}, V(L') + \frac{V(H')}{1+\epsilon}) \quad \leq \quad \zeta.$$

• Let $\widehat{p}$ be the largest value for which steps 1 and 2 passed for some $(\widehat{p}_h,\widehat{\mu}_h,\widehat{V}_h)$. Suppose we used $(\overline{p},\overline{\mu},\overline{V})$ and H' in step 1 and L' in step 2 when accepting $(\widehat{p},\widehat{p}_h,\widehat{\mu}_h,\widehat{V}_h)$. Return $H' \cup L'$.

▶ **Lemma 12.** *Let ϵ, T_1, T_2 be as defined before. Let I^* be an optimal feasible integral solution with value P^*, mean μ^* and variance V^*. Let $\widehat{p}$ be such that $P^* \leq \widehat{p} \leq (1+\epsilon)P^*$. Define $H = \left\{i \mid p^{(i)} \geq \epsilon\widehat{p}\right\}$ and $L = \left\{i \mid p^{(i)} < \epsilon\widehat{p}\right\}$. Denote $H^* = I^* \cap H$ and $L^* = I^* \cap L$. Let $\widehat{p}_h,\widehat{\mu}_h,\widehat{V}_h$ be such that $p(H^*) \leq \widehat{p}_h \leq (1+\epsilon)p(H^*)$, $\mu(H^*) \leq \widehat{\mu}_h \leq (1+\epsilon)\mu(H^*)$ and $V(H^*) \leq \widehat{V}_h \leq (1+\epsilon)V(H^*)$. Then, IntegralRelaxed$(\epsilon)$ accepts $(\widehat{p},\widehat{p}_h,\widehat{\mu}_h,\widehat{V}_h)$ and the value of the associated set is at least $(1-5\epsilon)P^*$.*

Proof. One of the solutions the dynamic programming algorithm generates is $(\overline{p} = \overline{p}(H^*), \overline{\mu} = \overline{\mu}(H^*), \overline{V} = \overline{V}(H^*))$. It is clear that $\overline{p} = \overline{p}(H^*) \leq p(H^*) \leq \widehat{p}_h$. Similarly, $\overline{\mu} \leq \widehat{\mu}_h$ and $\overline{V} \leq \widehat{V}_h$. Also, we first notice that for every $i \in H$ we have

$$\overline{p}^{(i)} = \lfloor p^{(i)}\rfloor_{\widehat{p}_h/T_1} \geq p^{(i)} - \frac{\widehat{p}_h}{T_1} = p^{(i)} - \epsilon^2\widehat{p}_h \geq p^{(i)} - \epsilon^2\frac{p^{(i)}}{\epsilon} = (1-\epsilon)p^{(i)},$$

because $p^{(i)} \geq \epsilon\widehat{p} \geq \epsilon\widehat{p}_h$ (as $i \in H$). Therefore, $\overline{p} = \overline{p}(H^*) \geq (1-\epsilon)p(H^*) \geq \frac{1-\epsilon}{1+\epsilon}\widehat{p}_h \geq (1-2\epsilon)\widehat{p}_h$. Hence the check at step 1 passes. Let $H' = OUT((\overline{p},\overline{\mu},\overline{V}),|H|)$.

Next the algorithm does the check at step 2. We first notice that L^* has value $p(L^*) = p(I^*) - p(H^*) \geq P^* - \widehat{p}_h \geq \frac{1}{1+\epsilon}\widehat{p} - \widehat{p}_h$, and therefore the polygon $\Lambda_{\frac{1}{1+\epsilon}\widehat{p}-\widehat{p}_h}$ is not empty and there exists some X in the polygon that is supported over elements from L^*, i.e., $X = \sum_{i\in L^*}\alpha_i Q^{(i)} \leq L^*$. By Lemma (10) there exists a vertex I and an integral point L' such that $L' \leq X$, and $I - L' = \gamma Q^{(i)} + \delta Q^{(j)}$ for i and j that are the active items in vertex I and $0 \leq \gamma, \delta \leq 0$.

When the algorithm goes over all the vertices in $\Lambda_{\frac{1}{1+\epsilon}\widehat{p}-\widehat{p}_h}$ it also checks I, and it computes the integral solution L' (which is the vertex I with the active items removed), and when checking L' we have $\mu(L') \leq \mu(X) \leq \mu(L^*)$ and $V(L') \leq V(X) \leq V(L^*)$. Notice that $\mu(H') \leq \overline{\mu}(H') + n\frac{\widehat{\mu}_h}{T_2} \leq \overline{\mu}(H^*) + n\frac{(1+\epsilon)\mu(H^*)}{T_2} \leq (1+\epsilon)\mu(H^*)$. Similarly, $V(H') \leq (1+\epsilon)V(H^*)$. Together,

$$\begin{aligned} OFP_C(\mu(L') + \frac{\mu(H')}{1+\epsilon}, V(L') + \frac{v(H')}{1+\epsilon}) \quad &\leq \quad OFP_C(\mu(L^*) + \mu(H^*), V(L^*) + V(H^*)) \\ &= \quad OFP_C(\mu(I^*), V(I^*)) \leq \zeta. \end{aligned}$$

Thus, when the algorithm gets to check vertex I (and the algorithm checks all vertices I) the algorithm accepts and returns the solution $H' \cup L'$. Finally, notice that

$$\begin{aligned} p(H' \cup L') &= p(H') + p(L') \geq \overline{p}(H') + \frac{1}{1+\epsilon}\widehat{p} - \widehat{p}_h - 2\epsilon\widehat{p} \geq \overline{p} + (1-\epsilon)\widehat{p} - \widehat{p}_h - 2\epsilon\widehat{p} \\ &\geq (1-2\epsilon)\widehat{p}_h + (1-3\epsilon)\widehat{p} - \widehat{p}_h = (1-3\epsilon)\widehat{p} - 2\epsilon\widehat{p}_h \geq (1-5\epsilon)\widehat{p} \geq (1-5\epsilon)P^* \end{aligned}$$

◀

▶ **Lemma 13.** *If algorithm $IntegralRelaxed(\epsilon)$ returns the set I', then $OFP_{(1+\epsilon)C}(I') \leq \zeta$.*

Proof. Suppose the algorithm accepts and returns the integral set $I' = H' \cup L'$. As the algorithm passed both checks, we know that $OFP_C(\mu(L') + \frac{\mu(H')}{1+\epsilon}, V(L') + \frac{v(H')}{1+\epsilon}) \leq \zeta$. Hence,

$$\begin{aligned} OFP_{(1+\epsilon)C}(\mu(I'), V(I')) &= OFP_{(1+\epsilon)C}(\mu(L') + \mu(H'), V(L') + V(H')) \\ &\leq OFP_{(1+\epsilon)C}((1+\epsilon)(\mu(L') + \frac{\mu(H')}{1+\epsilon}), (1+\epsilon)^2(V(L') + \frac{V(H')}{1+\epsilon})) \\ &= OFP_C(\mu(L') + \frac{\mu(H')}{1+\epsilon}, V(L') + \frac{V(H')}{1+\epsilon}) \leq \zeta. \end{aligned}$$

◀

In [10] we prove:

▶ **Lemma 14.** *$IntegralRelaxed(\epsilon)$ takes $\tilde{O}(\frac{len^4 n^6}{\epsilon^8})$ time, where len is the input length.*

References

1. Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. Improved approximation results for stochastic knapsack problems. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1647–1665, 2011.
2. Daniel Blado, Weihong Hu, and Alejandro Toriello. Semi-infinite relaxations for the dynamic knapsack problem with stochastic item sizes. *SIAM Journal on Optimization*, 26(3):1625–1648, 2016.
3. Anindya De. Boolean function analysis meets stochastic optimization: An approximation scheme for stochastic knapsack. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1286–1305. SIAM, 2018.
4. Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *IEEE FOCS*, pages 579–586, 1999.
5. Vineet Goyal and R Ravi. A PTAS for the chance-constrained knapsack problem with random item sizes. *Operations Research Letters*, 38:162–164, 2010.
6. Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.
7. Jian Li and Wen Yuan. Stochastic combinatorial optimization via poisson approximation. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 971–980. ACM, 2013.
8. Evdokia Nikolova. Approximation algorithms for offline risk-averse combinatorial optimization. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 338–351, 2010.
9. Galia Shabtai, Danny Raz, and Yuval Shavitt. Risk aware stochastic placement of cloud services: the case of two data centers. In *International Workshop on Algorithmic Aspects of Cloud Computing*, pages 59–88. Springer, 2017.
10. Galia Shabtai, Danny Raz, and Yuval Shavitt. A relaxed FPTAS for Chance-Constrained Knapsack - Technical Report, 2018. To appear.