A Dynamic Programming Algorithm to Compute Joint Distribution of Order Statistics on Graphs

Rigel Galgana
Brown University

Amy Greenwald
Brown University

Takehiro Oyakawa
Brown University

Abstract

Order statistics play a fundamental role in statistical procedures such as risk estimation, outlier detection, and multiple hypothesis testing as well as in the analyses of mechanism design, queues, load balancing, and various other logistical processes involving ranks. In some of these cases, it may be desirable to compute the *exact* values from the joint distribution of d order statistics. While this problem is already computationally difficult even in the case of n independent random variables, the random variables often have no such independence guarantees. Existing methods obtain the cumulative distribution indirectly by first computing and then aggregating over the marginal distributions. In this paper, we simplify and generalize an existing dynamic programming solution which achieves a $O(\frac{d^{d-1}}{n})$ and $O(d^d)$ factor of improvement in both time and space complexity respectively over previous methods.

1 Introduction

Order statistics often arise in the theory and application of risk estimation and management as they possess strong robustness guarantees considering their ease of interpretation and computational simplicity ([? ? ?]). They also appear naturally across various disciplines such as mechanism design and auction theory ([?]), queue inference ([?]), and wireless communication and scheduling ([?]). As such, efficient computation of the joint distribution of order statistics is of significant practical interest.

To be more concrete, let $X_1, \ldots, X_n$ denote a collection of n real-valued random variables. The ith order statistic, denoted as $X_{(i)}$, is defined as the ith smallest value of $X_1, \ldots, X_n$. The problem of interest is as follows:

Definition 1. *We desire to compute the joint cumulative distribution function (cdf) $F_{\mathcal{C}}(\boldsymbol{x})$ of an ordered set $\mathcal{C} = (c_1, \ldots, c_d)$ of $d \in \mathbb{Z}_{>0}$ select order statistics of $n \geq d$ random variables $X_1, \ldots, X_n \sim \boldsymbol{F}$, where $\boldsymbol{x} \in \mathbb{R}^d$. Letting $f_{X_{\mathcal{C}}}(\boldsymbol{x})$ be the corresponding joint probability density function, we have:*

$$F_{X_{\mathcal{C}}}(\boldsymbol{x}) = F_{X_{(c_1)},\ldots,X_{(c_d)}}(\boldsymbol{x}) = \mathbb{P}\left(\bigcap_{j=1}^{d} \{X_{(c_j)} \leq x_j\}\right)$$

The primary main contribution of this work is an efficient algorithm to compute $F_{X_{\mathcal{C}}}$ in the both the independent but not identical setting and dependent random variable settings.

Related Work As mentioned previously, there are several ways of approaching this problem in various special cases. Under the i.i.d. assumption, several efficient algorithms with space and time complexities polynomial in n and d are described in Noe and Vandewiele [?], Kotel-Nikova and Khmaladze [?], Moscovich [?]. Many of these algorithms can be extended to the case of two-sided bounds on the order statistics— $\mathbb{P}(\cap_{j=1}^{d} l_j \leq X_{(c_j)} \leq u_j)$. In von Schroeder and Dickhaus [?], several of these algorithms are generalized to the case where the variables are described by $m = 2$ homogeneous populations. For the general case of m-populations with $m > 1$ and population sizes $n_1, \ldots, n_m$, Glueck [?] provided an simplified the block-permanents in the exact expression of the joint order statistic distribution given by Bapat-Beg [?], reducing the number of computations from $O(n^n)$ to $O(\prod_{k=1}^{m} n_k^d)$. Galgana and Shi [**?**] generalized the method of Khmaladze [**?**] to the m-populations setting, further reducing the time complexity to $O(d\prod_{k=1}^{m} n_k^2)$. In the independent but not identical setting— i.e.$m = n$— Boncelet Jr. [**?**] provides an $O(dn^d)$ algorithm. Boncelet Jr. generalizes his approach to the dependent setting, though the exact implementation and complexity analysis are omitted.

Most of these methods explicitly or implicitly compute the joint order statistic distribution by solving an equivalent combinatorial problem. At a high level, this combinatorial problem involves a partition of the real line into contiguous regions, which we refer to as bins, and tracks the probabilities of the various ways that the random variables could fall within these bins such that certain constraints are satisfied. For example, the algorithm described in [**?**]—which we examine more closely in the following section—recursively updates the joint probability mass function of the number of the first i random variables that fall within the bins. The main contribution of our paper is an acceleration of Boncelet Jr.'s method by exploiting the structure of $\mathcal{C}$. We additionally provide the full generalization of both approaches to the case of two-sided bounds and dependent random variables.

The paper is outlined as follows: In the next section, we focus on the independent, but not necessarily identically distributed random variables case and translate the inherently continuous joint cdf problem as in Equation (1) to an equivalent combinatorial problem (Section 2) . Using this combinatorial setup, we describe Boncelet Jr.'s algorithm, as well as motivate our key insight to improve its performance in (Section 3. We then provide rigorous justification to our algorithmic speedup and state the key theorems and recurrence relation. Afterwards, we state the algorithm in its entirety for the i.i.d. case with accompanying complexity analysis and experiments. Lastly, we describe the extension to two-sided bounds and dependent random variables as well as its impact on time and space complexities (Section 5). We then conclude with a summary of our work (Section 4).

2 The Combinatorial Problem

In this section, we translate the problem of computing the joint cdf of a collection of order statistics to an equivalent combinatorial problem of tossing balls into bins. We assume independence with $X_i \sim F_i \ \forall i \in [n]$ for the following three sections before generalizing to dependent random variables. Restating this computation as a problem involving balls and bins, we define $x_0 = -\infty$ and $x_{d+1} = +\infty$, and partition the real line into $d+1$ intervals, $I_1, \ldots, I_{d+1}$, hereafter bins: $(-\infty, +\infty) = (x_0, x_{d+1}) = \cup_{j=1}^{d}(x_{j-1}, x_j] \cup (x_d, x_{d+1}) = \cup_{j=1}^{d+1} I_j = I_{1:d+1}$. Here, $I_{j:k} = \cup_{i=j}^{k} I_i$ denotes the union of bins I_j through I_k inclusive. Moreover, we let $p_{i,j} = \mathbb{P}(\{X_i \in I_j\}) = F_i(x_j) - F_i(x_{j-1})$ denote the probability that the ith ball resides in the jth bin, for all $j \in [d+1]$. For $j, j' \in [d+1]$, we define the key combinatorial objects:

- Define $C_{i,j} \doteq \sum_{k=1}^{i} \mathbf{1}_{X_k \in I_j}$ as the "number of the first i balls that reside in bin I_j." Overloading notation, we use the random variable $C_{i,j:j'} = \sum_{k=1}^{i} \mathbf{1}_{X_k \in I_{j:j'}}$ as "the number of the first i balls that reside in bins I_j through $I_{j'}$."
- Furthermore, we define the random vectors $\boldsymbol{C}_i \doteq (C_{i,1}, \ldots, C_{i,d}) \in [i]^d$ to denote "the number of the first i balls that reside in each bin."
- For $j \in [d]$, we define the event $D_{i,j} \doteq \{C_{i,1:j} \geq c_j\}$ as "the event that at least c_j of the first i balls reside in the first j bins." We also define $D_{i,j:j'} \doteq \cap_{k=j}^{j'} D_{i,k}$ as "the event that at least c_k of the first balls reside in the first k bins, $I_{1:k}$, for all $k \in \{j, \ldots, j'\}$."

To relate the original problem of computing $F_{\mathcal{C}}(\boldsymbol{x})$ and this combinatorial setup, note that the event $X_{(c_j)} \leq x_j$ is means "the c_jth smallest value of the n random variables is less than or equal to x_j." Equivalently, in the combinatorial setup we have that "there are at least c_j balls in bins $I_{1:j}$": i.e., the event $D_{n,j}$ holds where call $D_{n,j}$ the jth bin condition. With this, we revisit Equation (1) and see that $F_{\mathcal{C}}(\boldsymbol{x})$ is equivalent to:

$$\mathbb{P}(\bigcap_{j=1}^{d} X_{(c_j)} \leq x_j) = \mathbb{P}(\bigcap_{j=1}^{d} C_{n,1:j} \geq c_j) = \mathbb{P}(D_{n,1:d}).$$

In other words, we are interested in computing the probability of satisfying all d bin conditions.

Definition 2. *Assuming n independent random variables, we let $n, d, \boldsymbol{x}, \mathcal{C}$ be as in Equation* (1) *where $X_i \sim F_i$ for all $i \in [n]$. With $x_0 = -\infty$, $x_{d+1} = \infty$, and $\boldsymbol{p} = (p_{i,j})_{i\in[n], j\in[d^+]} = (F_i(x_j) - F_i(x_{j-1}))_{i\in[n], j\in[d^+]}$, we define the combinatorial problem* IJCDFOS$(\mathcal{C}, \boldsymbol{p})$ *as computing the probability $\mathbb{P}(D_{n,1:d})$.*

This problem statement is fairly general and allows for both continuous and discrete distributions, with the only major assumption being independence. When the independence assumption is relaxed, the definition will change slightly and we will need to construct a corresponding graphical model with corresponding ball throwing order—a schedule. We reserve the details for the later sections, as much of the notation is unnecessary and may detract from the intuitions gained from the solution to the independent variables problem.

3 Our Solution

The primary goal of this paper is to improve the performance of Boncelet Jr.'s algorithm using combinatorial simplifications. Boncelet Jr.'s algorithm's time and space complexity are exponential in d with base n making it viable only in cases where d or n are both small. While our proposed procedure will remain exponential in d like Boncelet Jr.'s algorithm, there is still significant improvement in both space and time complexity. To that end, we first describe both pieces in order to understand their interaction.

3.1 Boncelet Jr.'s Algorithm

At a high level, Boncelet Jr.'s algorithm to solve $\textsc{ijcdfos}(\mathcal{C},\boldsymbol{p})$ maintains probability tables describing the number of balls in each bin. Using our terminology, the algorithm recursively updates a table of probabilities of $\boldsymbol{C}_i = \boldsymbol{k}$:

Definition 3. *Let $\mathcal{C}$ and $\boldsymbol{p}$ be as in $\textsc{ijcdfos}(\mathcal{C},\boldsymbol{p})$. We define $e_j = (0,\ldots,1,\ldots,0)$ as a length $d+1$ vector of 0's with a 1 in the jth position. Then by the law of total probability, we have:*

$$\mathbb{P}(\boldsymbol{C}_i = \boldsymbol{k}) = \sum_{\{j \mid C_j > 0\}} p_{i,j}\mathbb{P}(\boldsymbol{C}_{i-1} = \boldsymbol{k} - e_j).$$

For $\mathcal{I} \doteq \{\boldsymbol{i} \in [n*]^{d+1} \mid \sum_{j'=1}^{j} i_{j'} \geq c_j, \forall j \in [d]\}$, the probability of the corresponding $\textsc{ijcdfos}$*is given by $\mathbb{P}(D_{1:d}) = \sum_{\boldsymbol{k}\in\mathcal{I}} \mathbb{P}(\boldsymbol{C}_n = \boldsymbol{k})$.*

Using above recurrence, Boncelet Jr.'s algorithm maintains a table $T_i : [n]^{d+1} \to [0,1]$ of the probabilities of $\mathbb{P}(\boldsymbol{C}_i = \boldsymbol{k})$ as a function of table T_{i-1}. It then sums over the entries in $T_n(\boldsymbol{k})$ such that $\boldsymbol{C}_n = \boldsymbol{k}$ satisfies the bin conditions $D_{n,1:d}$. Since $\mid \boldsymbol{C}_i \mid = i$, the size of dynamic programming table i is given by $\binom{i+d}{d}$. Using the Hockey Stick identity, the total number of updates is given by:

$$\sum_{i=0}^{n} \binom{i+d}{d} = \binom{n+d+1}{n}$$

Each of these updates requires summing over $O(d)$ terms, yielding time complexity $O(d\binom{n+d+1}{n})$. The space complexity is $O(\binom{n+d}{n})$ as table i can be discarded once table $i+1$ is computed. We further note that this algorithm extends to the dependent setting as well, though a modification must be made to track the locations of certain balls required to obtain subsequent conditional distributions. This generalization incurs additional space and computation, though we defer the complexity analysis to the dependent variables section.

3.2 Combinatorial Simplification

In this section, we compress Boncelet Jr.'s probability tables to contain only information relevant to computing the joint cumulative distribution. Indeed, our new method maintains significantly smaller tables at the cost of only a slightly more computationally expensive updating function. This improvement is accomplished by instead tracking the distribution of a transformation $S : [n^*]^{d+1} \to [n^*]^d$ of the $\boldsymbol{C}_i$'s operating on ball count configurations that allows us to ignore the exact placement of "superfluous" balls. In accordance with our goal of efficiently computing $\textsc{ijcdfos}(\mathcal{C},\boldsymbol{p})$, this S must have the following properties:

1. The distribution of $S(\boldsymbol{C}_n)$ is easier to maintain than that of $\boldsymbol{C}_n$.

2. We must be able to compute $\mathbb{P}(D_{n,1:d})$ given only the distribution of $S(\boldsymbol{C}_n)$.

Motivating our choice of S are the following two observations. First, letting $S_{1:j}(\boldsymbol{k})$ denote the sum of the first j entries of $S(\boldsymbol{k})$, then $S_{1:j}(\boldsymbol{C}_n) \leq C_{n,1:j}$ and $c_j \leq S_{1:j}(\boldsymbol{C}_n)$ implies $D_{n,j}$. In other words, if the some of the balls in $\boldsymbol{C}_n$ are spilled over to the right—i.e. relocated to a higher indexed bin—then any bin condition satisfied by this right-shifted ball count configuration will also be satisfied in the original. With this, we define this transform such that when there are at least $\delta_j = c_j - c_{j-1}$ balls in bin I_j for $j \in [d]$, any additional balls that fall (or are spilled into) in bin I_j are spilled into bin I_{j+1}.

Definition 4. *Let $\mathcal{C},\boldsymbol{p}$ be as in $\textsc{ijcdfos}(\mathcal{C},\boldsymbol{p})$ and assume $c_0 = 0$ and $\delta_{j+1} = n$. We define the spilling transformation $S : [n^*]^{d+1} \to \bigotimes_{j\in d+1}[\delta_j^*]$ as follows:*

$$S_j(\boldsymbol{k}) = \min\,(\delta_j, \max_{j'\in[j]}\,(\sum_{i=j'}^{j} k_i - \sum_{i=j'}^{j-1} \delta_i)).$$

With this definition of S, we can show equivalence between the event that $S(\boldsymbol{C}_n) = (\delta_1, \ldots, \delta_d)$ and bin conditions $D_{n,1:d}$.

Theorem 1. *For any $\boldsymbol{C}_n \in [n]^d$ and $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_{d+1})$, we have that $D_{n,1:d}$ holds if and only if $S(\boldsymbol{C}_n) = \boldsymbol{\delta}$. Hence $\mathbb{P}\left(S(\boldsymbol{C}_n)) = \boldsymbol{\delta}\right) = \mathbb{P}(D_{n,1:d})$.*

Proof. We first start with the forward direction. Assume that $D_{n,1:d}$ holds. By definition,

$$D_{n,1:d} \leftrightarrow \bigcap_{j=1}^{d} \{C_{n,1:j} \geq c_j\} \leftrightarrow \bigcap_{j=1}^{d} \{C_{n,1:j} \geq \sum_{i=1}^{j} \delta_i\}.$$

As $(\sum_{i=1}^{j} C_{n,i} - \delta_i)$ is non-negative,

$$\delta_j \leq (\sum_{i=1}^{j} C_{n,i} - \delta_i) + \delta_j \leq \max_{j' \in [j]} (C_{n,j:j'} - \sum_{i=j'}^{j-1} \delta_i).$$

Following this, we have for all $j \in [d]$:

$$S_j(\boldsymbol{C}_n) = \min(\delta_j, \max_{j' \in [j]} (C_{n,j':j} - \sum_{i=j'}^{j-1} \delta_i)) = \delta_j.$$

Now we prove the backwards direction; assume that $S_j(\boldsymbol{C}_n) = \delta_j$ for all j. We will show that $\bigcap_{i=1}^{j} \{S_i(\boldsymbol{C}_n) = \delta_i\}$ implies $D_{n,1:j}$ by induction. Starting with the base case $j = 1$:

$$\begin{aligned}\delta_1 = S_1(\boldsymbol{C}_n) = \ & \min(\delta_1, \max_{j' \in [1]} (C_{n,j':1} - \sum_{i=j'}^{0} \delta_i)) \\ & \geq \min(\delta_1, C_{n,1}).\end{aligned}$$

Thus, $C_{n,1} \geq \delta_1 = c_1$ and the base case holds. For the recursive case:

$$\begin{aligned}\delta_j = S_j(\boldsymbol{C}_n) &= \min(\delta_j, \max_{j' \in [j]} (C_{n,j':j} - \sum_{i=j'}^{j-1} \delta_i)) \\ \implies \delta_j &\leq \max_{j' \in [j]} (C_{n,j':j} - \sum_{i=j'}^{j-1} \delta_i).\end{aligned}$$

Adding $\sum_{i=1}^{j-1} \delta_i$ to both sides, we have:

$$\begin{aligned}\delta_j + \sum_{i=1}^{j-1} \delta_i &\leq \max_{j' \in [j]} (C_{n,j':j} + \sum_{i=1}^{j'-1} \delta_i) \\ &= \max_{j' \in [j]} (C_{n,j':j} + c_{j'-1}) \\ &= \max_{j' \in [j]} (C_{n,1:j} - (C_{n,1:j'-1} - c_{j'-1})).\end{aligned}$$

However, by strong induction we know that $C_{n,1:j'} \geq c_{j'}$ for all $j' < j$, thus the last term is always non-negative. Hence, $j' = 1$ corresponds to the maximum and as $\delta_j + \sum_{i=1}^{j-1} \delta_i = c_j$, we finish with:

$$c_j \leq \max_{j' \in [j]} (C_{n,1:j} - (C_{n,1:j'-1} - c_{j'-1})) = C_{n,1:j}.$$

□

Having established $\mathbb{P}\left(S(\boldsymbol{C}_n)) = \boldsymbol{\delta}\right) \equiv \mathbb{P}(D_{n,1:d})$, all that remains is to actually compute the probability of the former. In particular, we need to be able to compute this without implicitly computing the distribution of $\boldsymbol{C}_n$, as this would amount to the work that Boncelet Jr.'s algorithm already does. One approach is instead of recursively maintaining the distribution of $\boldsymbol{C}_i$ like in Boncelet Jr.'s algorithm, we instead track the distribution of $S(\boldsymbol{C}_i)$. Using the spilling analogy of S as motivation, the recurrence relation comes naturally. To help with this, we define an additional set-valued function $\sigma(j, \boldsymbol{\kappa}) = \{j\} \cup j' < j : \bigcap_{i=j'}^{j-1} \{\kappa_i = \delta_i\}$ to denote the set of bins such that when a ball is thrown into any of these bins given $S(\boldsymbol{k}) = \boldsymbol{\kappa}$, the ball will land in or spill over into I_j under transformation S. Note that to avoid confusion we use $\boldsymbol{k} \in [n^*]^{d+1}$ and $\boldsymbol{\kappa} \in \bigotimes_{j=1}^{d+1} [\delta_j^*])$. The only way to obtain the event $S(\boldsymbol{C}_i) = \boldsymbol{\kappa}$ is if $S(\boldsymbol{C}_{i-1}) = \boldsymbol{\kappa} - e_j$ and ball i falls inside $\sigma(j, \boldsymbol{\kappa})$ with $S_j(\boldsymbol{C}_i) < \delta_j$. With this observation, and the law of total probability, we can formally state our key recurrence relation to obtain $\mathbb{P}(S(\boldsymbol{C}_i) = \boldsymbol{\kappa})$ as a function of the distribution of $S(\boldsymbol{C}_{i-1})$.

Theorem 2. *Let $\mathcal{C}, \boldsymbol{p}$ be as in* IJCDFOS$(\mathcal{C}, \boldsymbol{p})$ *and define S to be as in Definition 4. Then for $p_{i,j,\boldsymbol{\kappa}} = \sum_{j' \in \sigma(j,\boldsymbol{\kappa})} p_{i,j'}$:*

$$\mathbb{P}(S(\boldsymbol{C}_i) = \boldsymbol{\kappa}) = \sum_{j=1}^{d+1} \mathbb{P}(S(\boldsymbol{C}_{i-1}) = \boldsymbol{\kappa} - e_j) p_{i,j,\boldsymbol{\kappa}}. \tag{1}$$

3.3 Algorithm and Performance Analysis

We begin by defining dynamic programming tables $T_i : \bigotimes_{j \in [d+1]} [\delta_j^*] \to [0, 1]$ for $i \in [n^*]$ with the intention that with the recurrence relation from above, we have $T_i(\boldsymbol{\kappa}) = \mathbb{P}(S(\boldsymbol{C}_i) = \boldsymbol{\kappa})$. We describe our procedure in Algorithm 1.

The total number of entries to be updated in our algorithm is upper bounded as follows:

$$\sum_{i=1}^{n} | \{\boldsymbol{\kappa} : \kappa_{1:j} = i, \boldsymbol{\kappa} \leq \boldsymbol{\delta}\} | = O(n \prod_{j=1}^{d} \delta_j) \tag{2}$$

Algorithm 1 Spilling Algorithm to solve IJCDFOS$(\mathcal{C}, \boldsymbol{p})$

```
Input: $n, d \in \mathbb{N}, n \geq d, \mathcal{C} = (c_1, \ldots, c_d) \in [n]^d, \boldsymbol{p} \in [0,1]^{n \times (d+1)}$
Output: $\mathbb{P}(D_{n,1:d})$
1: $\delta_j = c_j - c_{j-1}, \forall j \in [d]$ for $c_{j-1} = 0$
2: Define tables $T_i : \bigotimes_{j \in d} [\delta_j^*] \to [0,1], \forall i \in [n^*]$
3: Initialize $T_0(\boldsymbol{\kappa}) \leftarrow 1$ if $\boldsymbol{\kappa} = \mathbf{0}$, else 0 for $\boldsymbol{\kappa} \in \bigotimes_{j \in d} [\delta_j^*]$
4: for $i \leftarrow 1$ to $n$ do
5:     $T_i(\boldsymbol{\kappa}) = \sum_{j=1}^{d+1} T_{i-1}(\boldsymbol{\kappa} - e_j) \sum_{j' \in \sigma(j, \boldsymbol{\kappa})} p_{i,j'}$
6: end for
7: return $\mathbb{P}(D_{n,1:d}) = T_n(\boldsymbol{\delta})$ for $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_d)$
```

We note that this upper bound ignores the constraint $\kappa_{1:j} = i$. While this was the primary constraint in the complexity analysis of Boncelet Jr.'s algorithm, it only complicates that of our own algorithm. Each entry of table T_i for $i \in [n]$ requires summing over $O(d^2)$ terms as per Theorem 2, which yields space and time complexities of $O(\prod \delta_j)$ and $O(nd^2 \prod_{j \in d} \delta_j)$ respectively. The worst case improvement corresponds to when the order statistics given by $\mathcal{C}$ are evenly spaced, which yields $\delta_j \approx \frac{n}{d}$. The space and time complexities are $O(n(n+d)^d d^{-d})$ and $O(n(n+d)^d d^{2-d})$ respectively. We can now take performance ratios of a lower bound on Boncelet Jr.'s time complexity versus that of Algorithm 1 to obtain at least a $O((\frac{n+d}{n})^n)$ and $O((\frac{n+d}{n})^n d^{-2} n^{-1})$ factor of improvement in space and time complexities respectively.

One possible modification to our algorithm precomputing and store the values of $\sum_{j \in \sigma(j', \boldsymbol{\kappa})} p_{i,j}$, which saves a factor of d computations, at a cost of a factor of d memory. Regardless of the algorithm modification, the spilling algorithm is significantly more efficient than Boncelet Jr.'s in cases where the desired order statistic indices are close together.

3.4 Extension to Two-Sided Bounds

In the previous section, we considered one-sided bounds on the order statistics of the form $\mathbb{P}(\bigcap X_{(c_j)} \leq x_j)$. We can straightforwardly generalize ours (and Boncelet Jr.'s) method to two-sided bounds of the form $\mathbb{P}(\bigcap x_j^- \leq X_{(c_j)} \leq x_j^+)$. The primary insight in this case is that, similar to $X_{(c_j)} \leq x_j^+$ implying a (loose) lower bound of c_j on the number of balls less than x_j^+, $x_j^- \leq X_{(c_j)}$ implies an (strict) upper bound of c_j on the number of balls less than x_j^-. With this in mind, the modification is as straightforward to maintain the distribution of the balls w.r.t. the new lower bounds simultaneously. In particular, in addition to T containing the usual joint distribution of the number of balls in the bins defined by $\boldsymbol{x}^+$ which we call $\boldsymbol{C}^+$, we also maintain the distribution of balls over the bins defined by $\boldsymbol{x}^-$, which we call $\boldsymbol{C}^-$.

Since Boncelet Jr.'s algorithm tracks the exact location of each ball, the generalization is quite simple. We construct a partition $\boldsymbol{x}$ of $\mathbb{R}$ with the partitions at the points in $\boldsymbol{x}^-$ and $\boldsymbol{x}^+$. We assume that $\boldsymbol{x}^- < \boldsymbol{x}^+$ element-wise, as well as $\boldsymbol{x}^-$ and $\boldsymbol{x}^+$ are sorted. With $x_0 = -\infty$ and $x_{2d+1} = \infty$, we construct our probability vector $\boldsymbol{p} = (p_{i,j} = \mathbb{P}(X_i \in (x_{j-1}, x_j)))_{i \in [n], j \in [2d+1]}$. Boncelet Jr.'s two-sided algorithm proceeds by simultaneously updating the distributions of $\boldsymbol{C}_i^+$ and $\boldsymbol{C}_i^-$ according to $\boldsymbol{p}$ and T_{i-1}. In particular, we define function $\phi^- : [2d+1] \to [d+1]$ such that $\phi^-(j) = \sum_{j'=1}^{j} 1_{x_j \in \boldsymbol{x}^-}$ denotes the index of the bin defined by $\boldsymbol{x}^-$ that corresponds to the j'th bin defined by $\boldsymbol{x}$. We define ϕ^+ similarly. We have the generalized updating step:

Theorem 3 (**Boncelet Jr. Two Sided**). *Let T_i denote the joint distribution of the configuration of the first i balls w.r.t. both $\boldsymbol{x}^-$ and $\boldsymbol{x}^+$— $T_i(\boldsymbol{k}_i^-, \boldsymbol{k}_i^+) = \mathbb{P}(\boldsymbol{C}_i^- = \boldsymbol{k}_i^-, \boldsymbol{C}_i^+ = \boldsymbol{k}_i^+$. Then,*

$$T_i(\boldsymbol{k}_i^-, \boldsymbol{k}_i^+) = \sum_{j=1}^{2d+1} p_{i,j} T_{i-1}(\boldsymbol{k}_i^- - e_{\phi^-(j)}, \boldsymbol{k}_i^+ - e_{\phi^+(j)})$$

For the final step where instead of summing over the set $\mathcal{I}^- \times \mathcal{I}^+$ where $\mathcal{I}^- \doteq \{\boldsymbol{i} \in [n*]^{d+1} \mid \sum_{j'=1}^{j} i_{j'} \geq c_j, \forall j \in [d]\}$ and $\mathcal{I}^+ \doteq \{\boldsymbol{i} \in [n*]^{d+1} \mid \sum_{j'=1}^{j} i_{j'} < c_j, \forall j \in [d]\}$, the probability of the corresponding IJCDFOSis given by $\sum_{(\boldsymbol{k}^-, \boldsymbol{k}^+) \in \mathcal{I}^- \times \mathcal{I}^+} \mathbb{P}(\boldsymbol{C}_n^- = \boldsymbol{k}^-, \boldsymbol{C}_n^+ = \boldsymbol{k}^+)$. As in the single-sided case, we can use a similar spilling function to speed up Boncelet Jr.'s two-sided algorithm. In particular, we maintain both the distribution of $S^-(\boldsymbol{k}_i^-) = \boldsymbol{\kappa}_i^-$ which is the right-wards spilling transform of $\boldsymbol{k}_i^-$ and $S^+(\boldsymbol{k}_i^+) = \boldsymbol{\kappa}_i^+$ which is the *left-wards* spilling transform of $\boldsymbol{k}_i^+$. Note that for $\boldsymbol{\kappa}_i^+$, we spill leftwards to maintain an *upper bound* on the number of balls in the first j bins, as opposed to a lower bound. We define $\sigma^-(j, \boldsymbol{\kappa}^-) = \{j\} \cup j' < j : \bigcap_{i=j'}^{j-1} \{\kappa_i^- = \delta_i\}$ denote the set of bins that will right-spill into (x_{j-1}^-, x_j^-), where $\boldsymbol{\kappa}^-$ is the transformed ball distribution with respect to $\boldsymbol{x}^-$. We define $\sigma^+(j, \boldsymbol{\kappa}^+) = \{j\} \cup j' > j : \bigcap_{i=j+1}^{j'} \{\kappa_i^+ = \delta_i\}$ analogously, with $p_{i,j,\boldsymbol{\kappa}^-,\boldsymbol{\kappa}^+} \equiv \sum_{j' \in \{\sigma^-(j,\boldsymbol{\kappa}^-) \cap \sigma^+(j,\boldsymbol{\kappa}^+)\}} p_{i,j'}$. We can now define the two-sided spilling algorithm update step.

Theorem 4 (**Spillover Two-Sided**). *Let T_i de-*

(Two Sided)	Time Complexity	Space Complexity
Boncelet Jr.	$O(d\binom{n+d+1}{n}^2)$	$O(\binom{n+d}{n}^2)$
Spillover	$O(nd^2\prod_{j=1}^d \delta_j^2)$	$O(\prod_{j=1}^d \delta_j^2)$

Table 1: **Two-Sided Complexity Comparison**. The complexity analysis is somewhat difficult as it depends on the relationship and overlap between $\boldsymbol{x}^-$ and $\boldsymbol{x}^+$. That is, $\boldsymbol{C}^+$ and $\boldsymbol{C}^-$ are intimately related, so the total number of entries to be updated is significantly smaller than the product of the total number of possible $\boldsymbol{C}^+$ and $\boldsymbol{C}^-$. Because of this additional complication, we give rather loose upper bounds on the space and time complexity of both generalized Boncelet Jr.'s method assuming that $\boldsymbol{C}^-$ and $\boldsymbol{C}^+$ are independent.

(Markov)	Time Complexity	Space Complexity
Boncelet Jr.	$O(K\binom{n+d+1}{n})$	$O(K\binom{n+d}{n})$
Spillover	$O(ndK\prod_{j=1}^d \delta_j)$	$O(K\prod_{j=1}^d \delta_j)$

Table 2: **Dependent Case Complexity Comparison for Markov Chains**. We provide a complexity analysis for the Markov dependent random variables extension of Boncelet Jr.'s and our Spillover algorithms.

note the joint distribution of the transform of the configuration of the first i balls w.r.t. both $\boldsymbol{x}^-$ and $\boldsymbol{x}^+$—$T_i(\boldsymbol{\kappa}_i^-, \boldsymbol{\kappa}_i^+) = \mathbb{P}(S^-(\boldsymbol{C}_i^-) = \boldsymbol{\kappa}_i^-, S^+(\boldsymbol{C}_i^+) = \boldsymbol{\kappa}_i^+$. Then, $T_i(\boldsymbol{\kappa}^-, \boldsymbol{\kappa}^+)$ is given by

$$\sum_{j=1}^{2d+1} T_{i-1}(\boldsymbol{\kappa}_i^- - e_{\phi^-(j)}, \boldsymbol{\kappa}_i^+ - e_{\phi^+(j)}) p_{i,j,\boldsymbol{\kappa}_i^-,\boldsymbol{\kappa}_i^+}$$

3.5 Extension to Dependent Variables

One of Boncelet Jr.'s algorithm primary advantages is its ability to handle dependent random variables, albeit with additional memory and computational cost. This flexibility can be especially useful in the analysis of Markov processes, hidden Markov models, or autoregressive models which are commonly used for risk modeling in insurance or financial mathematics. The generalization to dependent random variables requires some changes to the information stored in the dynamic programming tables, as we will need to understand the conditional distribution of future balls yet to be thrown given the placement of the balls already thrown. Of course, the task of tracking the exact placement of each ball was one that we wanted to avoid in the first place. We show that for a fixed Markov Random Field structure, we can limit our memory of exact placement to a small subset of the balls. We also restrict ourselves to the simple case of Markov random variables, though we provide the arbitrary dependency structure case in the appendix.

1. For simplicity, let $X_i, x_j^-, x_j^+ \in \mathbb{Z}$ for all i, j. We also assume lower and upper bounds on the range of the X_i's of 1 and K. Define $\psi : [K] \to [d+1]$ such that $\psi(k) = \min_j\{j : k \leq c_j\}$ denotes the index of the bin that k belongs to.
2. Let $p : [n] \times [K] \to [0,1]^K$ with $p_{i,k}$ denoting the conditional distribution of X_i over $[K]$ given that $X_{i-1} = k$.
3. Let $\mathbf{1}_{j,k,\boldsymbol{\kappa}} \equiv \mathbf{1}_{\boldsymbol{\kappa}_{\psi(k):j-1} = \boldsymbol{\delta}_{\psi(k):j-1}}$ for $j \in [d], k \in [K], \boldsymbol{\kappa} \in \bigotimes_{j=1}^{d+1}[\delta_j^*]$.

In conjunction with our existing combinatorial setup, we are ready to define both the key recursions in the dependent generalization of Boncelet Jr.'s algorithm and our Spillover algorithm.

Theorem 5 (Boncelet Jr. Markov)**.** *For all $\boldsymbol{k}_i \in [n^*]^{d+1}$ and $k \in [K^*]$ such that $\{\boldsymbol{C}_i = \boldsymbol{k}_i\}$ and $\{X_i = k\}$ are non-disjoint events, we have that with $T_i(\boldsymbol{k}_i, k) = \mathbb{P}(\boldsymbol{C}_i = \boldsymbol{k}_i, X_i = k)$,*

$$T_i(\boldsymbol{k}_i, k) = p_{i,k} T_{i-1}(\boldsymbol{k}_i - e_{\psi(k)}, X_i = k)$$

Theorem 6 (Spillover Markov)**.** *For all $\boldsymbol{\kappa}_i \in \bigotimes_{j=1}^{d+1}[\delta_j^*]$ and $k \in [K^*]$ such that $\{S(\boldsymbol{C}_i) = \boldsymbol{\kappa}_i\}$ and $\{X_i = k\}$ are non-disjoint events, we have that with $T_i(\boldsymbol{\kappa}_i, k) = \mathbb{P}(S(\boldsymbol{C}_i) = \boldsymbol{\kappa}_i, X_i = k)$,*

$$T_i(\boldsymbol{\kappa}_i, k) = \sum_{j=\psi(k)}^{d+1} p_{i,k} T_{i-1}(\boldsymbol{\kappa}_i - e_j, X_i = k)\mathbf{1}_{j,k,\boldsymbol{\kappa}_i}$$

3.6 Experiments

To verify our algorithm's performance guarantees, we directly compare with Boncelet Jr.'s algorithm. We conduct two experiments, the first of which varies the values of $n, d, \mathcal{C}$ in the independent case. Then, for the experiments under the dependent setup, we have random variables $X_1, \ldots, X_n$, where X_i's state space is $\{-i, -i+1, \ldots, i-1, i\}$. In the independent setting, $X_i = \nu_i$ where ν_i takes value in $\{-1, 0, 1\}$ with equal probability. In the setting $b(*) = 1$, the random variables are Markov such that $X_{i+1} = X_i + \nu_i$ with $X_0 = 0$. For $b(*) = 2$, we let $X_{i+2} = X_i + \nu_i$ with $X_0, X_1 = 0$. As in the independent random variables section, we implement the version of both algorithms that disregard irrelevant intermediary entries that cannot satisfy $D_{1:d}$. For

n	6	12	18	24	30
$d = 1$	6.9E-5	1.1E-4	2.0E-4	1.7E-4	1.8E-4
$d = 2$	8.5E-4	1.6E-4	2.3E-4	2.7E-4	3.7E-4
$d = 3$	1.8E-4	3.1E-4	4.4E-4	5.8E-4	7.6E-4
$d = 4$	3.4E-4	6.4E-4	9.5E-4	1.4E-3	1.6E-3
$d = 5$	7.2E-4	1.4E-3	2.1E-3	2.8E-3	3.7E-3
$d = 6$	1.5E-3	3.0E-3	5.6E-3	6.5E-3	7.9E-3

Table 3: **Spillover Algorithm Elapsed Time**. We ran an experiment from Galgana and Shi, et. al. (2021) with $n \in \{6, 12, 18, 24, 30\}$ and $\mathcal{C} \in \{[d]\}$ for $d \in [6]$.

n	6	12	18	24	30
$d = 1$	1.1E-4	4.0E-4	5.8E-4	1.1E-3	1.6E-3
$d = 2$	6.4E-4	2.8E-3	7.3E-3	1.7E-2	3.0E-2
$d = 3$	1.3E-3	1.4E-2	6.1E-2	1.7E-1	4.1E-1
$d = 4$	3.8E-3	6.5E-2	3.8E-1	1.4E0	4.0E0
$d = 5$	9.4E-3	2.4E-1	2.0E0	9.4E0	3.2E1
$d = 6$	2.1E-2	8.4E-1	8.8E0	5.3E1	2.2E2

Table 4: **Boncelet Jr.'s Algorithm Elapsed Time**. Our algorithm is significantly faster than the improved version of Boncelet Jr.'s especially for larger n or d.

the first experiment, we fix $n = 25$ and vary $d \in [12]$ and $\mathcal{C} = \{1, \ldots, d\}$. In the second experiment, we fix $d = 2$, $\mathcal{C} = \{\frac{n}{2}, n\}$ and vary $n \in [128]$. The experiments were run a computer with an Intel Xeon E3-1240V5 CPU with 15.6 GB of memory.

4 Conclusion

In this paper, we describe the computation of the joint cdf of d order statistics of n random variables; in particular, we focus on the method provided in [**?**], and show how the computational complexity of Boncelet Jr.'s algorithm can be improved by compressing dynamic programming tables to store only information pertinent to computing the cdf rather than the entire joint distribution of the ball configurations. We extend our procedure to handle dependent random variables and show how this affects the space and time complexity. In the worst case of evenly spread d order statistics, our algorithm improves the space and time complexity over Boncelet Jr.'s by a factor of at least $O((\frac{n+d}{n})^n)$ and $O((\frac{n+d}{n})^n d^{-2} n^{-1})$ respectively.

5 Appendix

One of the key advantages to using Boncelet Jr.'s algorithm over its competitors is its flexibility to handle dependent random variables. Specifically, Boncelet Jr.'s algorithm allows for dependency between random variables, albeit with additional memory and computational cost. This flexibility can be especially useful for various stochastic processes such as Markov processes or more general autoregressive models A common application of computing order statistic distributions for dependent random variables is estimating the maximum length of a queue. The generalization to dependent random variables requires some changes to the information stored in the dynamic programming tables, as we will need to understand the conditional distribution of balls yet to be thrown given the placement of the balls already thrown. Of course, the task of tracking the exact placement of each ball was one that we wanted to avoid in the first place. In this section, we show that, in cases of sparse random variable dependency structures, then we can limit our memory of exact placement to a small subset of the balls.

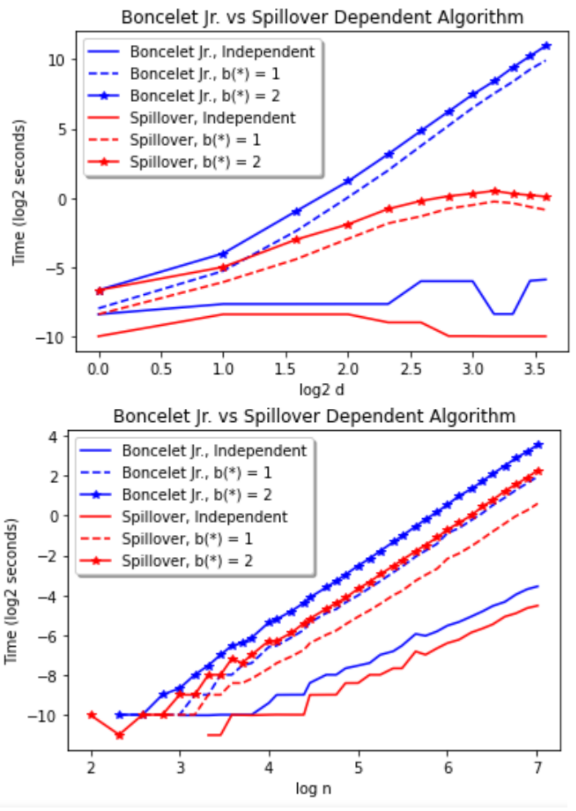

Figure 1: In the first figure, note that the slope of the Spillover algorithm is about half of Boncelet Jr.'s. The dip towards larger d, which is more noticeable for the Spillover algorithm, is due to the pruning of irrelevant intermediary entries. In the second figure, the slopes of each pair of experiments is effectively equal, though the Spillover algorithm is a constant multiplicative factor faster.

5.1 Preliminaries

Given n random variables and their joint distribution $\boldsymbol{F}$, we can construct the corresponding Markov random field (MRF). In order to minimize the number of random variables tracked in our algorithm, we utilize the local Markov property and the structure of the conditional dependencies in the MRF. We adopt most of the combinatorial notation from the previous section, however we will use an additional partitioning of each bin to simulate exact placement of each random variable. Letting node i of the MRF represent variable X_i: [1]

1. Define micro-bins and micro-bounds $I_{j,h}$ and $x_{j,h}$ for $j \in [d^+]$ and $h \in [H]$, for granularity factor $H \in \mathbb{N}$. Here, $x_j = x_{j,0} \leq \ldots, \leq x_{j,H} = x_{j+1}$ and $I_{j,h} = (x_{j,h-1}, x_{j,h}]$.

2. Define the vector $\mathcal{N}_i \in ([d^+] \times [H])^{n(i)}$ to denote the micro-bin locations of the neighboring set of i—the lower-indexed neighbors of ball i. By the local Markov property, i is conditionally independent of $\{1, \ldots, i-1\}$ given $\mathcal{N}_i$. Here, $n(i)$ denotes the number of lower-indexed neighbors of i.

3. Define the vector $\mathcal{B}_i \in ([d^+] \times [H])^{b(i)}$ to denote the micro-bin locations of the boundary set of i—the neighbors of balls $i, \ldots, n$ among the first $i-1$ balls. Again by the local Markov property, balls i through n are conditionally independent of $\{1, \ldots, i-1\}$ given $\mathcal{B}_i$. Here, $b(i)$ denotes the number of neighbors of balls i through n among the first $i-1$ balls.

4. Define the vector-valued transform $N_i : ([d^+] \times [H])^{b(i)} \to ([d^+] \times [H])^{n(i)}$ to select only the entries of $\mathcal{B}_i \in ([d^+] \times [H])^{b(i)}$ corresponding to the neighbors of i, namely $\mathcal{N}_i$. That is, $N_i(\mathcal{B}_i) = \mathcal{N}_i$.

5. Define $\boldsymbol{p}_i(\cdot) \doteq \boldsymbol{p}_i(\cdot \mid N_i(\mathcal{B}_i) = \boldsymbol{j}_{\mathcal{N}}) \doteq \boldsymbol{p}_i(\cdot \mid \mathcal{N}_i = \boldsymbol{j}_{\mathcal{N}})$ to be the conditional distribution of ball i given $\{\mathcal{N}_i = \boldsymbol{j}_{\mathcal{N}}\}$. Here, $\boldsymbol{j}_{\mathcal{N}} \in ([d^+] \times [H])^{n(i)}$. In practice, $p_i(\cdot)$ can be estimated through MCMC-sampling methods or by numerical integration and marginalizing out its higher-indexed neighbors. If the random variables are discrete, it may be computed exactly.

[1] We can instead let node i represent a different variable v_i, which creates a different visitation scheme $\boldsymbol{v}$, neighbor sets, and boundary sets. Consequently, the choice of visitation scheme also impacts the algorithm's space and time complexity, and in general, computing the optimal re-ordering is NP-hard. For simplicity, we stick to the original ordering.

With our new notation, we can formally define the dependent version of IJCDFOS$(\mathcal{C}, \boldsymbol{p})$.

Definition 5. *Assuming random variables $X_1, \ldots, X_n$, we let $n, d, \boldsymbol{x}, \mathcal{C}, \boldsymbol{F}$ be as in Definition (1). We define $\boldsymbol{p} = (p_{i,j,h}(\boldsymbol{j}_{\mathcal{N}}))_{i\in[n], j\in[d^+], h\in[H], \boldsymbol{j}_{\mathcal{N}}\in([d^+]\times[H])^{n(i)}} = (F_i(x_j \mid \mathcal{N}_i = \boldsymbol{j}_{\mathcal{N}}) - F_i(x_{j-1} \mid \mathcal{N}_i = \boldsymbol{j}_{\mathcal{N}}))_{i\in[n], j\in[d], h\in[H], \boldsymbol{j}_{\mathcal{N}}\in([d^+]\times[H])^{n(i)}}$. That is, $p_{i,j,h}(\boldsymbol{j}_{\mathcal{N}})$ is the probability that ball i falls into micro-bin $I_{j,h}$ given $\mathcal{N}_i = \boldsymbol{j}_{\mathcal{N}}$. We define the combinatorial problem* DJCDFOS$(\mathcal{C}, \boldsymbol{p})$ *as computing the probability $\mathbb{P}(D_{n,1:d})$.*

At first, it seems there are only three additions to Algorithm 1 required to solve Equation (1) with dependent random variables. First, we must further sub-divide each bin. Second, we need our algorithm to store $\mathcal{N}_i$ in micro-bin space by iteration i. Third, we need to change our definition of $\boldsymbol{p}_i$ to reflect the dependency between the current random variable and those in $\mathcal{N}_i$. In particular, instead of having $\boldsymbol{p}_i : [d] \to [0, 1]$ as the singular distribution of the current ball over all bins, we can instead have a distribution $\boldsymbol{p}_i(\cdot) = \boldsymbol{p}_i(\cdot \mid \mathcal{N}_i = \boldsymbol{j}_{\mathcal{N}})$. However, these three changes are insufficient as the algorithm must also track the location of balls that will be needed to compute the conditional distribution of not only ball i but also $i+1, \ldots, n$. More specifically, our algorithm must not "forget" the location of the balls in $\mathcal{B}_i$ by storing the configuration in memory at each time step i.

5.2 Updated Recurrence

The updated recurrence relation is similar to Theorem (2), except now we also need to take into account $\mathcal{B}_i$ and how they both impact the distribution of ball i and also the placement of balls $i+1$ through n. Moreover, we will also benefit from defining set-valued function ψ_i that takes argument $\boldsymbol{j}_{\mathcal{B}} \in ([d^+] \times [H])^{b(i+1)}, \boldsymbol{\kappa} \in \bigotimes_{j\in d}[\delta_j^*]$ and returns the set of $\boldsymbol{j}'_{\mathcal{B}} \in ([d^+] \times [H])^{b(i)}$ such that $\{\mathcal{B}_i = \boldsymbol{j}'_{\mathcal{B}}\}$, $\{\mathcal{B}_{i+1} = \boldsymbol{j}_{\mathcal{B}}\}$, and $\{S(\boldsymbol{C}_i) = \boldsymbol{\kappa}\}$ are non-disjoint events—the ball configurations across the three events are consistent. Lastly, we must make an important distinction between when i is in the boundary set of $i+1$. As such, we define a piece-wise function γ_i that takes as arguments $\boldsymbol{j}_{\mathcal{B}} \in ([d^+] \times [H])^{b(i+1)}, \boldsymbol{j}'_{\mathcal{B}} \in ([d^+] \times [H])^{b(i)}, \boldsymbol{\kappa} \in \bigotimes_{j\in d}[\delta_j^*]$. If i is in the boundary set of $i+1$, then let j, h denote the last entry of $\boldsymbol{j}_{\mathcal{B}}$ and set $\gamma_i(\boldsymbol{j}_{\mathcal{B}}, \boldsymbol{j}'_{\mathcal{B}}, \boldsymbol{\kappa}) = p_{i,j,h}(N_i(\mathcal{B}_i) = \boldsymbol{j}'_{\mathcal{B}})$. Otherwise, we set it to $\sum_{j'\in\sigma(j,\boldsymbol{\kappa})} \sum_{h=1}^{H} p_{i,j',h}(N_i(\mathcal{B}_i) = \boldsymbol{j}'_{\mathcal{B}})$. As ball i is conditionally independent of the event

$S(\boldsymbol{C}_i) = \boldsymbol{\kappa}_i$ given $\{N_i(\mathcal{B}_i) = \boldsymbol{j}_\mathcal{N}\}$ or $\{\mathcal{N}_i = \boldsymbol{j}_\mathcal{N}\}$, the following recurrence relation follows from the law of total probability, much like in Equation (2):

Theorem 7. *Let $\mathcal{C}, \boldsymbol{p}$ be as in* DJCDFOS*$(\mathcal{C}, \boldsymbol{p})$. Define S to be as in Definition 4 and $\mathcal{N}_i, \mathcal{B}_i, \psi_i, \gamma_i$ to be as in the previous section. Then: Then:*

$$\begin{aligned} &\mathbb{P}\left(S(\boldsymbol{C}_i) = \boldsymbol{\kappa}, \mathcal{B}_{i+1} = \boldsymbol{j}_\mathcal{B}\right) \qquad (3)\\ &= \sum_{j' \in \sigma(d+1, \boldsymbol{\kappa})} \sum_{\boldsymbol{j}'_\mathcal{B} \in \psi_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{\kappa})} \mathcal{P}(i, \boldsymbol{\kappa}, \boldsymbol{j}'_\mathcal{B}) \gamma_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{j}'_\mathcal{B}, \boldsymbol{\kappa})\\ &\quad + \sum_{j=1}^{d} \sum_{\boldsymbol{j}'_\mathcal{B} \in \psi_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{\kappa})} \mathcal{P}(i, \boldsymbol{\kappa} - e_j, \boldsymbol{j}'_\mathcal{B}) \gamma_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{j}'_\mathcal{B}, \boldsymbol{\kappa}). \end{aligned}$$

Where, for brevity, we let $\mathcal{P}(i, \boldsymbol{\kappa}, \boldsymbol{j}_\mathcal{B}) = \mathbb{P}(S(C^{i-1}) = \boldsymbol{\kappa}, \mathcal{B}_i = |_\mathcal{B})$.

5.3 Algorithm and Performance Analysis

We begin by defining dynamic programming tables $T_i : \bigotimes_{j \in d} [\delta_j^*] \times ([d^+] \times [H])^{b(i)} \to [0, 1]$ for $i \in [n]$ with the intention that with the recurrence relation from above, $T_i(\boldsymbol{\kappa}, \boldsymbol{j}_\mathcal{B}) = \mathbb{P}(S(\boldsymbol{C}_i) = \boldsymbol{\kappa}, \mathcal{B}_{i+1} = \boldsymbol{j}_\mathcal{B})$. We describe our procedure in Algorithm 2.

Algorithm 2 Spilling Algorithm to solve DJCDFOS$(\mathcal{C}, \boldsymbol{p})$

```
Input: $n, d, H \in \mathbb{N}, n \geq d, \mathcal{C} \in [n]^d, \boldsymbol{p} \in [0,1]^{n \times (d+1) \times H \times (H(d+1))^{n(i)}}$
Output: $\mathbb{P}(D_{n,1:d})$
 1: $\delta_j = c_j - c_{j-1}, \forall j \in [d]$ for $c_{j-1} = 0$
 2: Define tables $T_i : \bigotimes_{j \in d} [\delta_j^*] \to [0,1], \forall i \in [n^*]$
 3: Initialize $T_0(\boldsymbol{\kappa}, \emptyset) \leftarrow 1$ if $\boldsymbol{\kappa} = \boldsymbol{0}$, else 0 for $\boldsymbol{\kappa} \in \bigotimes_{j \in d} [\delta_j^*]$
 4: for $i \leftarrow 1$ to $n$ do
 5:     for $\boldsymbol{j} \in [H(d+1)]^{b(i+1)}$ do
 6:         $\alpha = \sum_{j' \in \sigma(d+1,\boldsymbol{\kappa})} \sum_{\boldsymbol{j}'_\mathcal{B} \in \psi_i(\boldsymbol{j}_\mathcal{B},\boldsymbol{\kappa})} T_i(\boldsymbol{\kappa}, \boldsymbol{j}'_\mathcal{B}) \gamma_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{j}'_\mathcal{B}, \boldsymbol{\kappa})$
 7:         $\beta = \sum_{j=1}^{d} \sum_{\boldsymbol{j}'_\mathcal{B} \in \psi_i(\boldsymbol{j}_\mathcal{B},\boldsymbol{\kappa})} T_i(\boldsymbol{\kappa} - e_j, \boldsymbol{j}'_\mathcal{B}) \gamma_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{j}'_\mathcal{B}, \boldsymbol{\kappa})$
 8:         $T_i(\boldsymbol{\kappa}, \boldsymbol{j}_\mathcal{B}) = \alpha + \beta$
 9:     end for
10: end for
11: return $\mathbb{P}(D_{n,1:d}) = T_n(\boldsymbol{\delta}, \emptyset)$ for $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_d)$
```

Just like in Algorithm 1, table T_i is disposable once table T_{i+1} is computed. Hence, the space complexity is bounded by size of the largest table, which is of size $O((Hd)^{b(*)} \prod_{j \in d}(1 + \delta_j))$, where $b(*) = \max_{i \in [n]} b(i)$. We note that $b(*)$ is dependent on the graph visitation scheme, which we assume to be simply $1, \ldots, n$ wlog. Intuitively, the memory required grows in the size of the largest boundary set. With regards to the time complexity, we have that the updating step to compute $T_i(\boldsymbol{\kappa}, \boldsymbol{j}_\mathcal{B})$ requires a summation over $O(d)$ possible values of j', $O(|\psi_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{\kappa})|)$ values of $\boldsymbol{j}'_\mathcal{B}$, and another $O(d^2)$ possible values of j. Here, the quantity $|\psi_i(\boldsymbol{j}_\mathcal{B}, \boldsymbol{\kappa})|$ is difficult to measure, so we upper bound it simply by $O((Hd)^{\Delta_i})$, where Δ_i denotes the number of variables included in the boundary set of i but not of $i+1$. Notice that $\Delta_i \leq b(i) \leq b(*)$. With this, since there are $O((Hd)^{b(*)} \prod_{j \in d}(1 + \delta_j))$ table entries in T_i to compute for each $i \in [n]$, we have a total time complexity of $O(n(Hd)^{1+2b(*)} \prod_{j \in d}(1 + \delta_j))$. Like in the independent case, we can assume worst case with evenly spaced order statistics, which yields total space and time complexities of $O((Hd)^{b(*)} n^{d+1} d^{-d})$ and $O(n(Hd)^{1+2b(*)} n^{d+1} d^{1-d})$ respectively.

	Time Complexity
Bonc	$O((Hd)^{1+2b(*)}(n+d)^{n+d} n^{-\frac{1}{2}-n} d^{-d})$
Spill	$O((Hd)^{1+2b(*)} n(n+d)^d d^{1-d})$

Table 5: **Dependent Complexity Comparison**. H denotes the granularity factor and $b(*)$ is the size of the largest boundary set. $H = 1$ and $b(*) = 0$ when the random variables are independent.