# A faster FPTAS for counting two-rowed contingency tables

Tzvi Alon * Nir Halman †

August 10, 2018

**Abstract**

In this paper we provide a deterministic fully polynomial time approximation scheme (FPTAS) for counting two-rowed contingency tables that is faster than any either deterministic or randomized approximation scheme for this problem known to date. Our FPTAS is derived via a somewhat sophisticated usage of the method of $K$-approximation sets and functions introduced by Halman et al. [*Math. Oper. Res.*, 34, (2009), pp. 674–685].

**Keywords:** contingency tables, dynamic programming, $K$-approximation sets and functions.

## 1 Introduction

The field of approximate counting is largely based on Markov chain Monte Carlo sampling, a technique that is inherently randomized, and is used to design fully polynomial randomized approximation schemes (FPRASes). Given an input instance $I$, an approximation parameter $\epsilon > 0$, and an uncertainty parameter $1 > \delta > 0$, an FPRAS estimates with a success probability of at least $1 - \delta$ the number of solutions for $I$ within a relative error of $\epsilon$, i.e., multiplicative error of $1 + \epsilon$. The running time of the scheme is polynomial in the (binary) input size $|I|$, in $\frac{1}{\epsilon}$ and in $\log \frac{1}{\delta}$. An incomplete list of counting problems that exhibit FPRASs is: Hamiltonian cycles in dense graphs [5], knapsack solutions [8], contingency tables with constant number of rows [6, 2, 3], Eulerian orientations of a directed graph [21], matchings in graphs with constant maximum degree [17, 18] and colorings [16]. There are considerably fewer examples of counting problems that admit FPTASes, i.e., deterministic algorithms that approximate the number of solutions within relative error $\epsilon$ in time polynomial in $|I|$ and $\frac{1}{\epsilon}$. Examples include counting independent sets for graphs of maximum degree $d$ [24], matchings [1], knapsack solutions [10, 23, 22, 11] and contingency tables [10, 23].

In this paper we consider the problem of counting two-rowed contingency tables. Before elaborating on this problem, we give its formal definition. Let us denote by $\mathbb{N}$ the set of natural numbers and by $\mathbb{Z}^+$ the set of nonnegative integers. In the problem of *counting contingency tables with a constant number of rows* we are given vectors $\vec{r} = (r_1, \ldots, r_m) \in \mathbb{N}^m$, $\vec{s} = (s_1, \ldots, s_n) \in \mathbb{N}^n$, and a number $N \in \mathbb{N}$, such that $\vec{r}, \vec{s}$ are partitions of $N$, i.e., $N = \sum_{k=1}^{m} r_k = \sum_{k=1}^{n} s_k$. The set $\Sigma_{r,s}$ of contingency tables with row sums $r$ and column sums $s$ is defined by

$$\Sigma_{r,s} = \left\{ Z \in \mathbb{Z}^{+^{m \times n}} : \sum_{j=1}^{n} Z_{ij} = r_i \text{ for } 1 \leq i \leq m, \sum_{i=1}^{m} Z_{ij} = s_j \text{ for } 1 \leq j \leq n \right\}. \quad (1)$$

*Hebrew University of Jerusalem, Israel, E-mail: tzvi.alon@mail.huji.ac.il.

†Hebrew University of Jerusalem, Israel, E-mail: halman@huji.ac.il. Supported in part by the Israel Science Foundation, grant number 399/17.

Sampling and counting contingency tables have been studied intensively, see the discussion and references in [8], perhaps because of the relevance of this problem to practice. See, e.g., Diaconis and Efron [4], who discuss the interpretation of rejection in chi-square independency tests when the number of contingency tables is known.

**Our result.** The fastest approximation scheme to estimate the number of two-rowed contingency tables (i.e., for $m = 2$) known to date is the (randomized) FPRAS of Dyer and Greenhill [6], which is tailor-made to the case of two rows. While the running time of their algorithm is only given implicitly, a simple analysis shows that the running time is $O\left(\frac{n^5}{\epsilon^2}\log^3 N \log(n\log N)\log\frac{Nn\log N}{\epsilon}\right)$, see Appendix A. In this paper we give a (deterministic) FPTAS for two-rowed contingency tables, that although being deterministic, runs faster than the (randomized) FPRAS of Dyer and Greenhill [6] by a factor of $\frac{n^2}{\epsilon}$, up to log terms.

**Theorem 1.1** *Let $\vec{r} = (r_1, r_2)$, $\vec{s} = (s_1, \ldots, s_n)$ and $N$ be an instance of contingency tables with two rows. Let $R = \min\{r_1, r_2\}$ and $S = \max\{s_1, \ldots, s_n\}$. For every $0 < \epsilon < 1$ there exists a deterministic $O(\frac{(n\log S)^3}{\epsilon}\log\frac{n\log S}{\epsilon}\log R)$ time algorithm that estimates the number of distinct two-rowed contingency tables within a relative error $\epsilon$.*

**Literature review.** Dyer introduces a randomized algorithm for the general case of contingency tables with $m$ rows, which is strongly polynomial [8]. The running time of his algorithm is $O\left(n^{4m+1} + \frac{n^{3m}}{\epsilon^2}\right)$. For $m = 2$ the running time is $O\left(n^9 + \frac{n^6}{\epsilon^2}\right)$. Our algorithm is faster by a factor of at least $\frac{n^3}{\epsilon}$, up to log terms, and is deterministic, but not strongly polynomial. Kijima and Matsui [19] also introduce an FPRAS for the general case, which is an extension and modification of [6]. Their running time is $O\left(M\left(m\left(\frac{M}{\epsilon}\right)^2\log\frac{M}{\delta} + \tau(\epsilon)\right)\right)$, where $M = mn\log N$ and $\tau(\epsilon)$ is the mixing time to attain an approximation uniform sampler from $\Sigma_{r,s}$. They give [2] as an example for such a sampler, with $\tau(\epsilon) \geq n^{34m^3+452m^2}$. Gopalan et al. give the first (deterministic) FPTAS for contingency tables with $m$ rows [10] (see also [9]) which runs in time $O\left(m2^{4m+1}n^{5n+2\frac{\log R}{\epsilon}}\right)^m$, which translated in the case of $m = 2$ to $O\left(n^{24}\left(\frac{\log R}{\epsilon}\right)^2\right)$. Their algorithm is not strongly polynomial, and is slower than the randomized algorithm of Dyer in both $n$ and $\frac{1}{\epsilon}$. We note in passing that the running time dependency on $m$ of both algorithms of [8, 10] is exponential. In addition, the algorithm of Gopalan et al. is described by themselves as "fairly intricate and involve a combination of Dyer's FPRAS for counting contingency tables and our algorithms for counting general integer knapsack solutions and counting knapsack solutions under small space sources" [9, Appendix B]). We note that the real achievement of [9] is providing an FPTAS for the general case of the problem with $m$ rows. Because of this reason they use a somewhat more sophisticated approach than ours, relying on read-once branching programs and insight from Meka and Zuckerman [20]. Our algorithm is relatively simple, its running time dependence on $n$ (resp. $\epsilon$) is faster by a factor of $n^{21}$ (resp. $\frac{1}{\epsilon}$), up to log terms. However, its running time dependency on $R$ is slower by a factor of at most $\log^2 R$.

**Technique used.** Similarly to the FPTASes mentioned above, we also start with formulating the counting problem as a dynamic program (DP, to be distinguished from dynamic programming by context). But we use primal formulations, instead of dual formulations that are used in other FPTASes. We carefully choose DP formulation with which it is possible to apply the emerging technique of $K$-approximation sets and functions introduced in [14]. This technique has already been employed to yield FPTASes for various #P-hard problems such as single-item inventory control, single-item

batch dispatch, single-resource revenue management, growth models, lifetime consumption of risky capital, cash management, energy management and counting the number of integer knapsack solutions [14, 15, 13, 11, 12]. Our DP formulation also take advantage of the recent technique of "binding constraints" by [11] and the key Proposition 3.1 about a certain structure of the objective function. We note in passing that this paper makes the first usage of the technique of $K$-approximation sets and functions for functions that are neither monotone nor convex.

**Our contribution.** We note that counting two-rowed contingency tables is similar to counting integer knapsack solutions where all items have unit weights – the only difference being that we want to count solutions with total weight *equal* to the limit (as opposed to having total weight of *at most* the limit). It is perhaps for this reason that the results of [9, 10] about counting contingency tables and knapsack solutions are derived in similar ways. However, this difference is not negligible – apparently, this is the reason that until this work there was a gap of $\frac{n^2}{\epsilon}$, up to log terms, between the running times of the fastest FPTAS for counting integer knapsack solutions [11] and the fastest approximation scheme for two-rowed contingency tables. In this work we close this gap. We leave the design of a deterministic FPTAS for two-rowed contingency tables that runs faster than $\Omega(\frac{n^3}{\epsilon})$ as an interesting open problem.

## 2 An overview of $K$-approximation sets and functions

In this section we survey the method of $K$-approximation sets and functions as defined in [13]. Let $K \geq 1$ ($M \geq 1$) be an arbitrary real (integer) valued number, respectively. To simplify the discussion, we modify Halman et al.'s definitions and results in [13, Sect. 4-5] to nonnegative integer-valued nondecreasing functions $\varphi : \{A, \ldots, B\} \rightarrow \{0, \ldots, M\}$ over contiguous intervals of integer numbers (analogous definitions and results for nonincreasing functions can be found in [13].) We say that $\tilde{\varphi}$ is a *$K$-approximation function of* $\varphi$ if $\varphi(x) \leq \tilde{\varphi}(x) \leq K\varphi(x)$ holds for every point $x \in \{A, \ldots, B\}$. The method of $K$-approximation sets and functions enables us to build a $K$-approximation for $\varphi$ that is both *succinct* (of size polylogarithmic in $A, B, M$) and *efficient* (can be built in time polylogarithmic in these terms).

**Definition 2.1** *Let $K \geq 1$, and let $\varphi : \{A, \ldots, B\} \rightarrow \{0, \ldots, M\}$ be a nondecreasing function. Let $W = \{w_1, \ldots, w_r\}$ be a subset of $\{A, \ldots, B\}$, where $A = w_1 < w_2 < \ldots < w_r = B$. We say that $W$ is a* $K$-approximation set *of $\varphi$ if $\varphi(w_{j+1}) \leq K\varphi(w_j)$ for each $j = 1, \ldots, r-1$ that satisfies $w_{j+1} - w_j > 1$. The* approximation of $\varphi$ induced by $W$ *is:*

$$\hat{\varphi}(x) = \begin{cases} \varphi(x) & x \in W \\ \varphi(w_{i+1}) & w_i < x < w_{i+1} \quad \text{for some } i. \end{cases}$$

The following two propositions show the usefulness of Definition 2.1 to achieve a succinct $K$-approximation function. Proposition 2.2 tells us that the approximation function induced by a $K$-approximation set is indeed a $K$-approximation function, and Proposition 2.3 is about how we can construct efficiently a succinct $K$-approximation set.

**Proposition 2.2** *([13, Prop. 4.5]) Let $\varphi : \{A, \ldots, B\} \rightarrow \{0, \ldots, M\}$ be a nondecreasing function. Let $K \geq 1$, and let $W$ be a $K$-approximation set of $\varphi$. Let $\hat{\varphi}$ be the approximation of $\varphi$ induced by $W$. Then $\hat{\varphi}$ is a nondecreasing $K$-approximation function of $\varphi$. In addition, if $\varphi$ is stored as a sorted array $\{(x, \varphi(x)) \,|\, x \in W\}$, then for any $x \in \{A, \ldots, B\}$, $\hat{\varphi}(x)$ can be determined in $O(\log |W|)$ time.*

Algorithm ApxSet$(\varphi, D, x^*, K)$, [13, Alg. 1], computes a $K$-approximation set for a unimodal function $\varphi$ over a finite domain $D$ of real numbers that is minimized at $x^*$. In this paper we use this

algorithm for monotone functions on contiguous intervals of integer numbers. Thus, for simplicity, we omit $x^*$ from the input, and denote the algorithm by $\text{ApxSet}(\varphi, \{A, \ldots, B\}, K)$.

**Algorithm 1** Constructing a $K$-approximation set for a nondecreasing function $\varphi$

```
Function ApxSet(φ, {A, ..., B}, K)
x ← B,  W ← {A, B}
while x > A do
    x ← min{x − 1, min{y ∈ {A, ..., B} | Kφ(y) ≥ φ(x)}}
    W ← W ∪ {x}
end while
return W
```

**Proposition 2.3** *([13, Prop. 4.6]) Let $\varphi : \{A, \ldots, B\} \to \{0, \ldots, M\}$ be a nondecreasing function. Then, for every given parameters $\varphi, A, B$ and $K > 1$, function* ApxSet *(Algorithm 1) computes a $K$-approximation set of $\varphi$ in $O\left(t_\varphi \left(1 + \log_K M\right) \log |B - A|\right)$ time, where $t_\varphi$ is the maximum time needed to compute $\varphi(x)$ for any given $x$. This $K$-approximation set has cardinality of $O\left(1 + \log_K M\right)$.*

The following property provides a set of general computational rules of $K$-approximation sets and functions. Their validity follows directly from the definition of $K$-approximation sets and functions.

**Property 2.4** *[13, Prop. 5.1] (Calculus of $K$-approximation functions) For $i = 1, 2$ let $K_i > 1$, let $\varphi_i : \{A, \ldots, B\} \to \mathbb{Z}^+$ and let $\tilde{\varphi}_i : \{A, \ldots, B\} \to \mathbb{Z}^+$ be a $K_i$-approximation of $\varphi_i$. The following rules hold:*

***(1) Summation of approximation:*** *$\tilde{\varphi}_1 + \tilde{\varphi}_2$ is a $\max\{K_1, K_2\}$-approximation function of $\varphi_1 + \varphi_2$.*

***(2) Approximation of approximation:*** *If $\varphi_2 = \tilde{\varphi}_1$ then $\tilde{\varphi}_2$ is a $K_1 K_2$-approximation of $\varphi_1$.*

***(3) Maximization of approximation:*** *$\max\{\tilde{\varphi}_1, \tilde{\varphi}_2\}$ is a $\max\{K_1, K_2\}$-approximation function of $\max\{\varphi_1, \varphi_2\}$.*

# 3 Dynamic Programming Formulations

Our FPTAS for two-rowed contingency tables is designed via a somewhat sophisticated usage of the method of $K$-approximation sets and functions that involves a clever DP formulation. Before reaching this DP formulation we define two other DP formulations and explain why they need to be improved in order to get an FPTAS.

We start with some notations. Let $\vec{r}, \vec{s}, R, S$ be as stated in Theorem 1.1, and suppose w.l.o.g. that $R = r_1$. One can view the number $N$ as $N$ identical items to be partitioned in the table. For every $1 \le j \le R,\ 1 \le i \le n$, let $\mathcal{G}_i(j) = \left\{(x_1, \ldots, x_i) \in \mathbb{Z}^{+^i} : \sum_{k=1}^{i} x_k = j \text{ and } 0 \le x_k \le s_k \text{ for } 1 \le k \le i\right\}$ be the set of partial two-rowed $\Sigma_{r,s}$ contingency tables with $n$ columns (see equation (1)), where in the first row there are: (i) $j$ items in the first $i$ columns, and (ii) no items in the remaining columns. Note that $\mathcal{G}_n(R)$ consists of the actual contingency tables. Let $A_i(j) = |\mathcal{G}_i(j)|$. The objective function is then $|\Sigma_{r,s}| = A_n(R)$.

## 3.1 First DP formulation

Dyer and Greenhill give the following DP formulation [6, Page 271].

$$A_i(j) = \begin{cases} A_{i-1}(j) + A_i(j-1) & j - 1 < s_i, \\ A_{i-1}(j) + A_i(j-1) - A_{i-1}(j-1-s_i) & j - 1 \ge s_i, \end{cases} \tag{2}$$

where the boundary conditions are $A_i(0) = 1$ for $0 \leq i \leq n$, and $A_0(j) = 0$ for $1 \leq j \leq R$. An explanation for the formula is as follow: suppose $j - 1 < s_i$, and we want to assign $j$ (identical) items into cells $1, \ldots, i$. There are two cases:

Case 1: The $i$-th cell is empty. Then there are $A_{i-1}(j)$ combinations for the assignment in cells $1, \ldots, i-1$.

Case 2: There is at least one item in the $i$-th cell. We place item $j$ in cell $i$, and then there are $A_i(j-1)$ combinations to assign items $1, \ldots, j-1$ into cells $1, \ldots, i$. (Note: $j - 1 < s_i$, so there is no restriction on the number of items to put in the $i$-th cell.)

Suppose now $j - 1 \geq s_i$. Again, there are two cases:

Case 1: The $i$-th cell is empty. There are $A_{i-1}(j)$ combinations as before.

Case 2: There is at least one item in the $i$-th cell. We place item $j$ in cell $i$, and the number of combinations is as before ($A_i(j-1)$), but we have to preclude the case where the $i$-th cell contains $s_i$ items from items $1, \ldots, j-1$ as well as item j, i.e., to subtract $A_{i-1}(j-1-s_i)$. Note that $A_n(R)$ can be computed in $O(nR)$ time, i.e., pseudopolynomial in the input size.

We state next our key proposition about the structure of function $A_i(\cdot)$, $1 \leq i \leq n$. This special structure enables us to apply the technique of $K$-approximation sets and functions to our problem. We note in passing that in this paper we make the first usage of this technique for functions that are neither convex nor monotone.

**Proposition 3.1** *For every $i = 1, \ldots, n$ let $B_i = \sum_{k=1}^{i} s_k$. The following two properties hold:*

*(1) $A_i(\cdot)$ is symmetric around $\frac{B_i}{2}$ in the range $\{0, \ldots, B_i\}$. I.e., $A_i(j) = A_i(B_i - j)$ for $j = 0, \ldots, \lfloor \frac{B_i}{2} \rfloor$ and in particular $A_i(\lfloor \frac{B_i}{2} \rfloor) = A_i(\lceil \frac{B_i}{2} \rceil)$.*

*(2) $A_i(\cdot)$ is unimodal in the following way: $A_i(j)$ is nondecreasing for $j = 0, \ldots, \lfloor \frac{B_i}{2} \rfloor$, is nonincreasing for $j = \lceil \frac{B_i}{2} \rceil, \ldots, B_i$ and $A_i(j) = 0$ for $j > B_i$.*

*Proof.* We start with Property (1). Let $0 \leq j \leq \lfloor \frac{B_i}{2} \rfloor$. For any assignment of $j$ items in cells $1, \ldots, i$ of row 1, switching between cells $1, \ldots, i$ of rows 1 and 2 gives an assignment of $B_i - j$ items in cells $1, \ldots, i$ in row 1. This provides us a one-to-one correspondence between assignments of $j$ items in cells $1, \ldots, i$ in row 1, and assignments of $B_i - j$ items in cells $1, \ldots, i$ in row 1. This completes the proof of the first property.

We now turn to Property (2). For $j > B_i$, the cells cannot contain the items, so there are no valid assignments, therefore $A_i(j) = 0$. To complete the proof, it suffices to prove that $A_i(j)$ is nondecreasing for $j = 1, \ldots, \lfloor \frac{B_i}{2} \rfloor$.

We now prove by induction on $i$ that $A_i(\cdot)$ is nondecreasing over $\left\{1, \ldots, \lfloor \frac{B_i}{2} \rfloor\right\}$. Considering the base case of $i = 1$,we note that $A_1(\cdot) \equiv 1$ on $\{1, \ldots, B_1\}$, so the function is nondecreasing in the relevant range. This proves the base case.

The induction hypothesis for $i-1$ is that $A_{i-1}(\cdot)$ is nondecreasing over $\left\{1, \ldots, \lfloor \frac{B_{i-1}}{2} \rfloor\right\}$. Let $j$ be such that $2 \leq j \leq \lfloor \frac{B_i}{2} \rfloor$. We need to show that $A_i(j) \geq A_i(j-1)$.

Case 1: $j-1 < s_i$. By (2), $A_i(j) = A_i(j-1) + A_{i-1}(j)$. The proof follows due to the nonnegativity of $A_{i-1}(j)$.

Case 2: $j - 1 \geq s_i$. By (2), $A_i(j) - A_i(j-1) = A_{i-1}(j) - A_{i-1}(j-1-s_i)$. It therefore remains to show that $A_{i-1}(j) \geq A_{i-1}(j-1-s_i)$. Now,

$$j \leq \lfloor \tfrac{B_i}{2} \rfloor \Rightarrow j \leq \tfrac{B_i}{2} + \tfrac{1}{2} \Rightarrow 2j \leq B_{i-1} + s_i + 1 \Rightarrow j - \tfrac{B_{i-1}}{2} \leq \tfrac{B_{i-1}}{2} - (j - s_i - 1) \Rightarrow$$
$$\left| \tfrac{B_{i-1}}{2} - j \right| \leq \left| \tfrac{B_{i-1}}{2} - (j - s_i - 1) \right|.$$

By the symmetry of $A_{i-1}$ around $\frac{B_{i-1}}{2}$ and by the unimodality of $A_{i-1}$, we get that $A_{i-1}(j) \geq A_{i-1}(j-1-s_i)$ as required. □

**Remark.** After writing its proof, we noticed that the second part of Proposition 3.1 is given in [19, Lem. 2]. For the sake of completeness we choose to leave the full proof herein.

Note that function $A_i(\cdot)$, as a unimodal function with a known maximum, can be broken into two monotone functions. Therefore, $A_i(\cdot)$ can be succinctly approximated via the method of $K$-approximation sets and functions as follows. We evaluate $A_i(\cdot)$ only over $\left\{0,\ldots,\lfloor\frac{B_i}{2}\rfloor\right\}$, where it is nondecreasing. The approximation on the entire domain is clear by Proposition 3.1 (see the details in Algorithm 2).

We next introduce function CompressContingency, which is a version of function Compress in [11, Alg. 1].

**Algorithm 2** Calculating a step-wise $K$-approximation of $\varphi$

1: **Function CompressContingency**$(\varphi, K, B_i)$
2: $W \leftarrow \text{ApxSet}(\varphi, \{0,\ldots,\frac{\lfloor B_i\rfloor}{2}\}, K)$
3: Let $\hat{\varphi}$ be the approximation of $\varphi$ induced by $W$
4: Let $\hat{\varphi}(j) = \hat{\varphi}(B_i - j)$ for $j = \lceil\frac{B_i}{2}\rceil,\ldots,B_i$, and $\hat{\varphi}(j) \equiv 0$ for $j > B_i$
5: return $\hat{\varphi}$

The next proposition is similar to [11, Prop. 2.2] and is deduced from Propositions 2.2-2.4 and 3.1 above.

**Proposition 3.2** *Let $K_1, K_2 \geq 1$ be real numbers, $M > 1$ be an integer, and let $\varphi : \{0,\ldots,B\} \to \{0,\ldots,M\}$ be a function with structure as in Proposition 3.1. Let $\bar{\varphi}$ be a $K_2$-approximation function of $\varphi$. Then function CompressContingency $(\bar{\varphi}, K_1, B)$ returns in $O\left((1+t_{\bar{\varphi}})\left(\log_{K_1} M \log B\right)\right)$ time a piecewise step function $\hat{\varphi}$ with structure as in Proposition 3.1, with $O\left(\log_{K_1} M\right)$ pieces, which $K_1K_2$-approximates $\varphi$. The query time of $\hat{\varphi}$ is $O\left(\log\log_{K_1} M\right)$ if it is stored in a sorted array $\{(x,\hat{\varphi})\,|x \in W\}$.*

**Remark.** There are still two issues that prevent us from using the method of $K$-approximation sets and functions with DP formulation (2). The first is that the formulation involves subtraction. The second is that the evaluation of $A_i(\cdot)$ needs to rely only on the functions evaluated before, i.e., on $A_k(\cdot)$ for $k < i$. Therefore we turn to another DP formulation.

### 3.2 Second DP formulation

The boundary conditions and the objective function are the same as for Formulation (2).

$$A_i(j) = \sum_{k=0}^{\min(j,s_i)} A_{i-1}(j-k), \quad i = 1,\ldots,n, \;\; j = 1,\ldots,R. \tag{3}$$

The validation of this formulation is due to the fact that in order to count the number of combinations to assign $j$ items into $i$ cells, it suffices to sum over the number of items $k$ in the $i$-th cell (which cannot exceed $\min(j, s_i)$) and put the remaining items in the first $i-1$ cells. Note that $A_n(R)$ can be computed in $O(nRS)$ time (recall that $S = \max\{s_1,\ldots,s_n\}$).

**Remark.** Solving DP (3) takes pseudo polynomial time in the input size because of the following two reasons: (i) the domain of $A_i(\cdot)$ is of size $R$, and (ii) the summation size $\min(s_i, j)$ may be of order $O(R + s_i)$. (Note that the input size dependency on $R$ and $s_i$ is $\log R + \log s_i$.) By the method of $K$-approximation sets and functions we can overcome the first obstacle of exponential size of the domain of $A_i(\cdot)$. But in order to get rid of the second obstacle of exponential size of the summation, we need to turn to a third DP formulation.

### 3.3 Third DP formulation

The difficulty which arises in formulation (3) also arises in [11], which deals with the problem of counting integer knapsack solutions. Therefore the following arguments are similar to the ones used in [11].

We next give a third DP formulation, which is pseudo-polynomial in the size of $R$ only. Let $\log^+ x$ equal $\log x$ for $x \geq 1$ and 0 otherwise. Let $\mathrm{msb}(x,i) = \lfloor \log\left(x \bmod 2^i\right) \rfloor + 1$. Namely, $\mathrm{msb}(x,i)$ is the location of the most significant 1-digit of $\left(x \bmod 2^i\right)$ if $\left(x \bmod 2^i\right) > 0$, and is $-\infty$ otherwise. E.g., $\mathrm{msb}(5,2) = 1$ and $\mathrm{msb}(4,1) = -\infty$. Denote by $m_i(j) = \min(j, s_i)$, $w_i = 1$ for every $i = 1, \ldots, n$. Then apart from (4e), our DP formulation is identical to formulation (2) in [11].

$$z_{i,\ell,0}(j) = z_{i,\ell-1,0}(j) + z_{i,\ell-1,0}(j - 2^{\ell-1}) \tag{4a}$$

$$z_{i,\ell,1}(j) = z_{i,\ell-1,0}(j) + z_{i,\mathrm{msb}(s_i,\ell-1),1}(j - 2^{\ell-1}) \tag{4b}$$

$$z_{i,1,r}(j) = z_{i-1,\lfloor \log^+ m_{i-1}(j) \rfloor + 1,1}(j) + z_{i-1,\lfloor \log^+ m_{i-1}(j-1) \rfloor + 1,1}(j-1) \tag{4c}$$

$$z_{i,-\infty,1}(j) = z_{i-1,\lfloor \log^+ m_{i-1}(j) \rfloor + 1,1}(j) \tag{4d}$$

$$z_{1,\ell,r}(j) = 1 \qquad \ell = 1, \ldots, \lfloor \log^+ m_i(j) \rfloor + 1, -\infty \tag{4e}$$

$$z_{i,\ell,r}(j) = 0 \qquad \ell = 1, \ldots, \lfloor \log^+ m_i(j) \rfloor + 1, -\infty, i = 2, \ldots, n, j < 0 \tag{4f}$$

where $r = 0, 1$ , $i = 2, \ldots, n$ , $\ell = 2, \ldots, \lfloor \log^+ m_i(j) \rfloor + 1$ , and $j = 0, \ldots, R$ unless otherwise specified. The objective function is $z_{n,\lfloor \log s_n \rfloor + 1,1}(R)$. The complexity of this pseudo-polynomial algorithm is therefore $O(nR \log S)$.

An explanation for DP formulation (4) is as follows: In (3) the evaluation of $A_i(j)$ is done at once by summing over all the possible values for the number of items in the $i$-th cell. In the current formulation we break this evaluation into $\lfloor \log m_i^+(j) \rfloor + 1$ separate simple evaluations: In the $\ell$-th evaluation we look at the $\ell$-th digit of the binary representation of $m_i(j)$ and consider it to be 0 or 1, i.e., consider to put or not to put $2^{\ell-1}$ items in the $i$-th cell. For each one of these two cases we calculate the number of partial two-rowed contingency tables with $n$ columns, where only cells $1, \ldots, i$ can be used in the first row, and in the $i$-th cell of that row there are no more than $s_i \bmod 2^\ell$ items. After considering these two cases for all the digits in the binary representation of $m_i(j)$, we get $A_i(j)$.

For doing this Halman introduces the notion of *binding constraints* [11]. For $\ell \geq 1$ let $z_{i,\ell,0}(j)$ be the number of solutions for partial two-rowed contingency tables, where $j$ items are placed in the first row, that use cells $\{1, \ldots i\}$, put no more than $2^\ell - 1$ items in the $i$-th cell, and no more than $s_k$ items in the $k$-th cell, for $k = 1, \ldots, i-1$. For $\ell \geq 1$ let $z_{i,\ell,1}(j)$ be the number of solutions for two-rowed contingency tables, where $j$ items are placed in the first row, that use cells $\{1, \ldots i\}$, put no more than $s_i \bmod 2^\ell$ items in the $i$-th cell, and no more than $s_k$ items in the $k$-th cell, for $k = 1, \ldots, i-1$. In this way, the future assignments may affect the current assignment: The number of items we can place in the $\ell$-th step into the $i$-th cell is affected by the number of items we will assign in the next steps, i.e., whether the number of items assigned in next steps will leave enough capacity to place in the $\ell$-th step $2^{\ell-1}$ items or not. We consider these two cases by using the third index of $z_{i,\ell,r}(j)$: If $r = 0$ then the constraint of having no more than $s_i$ items in the $i$-th cell is assumed to be non binding (i.e., we assume there is enough capacity for placing $2^\ell - 1$ more items in the $i$-th cell). If, on the other hand, $r = 1$ then this constraint may be binding. E.g., if $s_i = 5$ and $\ell = 2$, and there are already 4 items in cell $i$, we are in the case of $r = 1$, since there is not enough remaining capacity for placing $2^2 - 1 = 3$ additional items. If the $i$-th cell is empty, we are in the case of $r = 0$, since there is enough capacity for placing 3 more items.

We now turn to a more detailed explanation of formulations (4a)-(4f). In the case of equation (4a) we assume that there is enough capacity for putting $2^\ell - 1$ more items in the $i$-th cell, and therefore,

in both cases of the values of the $\ell$-th bit, there is still enough capacity in the $i$-th cell for as many items as we want. In equation (4b) we assume the constraint of having no more than $s_i$ items in the $i$-th cell may be binding. So when putting $2^{\ell-1}$ items in the $i$-th cell, we have to take the constraint into account. If we do not put $2^{\ell-1}$ items, clearly the constraint will not be binding anymore.

The remaining four equations deal with boundary conditions: Equation (4c) deals with the case of $\ell = 1$, i.e., the possibility of having an odd number of items. Equation (4d) can is called only by (4b) when there are exactly $s_i$ items in the $i$-th cell. Equation (4e) deals with the base case of one cell, and the last equation deals with the boundary condition that there is not enough capacity in the $i$-th cell.

# 4 Algorithm

In order to design an FPTAS to our problem, we first extend the DP formulation (4) to any integer positive index $\ell$ by letting $z_{i,\ell,r}(j) = 0$ for $i = 1, \ldots, n\,, r = 0, 1$ and $\ell > \lfloor \log^+ m_i(j) \rfloor + 1$. The solution of the counting problem via this extended set of recurrences remains $z_{n,\lfloor \log s_n \rfloor + 1,1}(R)$. Now, we proceed exactly as in [11, Sec. 2.4] and use the algorithm CountIntegerKnapsackPrimal$(w, C, u, \epsilon)$ with the following parameter settings and changes: (i) $w_i = 1$ for every $i$, (ii) $C = R$, (iii) $u_i = s_i$, (iv) Denote $S = \max_{1 \le i \le n} s_i$, so $U = S$, and (v) Use CompressContingency instead of Compress. For the sake of completeness we state below the algorithm explicitly. For $i = 2, \ldots, n\,;\ \ell = 1, \ldots, \lfloor \log^+ m_i\,(j) \rfloor$ let

$$s_{i,\ell,r} = \begin{cases} 2^\ell - 1 & r = 0 \\ s_i \mod 2^\ell & r = 1. \end{cases} \tag{5}$$

**Algorithm 3** FPTAS for counting two-rowed contingency tables.

**Function CountContingencyTables**$(R, \vec{s}, \epsilon)$
$S \leftarrow \max\{s_1, \ldots, s_n\}, \quad K \leftarrow \sqrt[(n-1)(\lfloor \log S \rfloor + 1)]{1+\epsilon}$
**for** $\ell := 1$ **to** $\lfloor \log s_1 \rfloor + 1$ **and** $r = 0, 1$ **do** $\tilde{z}_{1,\ell,r} \equiv 1$
**for** $i := 2$ **to** $n$ **do**
  $\tilde{z}_{i,-\infty,1}(\cdot) \leftarrow \tilde{z}_{i-1,\lfloor \log^+ m_{i-1}(\cdot) \rfloor + 1,1}(\cdot)$
  **for** $r = 0, 1$ **do** $\tilde{z}_{i,1,r}(\cdot) \leftarrow$ CompressContingency$(\tilde{z}_{i-1,\lfloor \log^+ m_{i-1}(\cdot) \rfloor + 1,1}(\cdot) + \tilde{z}_{i-1,\lfloor \log^+ m_{i-1}(\cdot - 1) \rfloor + 1,1}(\cdot - 1), K, s_{i,1,r} + \sum_{k=1}^{i-1} s_k)$
  **for** $\ell := 2$ **to** $\lfloor \log s_i \rfloor + 1$ **do**
    $\tilde{z}_{i,\ell,0}(\cdot) \leftarrow$ CompressContingency$(\tilde{z}_{i,\ell-1,0}(\cdot) + \tilde{z}_{i,\ell-1,0}(\cdot - 2^{\ell-1}), K, s_{i,\ell,0} + \sum_{k=1}^{i-1} s_k)$
    $\tilde{z}_{i,\ell,1}(\cdot) \leftarrow$ CompressContingency$(\tilde{z}_{i,\ell-1,0}(\cdot) + \tilde{z}_{i,\mathrm{msb}(s_i,\ell-1),1}(\cdot - 2^{\ell-1}), s_{i,\ell,1} + \sum_{k=1}^{i-1} s_k)$
  **end for**
**end for**
**return** $\tilde{z}_{n,\lfloor \log s_n \rfloor + 1,1}(R)$

We first prove that the functions $z_{i,\ell,r}$ have the structure as stated in Proposition 3.1, so the calls to CompressContingency are well defined.

**Proposition 4.1** *The calls to* CompressContingency *in Algorithm 3 are well defined.*

*Proof.* One of the input arguments of the function CompressContingency is $B_i$. In Proposition 3.1 we define $B_i$ to be the number of items in row 1 in a two-rowed contingency table with $n$ columns and using only cells $1, \ldots, i$ of row 1. We show that some properties hold for the function $A_i(\cdot)$ and the constant $B_i$. We will show here that the same properties hold for functions $z_{i,\ell,r}\,(\cdot)$ and carefully chosen constants. Therefore, we can use the function CompressContingency in order to build a $K$-approximation set for $z_{i,\ell,r}\,(\cdot)$. For every set of indices $i, \ell, r$ such that $z_{i,\ell,r}$ is well defined, we next show that the properties of Proposition 3.1 hold for the function $z_{i,\ell,r}(\cdot)$ and the constant

$s_{i,\ell,r} + \sum_{k=1}^{i-1} s_k$, where $s_{i,\ell,r}$ is defined in (5). We will do this by showing that $z_{i,\ell,r}(\cdot)$ is equivalent to function $A_i(\cdot)$ of a modified two-rowed contingency table problem instance with $n$ columns, using only the first $i$ columns of row 1, with the same $R$, and the same $s_1, \ldots, s_{i-1}$, but with a modified value of the capacity of cell $i$ (instead of $s_i$). Note that due to recurrence (4) it suffices to show this only for (5a)-(5c).

Recall that function $z_{i,\ell,r}$ deals with the first $\ell$ bits of $s_i$. Thus, the capacity of the $i$-th cell in this case is $2^\ell - 1$ if there is no restriction on the amount of items, and is $s_i \bmod 2^\ell$ if the constraint of having no more than $s_i$ items in the $i$-th cell is binding. We next show that these situations are dealt with the two cases of $r = 0, 1$.

- **(5a)** - The call to (5a) is either by (5a) itself, or by (5b). In both cases the constraint is assumed to be non binding, so in the modified instance the capacity of cell $i$ as $2^\ell - 1$.
- **(5b)** - The call to (5b) is either by (5b) itself, by (5c), or by (5d) - In the first case, since the constraint is assumed to be binding and the $\ell$-th bit of $s_i$ is 1, when we call (5b) again, the constraint is still binding. In the other cases, the call is to $z_{i-1,\ell,r}$ and we need to take into account the possibility that the constraint is binding for $i-1$. Thus, the constraint of having no more than $s_i$ items in the $i$-th cell may be binding, and the remaining capacity of the $i$-th cell in the modified instance is therefore $s_i \bmod 2^\ell$.
- **(5c)** - If $r = 0$ then the constraint of having no more than $s_i$ items in the $i$-th cell is non binding (as in (5a)), i.e., the option to place an odd number of items in the $i$-th cell is valid, even if $s_i$ is an even number, so we treat the capacity of cell $i$ as 1. If $r = 1$, in a similar way to the explanation for (5b), we assume that the constraint may be binding, so the capacity of cell $i$ in the modified instance is 1 if $s_i$ is odd, and 0 if it is even.

□

Using the analysis in [11, Sec. 2.4], and applying Proposition 3.2 above instead of [11, Prop. 2.2], it is easy to see that the running time of Algorithm 3 is $O\left(\frac{(n \log S)^3}{\epsilon} \log \frac{n \log S}{\epsilon} \log R\right)$ and therefore Theorem 1.1 follows. For the sake of completeness we give below the analysis of correctness and running time of Algorithm 3 explicitly.

We next prove that $\tilde{z}_{n,\lfloor \log s_n \rfloor + 1,1}(R)$ returned by Algorithm 3 is a relative $(1+\epsilon)$- approximation of $|\Sigma_{r,s}|$. To do so, we first show by double induction over $i$ and $\ell$ that $\tilde{z}_{i,\ell,r}$ is a $K^{(i-2)(\lfloor \log S \rfloor + 1) + \ell}$- approximation of $z_{i,\ell,r}$ for $i = 2, \ldots, n, \ell = 1, \ldots, \lfloor \log s_i \rfloor + 1$ and $r = 0, 1$. (For the purpose of the induction we treat the case of $\ell = -\infty$ as if $\ell = 0$.) Note that due to step 3 and (4e), $\tilde{z}_{1,\ell,r}$ are 1-approximations of $z_{1,\ell,r}$.

We first treat the base case of $i = 2$ and $\ell < 2$. Considering step 5, due to (4d) we get that $\tilde{z}_{2,-\infty,1}$ is a 1-approximation of $z_{2,-\infty,1}$ as needed. Considering step 6, due to summation of approximation applied with parameters set to $\varphi_1(\cdot) = z_{1,\lfloor \log^+ m_1(\cdot) \rfloor + 1,1}(\cdot), \varphi_2(\cdot) = z_{1,\lfloor \log^+ m_1(\cdot - 1) \rfloor + 1,1}(\cdot - 1), \tilde{\varphi}_1(\cdot) = \tilde{z}_{1,\lfloor \log^+ m_1(\cdot) \rfloor + 1,1}(\cdot), \tilde{\varphi}_2(\cdot) = \tilde{z}_{1,\lfloor \log^+ m_1(\cdot - 1) \rfloor + 1,1}(\cdot - 1)$ and $K_1 = K_2 = 1$, we get that the function inside the CompressContingency operator is a 1-approximation of $z_{2,1,r}$. Applying Proposition 3.2 with parameters set to $\varphi(\cdot) = z_{2,1,r}(\cdot), \bar{\varphi}(\cdot) = \tilde{\varphi}_1(\cdot) + \tilde{\varphi}_2(\cdot), K_1 = K$ and $K_2 = 1$ we get that $\tilde{z}_{2,1,r}$ is a $K$-approximation of $z_{2,1,r}$ as needed.

We now perform an induction on $\ell$. For the base case of $\ell = 2$, repeating the same arguments for steps 8-9 we get that $z_{2,2,r}$ is a $K^2$-approximation of $z_{2,2,r}$, and by induction on $\ell$, that $\tilde{z}_{2,\ell,r}$ is a $K^\ell$-approximation of $z_{2,\ell,r}$ as needed, thus proving the base case of $i = 2$. The induction hypothesis is that $\tilde{z}_{i,\ell,r}$ is a $K^{(i-2)(\lfloor \log S \rfloor + 1) + \ell}$-approximation of $z_{i,\ell,r}$.

We show next that $z_{i+1,\ell,r}$ is a $K^{(i-1)(\lfloor \log S \rfloor + 1) + \ell}$-approximation of $z_{i+1,\ell,r}$. Due to summation of approximation applied with parameters set to $\varphi_1(\cdot) = z_{i,\lfloor \log^+ m_i(\cdot) \rfloor + 1,1}(\cdot), \varphi_2(\cdot) = z_{i,\lfloor \log^+ m_1(\cdot - 1) \rfloor + 1,1}(\cdot -$

1), and $K_1 = K_2 = K^{(i-1)(\lfloor \log S \rfloor + 1)+\ell}$ (note that the choice of $K_1, K_2$ is done by applying the induction hypothesis and using the inequalities $\log^+ m_i(\cdot) \leq \log s_i \leq \log S$) we get that $\tilde{\varphi}_1 + \tilde{\varphi}_2$ is a $K^{(i-1)(\lfloor \log S \rfloor + 1)+\ell}$- approximation of $z_{i+1,1,r}$. Applying once more Proposition 3.2 with parameters set to $\varphi = z_{i+1,1,r}, \bar{\varphi} = \tilde{\varphi}_1 + \tilde{\varphi}_2, K_1 = K$ and $K_2 = K^{(i-1)(\lfloor \log S \rfloor + 1)}$ we get that $\tilde{z}_{i+1,1,r}$ is a $K^{(i-1)(\lfloor \log S \rfloor + 1)+1}$-approximation of $z_{i+1,1,r}$ as needed. We now perform an induction on $\ell$. For the base case of $\ell = 2$, repeating the same arguments for steps 8-9 we get that $\tilde{z}_{i+1,2,r}$ is a$K^{(i-1)(\lfloor \log S \rfloor + 1)+2}$_ approximation of $z_{i+1,2,r}$, and by induction on $\ell$, that $\tilde{z}_{i+1,\ell,r}$ is a $K^{(i-1)(\lfloor \log S \rfloor + 1)+\ell}$-approximation of $z_{i+1,\ell,r}$. This completes the proof by induction.

Recalling the value of $K$ as set in step 2 and taking $i = n$ we get that $\tilde{z}_{n,\lfloor \log s_n \rfloor + 1,1}$ is indeed a $(1+\epsilon)$-approximation of $z_{n,\lfloor \log s_n \rfloor + 1,1}$, and specifically that $\tilde{z}_{n,\lfloor \log s_n \rfloor + 1,1}(R)$ is indeed a $(1+\epsilon)$-approximation of the solution of the counting problem.

It remains to analyze the complexity of the algorithm. Clearly, the running time of the algorithm is dominated by the operations done in the inner "for" loop, i.e., steps 8-9, which are executed $O(n \log S)$ times. We analyze, w.l.o.g., a single execution of step 8. By Proposition 3.2, the query time of each of the $\tilde{z}_{i,\ell-1,0}(\cdot)$ and $\tilde{z}_{i,\ell-1,0}(\cdot)$ is $O(\log \log_K M)$, where $M$ is an upper bound on the counting problem, e.g., $M = S^n$. Therefore, applying again Proposition 3.2, each call to CompressContingency runs in $O(\log_K M \log R \log \log_K M)$ time. Using the inequality $(1 + \frac{x}{n})^n \leq 1 + 2x$ which holds for $0 \leq x \leq 1$ we get that $K \geq 1 + \frac{\epsilon}{2\left((n-1)(\lfloor \log S \rfloor + 1)+1\right)}$. Using the inequality $\log(1+y) \geq y$ which holds for $y \in [0,1]$, and changing the bases of the logarithms to two, we get that the overall running time of the algorithm is $O(\frac{n^3}{\epsilon} \log^3 S \log R \log \frac{n \log S}{\epsilon})$.

# APPENDIX

## A Running time analysis of Dyer and Greenhill's FPRAS for counting two-rowed contingency tables

Dyer and Greenhill introduce a randomized algorithm based on mixing Markov chains to approximate the number of contingency tables with two rows [6, Sect. 3]. We now outline the analysis of its running time. By [6, page 269], let $d = \sum_{k=3}^{n} \lceil \log s_k \rceil$, $M = \lceil 150 e^2 \frac{d^2}{\epsilon^2} \log \frac{3d}{\delta} \rceil = O\left(\frac{d^2}{\epsilon^2} \log \frac{d}{\delta}\right)$, where the approximation ratio is guaranteed in probability of $1 - \delta$, and ,$T = \tau\left(\frac{\epsilon}{15de^2}\right)$, where $\tau(\epsilon)$ is the mixing time of the Markov chain. By [6, page 270], the running time of the algorithm is $O(dMT)$.

Now, according to Theorem 4.1 in [6], $\tau(\epsilon) = O\left(n^2 \log \frac{N}{\epsilon}\right)$. $M = O\left(\frac{d^2}{\epsilon^2} \log d\right)$ , and by page 269 $d = O(n \log N)$. So the running time is

$$O\left(n \log(N) \frac{d^2}{\epsilon^2} \log(d) n^2 \log \frac{dN}{\epsilon}\right) = O\left(\frac{n^5}{\epsilon^2} \log^3(N) \log(n \log N) \log \frac{Nn \log N}{\epsilon}\right).$$

## References


[1] M. Bayati, D. Gamarnik, D. Katz, C. Nair, and P. Tetali. Simple deterministic approximation algorithms for counting matchings. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 122–127, 2007.

[2] M. Cryan and M. Dyer. A polynomial-time algorithm to approximately count contingency tables when the number of rows is constant. *Journal of Computer and System Sciences*, 67(2):291–310, 2003.

[3] M. Cryan, M. Dyer, L. Goldberg, M. Jerrum, and R. Martin. Rapidly mixing markov chains for sampling contingency tables with a constant number of rows. *SIAM Journal on Computing*, 36(1):247–278, 2006.

[4] P .Diaconis and B .Efron. Testing for independence in a two-way table: new interpretations of the chi-square statistic. *The Annals of Statistics*, 845–874, 1985.

[5] M. Dyer, A. Frieze, and M. Jerrum. Approximately counting Hamilton paths and cycles in dense graphs. *SIAM Journal on Computing*, 27:1262–1272, 1998.

[6] M. Dyer and C. Greenhill. Polynomial-time counting and sampling of two-rowed contingency tables. *Theoretical Computer Science*, 246(1):265–278, 2000.

[7] M. Dyer, R. Kannan, and J. Mount. Sampling contingency tables. *Random Structures and Algorithms*, 10(4):487–506, 1997.

[8] M. E. Dyer. Approximate counting by dynamic programming. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), June 9-11, 2003, San Diego, CA, USA*, pages 693–699, 2003.

[9] P. Gopalan, A. Klivans, and R. Meka: Polynomial-time approximation schemes for Knapsack and related counting problems using branching programs. CoRR **abs/1008.3187** (2010)

[10] P. Gopalan, A. Klivans, R. Meka, D. Štefankovič, S. Vempala, and E. Vigoda. An FPTAS for #Knapsack and related counting problems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 817–826, 2011.

[11] N. Halman. A deterministic fully polynomial time approximation scheme for counting integer knapsack solutions made easy. *Theor. Comput. Sci.*, 645:41–47, 2016.

[12] N. Halman, G. Nannicini, and J. Orlin, James. On the complexity of energy storage problems. *Discrete Optimization*, 28:31–53, 2018.

[13] N. Halman, D. Klabjan, C.-L. Li, J. Orlin, and D. Simchi-Levi. Fully polynomial time approximation schemes for stochastic dynamic programs. *SIAM Journal on Discrete Mathematics*, 28:1725–1796, 2014.

[14] N. Halman, D. Klabjan, M. Mostagir, J. Orlin, and D. Simchi-Levi. A fully polynomial time approximation scheme for single-item stochastic inventory control with discrete demand. *Mathematics of Operations Research*, 34:674–685, 2009.

[15] N. Halman, J. B. Orlin, and D. Simchi-Levi. Approximating the nonlinear newsvendor and single-item stochastic lot-sizing problems when data is given by an oracle. *Operations Research*, 60:429–446, 2012.

[16] M. Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Structures and Algorithms*, 7(2):157–166, 1995.

[17] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18:1149–1178, 1989.

[18] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51:671–697, 2004.

[19] S. Kijima and T.Matsui. Approximate Counting Scheme for $m \times n$ Contingency Tables. *IEICE Transactions on Information and Systems*, 87:308–314, 2004.

[20] R. Meka and D. Zuckerman. Pseudorandom generators for polynomial threshold functions. In *Proc. of the 42nd ACM Symp. on Theory of Computing (STOC)*, pages 427–436, 2010.

[21] M. Mihail and P. Winkler. On the number of Eulerian orientations of a graph. *Algorithmica*, 16:402–414, 1995.

[22] R. Rizzi and A. Tomescu. Faster FPTASes for counting and random generation of knapsack solutions. In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA)*, pages 762–773, 2014.

[23] D. Štefankovič, S. Vempala, and E. Vigoda. A deterministic polynomial-time approximation scheme for counting knapsack solutions. *SIAM Journal on Computing*, 41:356–366, 2012.

[24] D. Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 140–149, 2006.