Slice Algorithms for Counting in U-Dimensional Space

Jaiwant Mulik
Computer and Information Sciences
Delaware State University, DE
jmulik@desu.edu

Jawahar Pathak
Mathematics and Computer Science
Lincoln University, PA
jpathak@lu.lincoln.edu

Abstract

This paper presents a computationally efficient algorithm for solving the following well known die problem: Consider a "crazy die" to be a die with n faces where each face has some "cost". Costs need not be sequential. The problem is to determine the *exact* probability that the sum of costs from U throws of this die is $\geq T, T \in \mathbb{R}$. Our approach uses "slice" volume computation in U-dimensional space. Detailed algorithms, complexity analysis and comparison with traditional generating functions approach are presented.

1 Introduction

We present a computationally efficient solution to a straightforward, but intractable combinatorial problem. Consider a "crazy die" to be a die with n faces with each face "cost" as $c_i \in \mathbb{R}$, $1 \leq i \leq n$. The problem is to determine the *exact* probability that the sum of costs from U throws of this die is $\geq T, T \in \mathbb{R}$. If N is the number of ways that the sum of cost from U throws $\geq T$, then N/n^U is the required probability.

In this paper we model the sum of costs of each possible sequence of U throws as a point in a U-dimensional virtual space V and then count the number of points in V with value $\geq T$. This count is the required N. Compared to the usual techniques of using generating functions and brute force techniques, our solution is computationally more efficient.

In Section 2 we formally define the problem, notations and virtual space and prove some interesting properties of V. Section 3 presents the slice al-

JCMCC **58** (2006), pp. 169-188

gorithms that determine N from V. Section 4 compares the computational complexity of our approach to generating functions and brute force.

2 Preliminaries

Let

$$V = \{(x_1, x_2 \ldots, x_U) \mid 1 \leq x_i \leq n, i = 1 \ldots U\}$$

Define cost of a point as

$$C : V \rightarrow \mathbb{R}_+$$

$$C((x_1, x_2 \ldots, x_U)) = \sum_{i=1}^{u} c_{x_i}$$

where, $c_i \leq c_j$ if $i < j$. Hence, V is a "virtual space" in $\mathbb{R}^U$ on n components arranged such that component costs are monotonically increasing along the axes of V.

Definition 2.1. A point $J = (j_1, j_2, \ldots, j_U)$ is a *neighbor* of point $I = (i_1, i_2, \ldots, i_U)$, if $I \neq J$ and $|i_k - j_k| \leq 1$, for all k, $1 \leq k \leq U$.

Definition 2.2. A point $J = (j_1, j_2, \ldots, j_U)$ is a *smaller neighbor* of point $I = (i_1, i_2, \ldots, i_U)$ if $j_k = i_k - 1$ for exactly one value of k. Note that $(1, 1, \ldots)$ does not have a smaller neighbor.

Let $T > 0$ be any fixed real number.

Definition 2.3. A point P is called an *anchor point* (a.p.) if cost $C(P) \geq T$ and if P has smaller neighbors then the cost of at least one of those smaller neighbors is $< T$.

Figure 1 illustrates anchor points in a 2-dimensional space.

Let σ be a permutation on $\{1, 2, \ldots, U\}$ objects. We define action of σ on a point $(x_1, \ldots, x_U)$ in V as follows: $\sigma((x_1, \ldots, x_U)) = (x_{\sigma(1)}, \ldots, x_{\sigma(U)})$

Lemma 2.1. *If A is a set of anchor points, then A is permutation stable.*

Proof. If I is an anchor point and J is a small neighbor of I with cost less than T, then $\sigma(J)$ is a small neighbor of $\sigma(I)$. Since cost is invariant under this action, $\sigma(I)$ is indeed an anchor point. □

Theorem 2.2. *If $P = (a_1, a_2, \ldots, a_U)$ is an anchor point such that $P \notin \{(1, 1 \ldots), (n, n, \ldots)\}$. Then if there are other anchor points, at least one neighbor of p must also be an anchor point.*

Proof. By the definition and hypothesis, there exists a smaller neighbor S of P such that $C(S) < T$. Since anchor points are stable under permutations of coordinates, we can assume that $S = (a_1 - 1, a_2, \ldots, a_U)$. Let $Q = (a_1, a_2, a_3 + 1, \ldots, a_U)$. Since $C(P) \geq T$, $C(Q) \geq T$. Consider $Q' =$

	1	*2*	*4*	*10*
1	2	3	5	11
2	3	4	6	12
4	5	6	8	14
10	11	12	14	20

Figure 1: Anchor points: Here $c_1 = 1, c_2 = 2, c_3 = 4, c_4 = 10, U = 2, T = 5$. Anchor points are highlighted.

$(a_1-1, a_2, a_3+1, \ldots, a_U)$. Q is a neighbor of P and S is a smaller neighbor of N.

CASE I: $C(N) < T$

Then Q is an anchor point since,

1. $C(Q) \geq T$
2. Q' is a smaller neighbor of B and $C(N) < T$

So Q is a neighbor of P and Q is an anchor point.

CASE II: $C(Q') \geq T$

Then Q' is an anchor point since,

1. $C(Q') \geq T$.
2. S is a smaller neighbor of Q' and $C(S) < T$.

So Q' is a neighbor of P and Q' is an anchor point. □

Lemma 2.3. *Suppose $I = (i_1, i_2, \ldots, i_U)$ and $J = (j_1, j_2, \ldots, j_U)$ are two points in V, $I \neq J$ and A is the set of anchor points. If $i_k > j_k$, $\forall k \leq U$, then either $I \notin A$ or $J \notin A$ (both I and J cannot be anchor points).*

Proof. Suppose I is an anchor point. By definition there is k such that the cost of $(i_1, \ldots, i_k - 1, \ldots, i_U)$ is less than T. Since $i_k > j_k, \forall k \leq U$, cost of J is also less than T. □

Lemma 2.4. *Suppose $(i_1, i_2, \ldots, i_U)$ and $(i_1, \ldots, i_l + r, \ldots, i_U)$ are anchor points, then $(i_1, \ldots, i_l + s, \ldots, i_U)$ is an anchor point $\forall s$, $1 \leq s < r$.*

Proof. From definition of anchor points and construction of virtual space described above, it is clear that $\forall s$, $1 \leq s < r$, $C(i_1, \ldots, i_l + s, \ldots, i_U) \geq T$. Now, $(i_1, \ldots, i_l + r, \ldots, i_U)$ has a smaller neighbor say $N = (i_1, \ldots, i_j -$

$1, \ldots, i_l + r, \ldots, i_U)$ ($j > l$, is also possible, and does not affect this proof), where $C(N) < T$ (required by definition of an anchor point). Now, $N_1 = (i_1, \ldots, i_j - 1, \ldots, i_l + r - 1, \ldots, i_U)$ is a smaller neighbor of $(i_1, \ldots, i_l + r - 1, \ldots, i_U)$. Also since $C(N) < T$, $C(N_1) < T$. Hence $(i_1, \ldots, i_l + r - 1, \ldots, i_U)$ is an anchor point. Inductively we can now prove that all points $(i_1, \ldots, i_l + s, \ldots, i_U)$ $1 \leq s < r$ are anchor points. □

Definition 2.4. We say that a subset X of V is *connected* if for any $I, I' \in X$, there exist $I_1, I_2, \ldots, I_r \in X$ such that I_1, I and I_r, I' are neighbors, and for $1 \leq j \leq r - 1$, I_j, I_{j+1} are neighbors. $\{I, I_1, \ldots, I_r, I'\}$ is called the *connected path.*

Lemma 2.5. *Suppose X is any connected set that contains points I, J such that $C(I) < T$ and $C(J) \geq T$, then X contains an anchor point or X contains a neighbor of an anchor point.*

Proof. Let $I = I_1, I_2, \ldots, I_r = J$ be a connected path from I to J. Since $C(I_1) < T$ and $C(I_r) \geq T$, $\exists s$, $1 \leq s \leq r - 1$ such that $C(I_s) < T$ and $C(I_{s+1}) \geq T$.

There can be more than one such s. We can also have $C(I_{s+1}) < T$ and $C(I_s) \geq T$ but for simplicity we assume that $C(I_s) < T$ and $C(I_{s+1}) \geq T$. The method of this proof holds in both cases.

Suppose $I_s = (a_1, \ldots, a_U)$ and $I_{s+1} = (b_1, \ldots, b_U)$. Since $C(I_s) < C(I_{s+1})$ and I_{s+1} is a neighbor of I_s, there exists an i such that $a_i + 1 = b_i$. Let $P_1 = (a_1, \ldots, a_{i-1}, b_i, a_{i+1} \ldots, a_U)$. If $C(P_1) \geq T$ then P_1 is in A. If not, we construct P_2 which is obtained in same way by replacing one coordinate a_j of P_1 by b_j where $i \neq j$. Notice that P_2 is a neighbor of both I_s and I_{s+1}. If $C(P_2) \geq T$, then $P_2 \in A$ and we are done. If not construct P_3 in same way. Inductively we can complete the proof. □

Theorem 2.6. *The set A of anchor points is connected.*

Proof. For $I = (i_1, \ldots, i_U)$, $J = (j_1, \ldots, j_U)$, define the distance,

$$d(I, J) = \sum_{l=1}^{U} |i_l - j_l|$$

Let $I, J \in A$, $I \neq J$, $d(I, J) = d$. We prove that $\exists I_1 \in A$ such that $d(I_1, J) < d$. Since $I \in A$, we assume without loss of generality that there exists a small neighbor $I_0 = (i_1 - 1, \ldots, i_U)$ with $C(I_0) < T$.

CASE I: $i_1 < j_1$

By Lemma 2.3 $\exists k$ such that $i_k > j_k$. Set $I' = (i_1, \ldots, i_k - 1, \ldots, i_U)$. Clearly $d(I', J) < d$.

CASE Ia: $C(I') \geq T$

Take $I_1 = I'$. Note that $C(i_1 - 1, \ldots, i_k - 1, \ldots i_U) \leq C(I_0) = C((i_1 - 1, \ldots, i_U)) < T$. Thus I_1 is an anchor point with smaller neighbor $(i_1 - 1, \ldots, i_k - 1, \ldots i_U)$ and we are done.

CASE Ib : $C(I') < T$

Consider $I'' = (i_1+1, \ldots, i_k-1, \ldots, i_U)$, then I' is a smaller neighbor of I''. If $C(I'') \geq T$, then $I_1 = I''$ is an anchor point with smaller neighbor I'. If $C(I'') < T$, then $I_1 = (i_1+1, \ldots, i_k, \ldots, i_U) \in A$ and $d(I, J) < d$.

CASE II : $i_1 > j_1$

By Lemma 2.3 $\exists k$ such that $i_k < j_k$. Set $I' = (i_1, \ldots, i_k + 1, \ldots, i_U)$. Clearly $C(I') \geq T$ and $d(I', J) < d$. Let $I'' = (i_1-1, \ldots, i_k+1, \ldots, i_U)$, then $d(I'', J) < d$.

If $C(I'') < T$, then $I_1 = I'$ and $I_1 \in A$. If $C(I'') \geq T$, then $I_1 = I''$ and $I_1 \in A$.

CASE III: $i_1 = j_1$

Find smallest k s.t. $i_k \neq j_k$. We can now use Case I or Case II.

By induction, we can find a connected path from I to J.

□

3 Slice Algorithms

Set

$$V_+ = \{P \in V \mid x_i(P) \leq x_{i+1}(P), i = 1, \ldots, u-1\}$$

where,

$$x_i(P) = i^{th}\text{coordinate of } P.$$

Definition 3.1. Let $< X >$ be an operator that generates all permutations of X.

Example 3.1.

$$< (1,2,3) >= \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$$

$$< \{(1,2), (2,3)\} >= \{(1,2), (2,1), (2,3), (3,2)\}$$

Hence, $< V_+ >= V$.

Definition 3.2. We define a 'bar' map

$$\begin{array}{rccc} \overline{}: & V & \rightarrow & V_+ \\ & P & \rightarrow & \overline{P} \end{array}$$

with $\overline{P}$ is obtained by rearranging coordinates of P in an ascending order.

Lemma 3.1. *For any $P \in V$, the cardinality of the set $\{Q \in V \mid \overline{Q} = \overline{P}\}$ is $\binom{u}{r_1, \ldots, r_s}$ where,*

$$\begin{aligned} r_1 &= \textit{repetition of } x_1(\overline{P}) \\ r_2 &= \textit{repetition of } x_{1+r_1}(\overline{P}) \\ r_3 &= \textit{repetition of } x_{1+r_1+r_2}(\overline{P}) \\ &\ldots \end{aligned}$$

Proof. See second paragraph on page 16 in [4]. □

Example 3.2. Let $V = \{(i_1, i_2, i_3, i_4) \mid 1 \leq i_j \leq 4\}$, and $P = (1, 2, 1, 3)$. Then $\overline{P} = (1, 1, 2, 3)$ and $r_1 = 2, r_2 = 1, r_3 = 1$, and

$$\left|\{Q \in V \mid \overline{Q} = \overline{P}\}\right| = \binom{4}{2, 1, 1} = \frac{4!}{2!1!1!} = 12.$$

Definition 3.3. *Slice anchor points* are elements of $A_+ = A \bigcap V_+$, where A is the set of anchor points.

It is clear that $< A_+ >= A$.

Definition 3.4. Multinomial $\binom{u}{r_1, \ldots, r_s}$ for $Q = (r_1, \ldots, r_s)$ is called the permutation degree (p-degree) of Q.

Lemma 3.2. *Suppose $Q_1, Q_2 \in V_+$ are such that $x_i(Q_1) \leq x_i(Q_2)$, $1 \leq i \leq U$ and $C(Q_1) < T$, $C(Q_2) \geq T$. Then there exists $P \in A_+$, such that $x_i(Q_1) \leq x_i(P) \leq x_i(Q_2)$.*

Proof. We use induction on $m = \sum_{i=1}^{U} [x_i(Q_2) - x_i(Q_1)]$. If $m = 1$, then $Q_2 \in A_+$ and we are done. Suppose j is a maximum index such that $x_j(Q_1) \neq x_j(Q_2)$. Set $Q = (x_1(Q_1), \ldots, x_j(Q_1) + 1, \ldots, x_U(Q_1))$. Now,

$$x_j(Q_1) < x_j(Q_2) \leq x_{j+1}(Q_2) = x_{j+1}(Q_1)$$

$$x_j(Q_1) + 1 \leq x_{j+1}(Q_1)$$

$$x_j(Q) \leq x_{j+1}(Q), \text{ hence, } Q \in V_+$$

If $C(Q) \geq T$, then $Q \in A_+$. If $C(Q) < T$, then since

$$\sum_{i=1}^{u} [x_i(Q_2) - x_i(Q_1)] \leq \sum_{i=1}^{u} [x_i(Q_2) - x_i(Q)] + 1$$

by induction there exists a $P \in A_+$ with required property. □

Lemma 3.3. *Suppose $(a_1, \ldots, a_r, \ldots, a_U) \in A_+$ has the following property:*

$$a_r = max\{x_r(P) \mid x_i(P) = a_i, i = 1, \ldots, r-1, P \in A_+\}$$

Then for any $Q \in V_+$, such that,

$$\begin{aligned} x_i(Q) &= a_i, \quad 1 \leq i < r \quad \textit{and,} \\ x_r(Q) &> a_r, \end{aligned}$$

$C(Q) > T$.

Proof. Suppose $(a_1, \ldots, a_r, \ldots, a_U) \in A_+$ had the property given in the hypothesis and $(b_1, \ldots, b_U) \in V_+$ with $a_i = b_i$ for $1 \leq i < r$ and $b_r > a_r$. We claim that $C(b_1, \ldots, b_U) \geq T$.

We prove this by contradiction. Suppose $C(b_1, \ldots, b_U) < T$. Set $Q = (a_1, \ldots, a_{r-1}, b_r, c_{r+1}, \ldots, c_U)$ with $c_i = \max(a_i, b_i)$, $r+1 \leq i \leq U$. Note that $b_r = \max(a_r, b_r) \leq \max(a_{r+1}, b_{r+1})$ since $a_r \leq a_{r+1}$ and $b_r \leq b_{r+1}$. Thus $Q \in V_+$. Further, $x_i(Q) \geq b_i \; \forall i$ and $C(Q) \geq C(a_1, \ldots, a_U) \geq T$. Hence by Lemma 3.2, there exist $P \in A_+$ with $x_i(Q) \geq x_i(P) \geq b_i$. But for $1 \leq i < r$, $x_i(Q) = b_i = a_i$ and $x_r(Q) = b_r$. Thus we have an anchor point P with $x_i(P) = a_i$, $1 \leq i < r$ and $x_r(P) > a_r$. This is a contradiction. Thus we must have $C(b_1, \ldots, b_r) \geq T$. □

Conversely,

Lemma 3.4. *For any $Q \in \{V_+ \setminus A_+\}$ with $C(Q) > T$, there exists $(a_1, \ldots, a_r, \ldots, a_U) \in A_+$, such that,*

$$\begin{aligned} x_i(Q) &= a_i, \quad 1 \leq i \leq r-1 \text{ and,} \\ x_r(Q) &> a_r. \end{aligned}$$

Proof. Suppose $(b_1, \ldots, b_U) \in \{V_+ \setminus A_+\}$ with $C(b_1, \ldots, b_U) \geq T$. Choose a point $(a_1, \ldots, a_r, \ldots, a_U) \in A_+$ such that r is a maximum index such that $a_i = b_i$, $1 \leq i < r$ and $a_r \neq b_r$.This condition is vacuous if $r = 1$. We claim $a_r < b_r$.

We prove by contradiction. Suppose $a_r > b_r$. Set $Q = (a_1, \ldots, a_{r-1}, b_r, c_{r+1}, \ldots, c_U)$ with $c_i = \min(a_i, b_i)$. It is easy to show that $Q \in V_+$. Since $C(Q) \leq C(a_1, \ldots, a_U)$, for any j,

$$C(x_i(Q), \ldots, x_j(Q) - 1, \ldots, x_U(Q)) \leq C(a_1, \ldots, a_j - 1, \ldots, a_U)$$

If $(a_1, \ldots, a_U) \in A_+$, then there exists j such that $C(a_1, \ldots, a_j - 1, \ldots, a_U) < T$ by definition of anchor point. Hence, $C(x_i(Q), \ldots, x_j(Q) - 1, \ldots, x_U(Q)) < T$. Thus if $C(Q) \geq T$, then Q is an anchor point, which contradicts the minimality of r. Thus $C(Q) < T$. Also $x_i(Q) \leq b_i$. Now we use Lemma 3.2 to find $P \in A_+$, such that $x_i(Q) \leq x_i(P) \leq b_i$. Since $x_i(Q) = b_i = a_i$, $1 \leq i \leq r$ and $x_r(Q) = b_r$, we found $P \in A_+$ such that $x_i(P) = b_i$ for $1 \leq i \leq r$. This violates the maximality of $(a_1, \ldots, a_U)$. Thus we must have $a_r < b_r$. □

Definition 3.5. Let $(a_1, \ldots, a_r, \ldots, a_U) \in A_+$ be an anchor point of Lemma 3.3. Consider,

$$\tau(a_1, \ldots, a_r) = \{P \in V \mid x_i(P) = a_i, 1 \leq i < r, \text{ and } x_r(P) > a_r\}$$

Lemma 3.5. *Suppose $P_1, P_2 \in A_+$ are such that,*

$$x_r(P_1) = \max\{x_r(Q) \mid x_i(Q) = x_i(P_1), 1 \leq i < r, Q \in A_+\}$$

$$x_s(P_2) = \max\{x_s(Q) \mid x_i(Q) = x_i(P_2), 1 \leq i < s, Q \in A_+\}$$

Then $\tau(x_1(P_1), \ldots, x_r(P_1))$ and $\tau(x_1(P_2), \ldots, x_s(P_2))$ are disjoint or identical.

Proof. We will show that if $(c_1, \ldots, c_U)$ is in,

$$\tau(x_1(P_1), \ldots, x_r(P_1)) \bigcap \tau(x_1(P_2), \ldots, x_s(P_2))$$

then $x_i(P_1) = x_i(P_2)$ for $r = s$ and $1 \leq i \leq r$.

We can assume that $r \leq s$. If $r < s$, then we have $x_r(P_1) \geq x_r(P_2)$ since $x_i(P_1) = x_i(P_2) = c_i$ for $1 \leq i < r$. However by Lemma 3.4 we know that $x_r(P_1) < c_r = x_r(P_2)$. This is a contradiction. Hence we must have $r = s$ and $x_r(P_1) = x_r(P_2)$. □

To find suitable $(a_1, \ldots, a_r)$ of lemma 3.5, we make use of lemmas 3.3 and 3.4. We make groups in A_+ of all the anchor points with first $r-1$ coordinates equal to $a_1, \ldots, a_{r-1}$ and choose $(a_1, \ldots, a_r)$ with maximum r^{th} coordinate a_r. Suppose X denotes the set of all such $(a_1, \ldots, a_r)$. Algorithm 1 is design to find X.

Now it is clear from lemma 3.5 that

$$\{Q \in V_+ \mid C(Q) \geq T\} \equiv A_+ \dot{\bigcup_X} \tau(a_1, \ldots, a_r)$$

where the union is disjoint. If we write $< Y >= \bigcup_{P \in Y} < P >$ for any set Y, then this implies that

$$\{Q \in V \mid C(Q) \geq T\} \equiv < A_+ > \dot{\bigcup_X} < \tau(a_1, \ldots, a_r) >$$

Therefore

$$|\{Q \in V \mid C(Q) \geq T\}| = |< A_+ >| + \sum_X |< \tau(a_1, \ldots, a_r) >| .$$

Once $A+$ is found We apply lemma 3.1 to find $| < A_+ > |$. It remains to find the cardinality $| < \tau(a_1, \ldots, a_r) > |$. Before computing that, we define some terms:

Recall that a partition E of a positive integer u is a sequence $s_1, \ldots s_m$ of positive integers such that $\sum_i s_i = u$ [4]. We will denote a partition of u by $E = \alpha_1^{p_1}, \alpha_2^{p_2} \ldots \alpha_r^{p_r}$, if α_i is repeated p_i times, $1 \leq i \leq r$. For example (1,1,1,2), a partition of 5 is denoted as $1^3 2^1$. The set of partitions of u is denote by X_u. In next lemma we assume $\binom{L}{0,m_1,\ldots,m_k} = \binom{L}{m_1,\ldots,m_k,0} = \binom{L}{m_1,\ldots,m_k}$

Lemma 3.6. *Cardinality of* $< \tau(a_1, \ldots, a_r) >$ *is*

$$\sum_{E=\beta_1^{p_1},\ldots\beta_r^{p_r} \in X_u, E_n \leq k} \binom{U}{\alpha_1, \ldots, \alpha_s, \underbrace{\beta_1, \beta_1}_{p_1} \ldots, \underbrace{\beta_t, \beta_t}_{p_t}} \binom{k}{E_n} \binom{E_n}{p_1 \ldots, p_t}$$

where $u = U - r - 1$, $E_n = \sum p_i$ *for any* $E = \alpha_1^{p_1}, \ldots \alpha_r^{p_r}$,

$$\begin{aligned}
\alpha_1 &= \textit{repetition of } a_1 \\
\alpha_2 &= \textit{repetition of } a_{1+\alpha_1} \\
\alpha_3 &= \textit{repetition of } a_{1+\alpha_1+\alpha_2} \\
&\ldots \\
\alpha_s &= \textit{repetition of } a_{r-1}.
\end{aligned}$$

and when $r = 1$ *we assume* $\alpha = 0$

Proof. Let $E = \beta_1^{p_1}, \ldots, \beta_t^{p_t}$ be any partition of u. Then for any choice of E_n values $b_1 < \ldots < b_{E_n}$ from $\{a_r + 1, \ldots, n\}$ let $P = (a_1, \ldots, a_{r-1}, \underbrace{b_1, b_1 \ldots}_{\beta_1} \ldots \underbrace{b_{E_n}, b_{E_n} \ldots}_{\beta_t})$ where each of $b_1, \ldots b_{p_1}$ are repeated β_1 times, each of $b_{p_1+1}, \ldots b_{p_2}$ are repeated β_2 times and so on. By lemma 3.3 and 3.4, $P \in \tau(a_1, \ldots, a_r)$. Now by lemma 3.1

$$| < P > | = \binom{U}{\alpha_1, \ldots, \alpha_s, \underbrace{\beta_1, \beta_1}_{p_1} \ldots \underbrace{\beta_t, \beta_t}_{p_t}}.$$

Since the cardinality of $\{a_r + 1, \ldots, n\}$ is k, there are $\binom{k}{E_n}$ choices for b_i,

If we write partition $E = \beta_1^{p_1}, \ldots, \beta_t^{p_t}$ as $\beta_1^{p_1 - l}, \beta_1^l, \beta_1^{p_2}, \ldots, \beta_t^{p_t}$ and follow the same procedure, we get points of $\tau(a_1, \ldots, a_r)$ Now there are $\binom{E_n}{p_1, \ldots, p_t}$ ways we can rearrange a partition E. Putting all these together we get our formula.

□

Example 3.3. Consider costs $[2, 3, 4, 9, 12]$ with $n = 5, U = 3$ and $T = 10$.

$A_+ = \{(1,1,4), (1,2,4), (1,3,3)(2,2,3)\}$. To construct X we start with $(1,1)$ and record it. Since with 4 is the maximum third coordinate, $(1.1,4) \in X$. This way we get $X = \{(1,1,4), (1,2,4), (1,3,3), (2,2,3), (1,3), (2,2), (2)\}$.

We first calculate $| < \tau(1.1.4) > |$. $U = 3$, $n = 5$ $k = 5 - 4 = 1$, $u = 3 - 3 + 1 = 1$ and $X_1 = \{1^1\}$. Therefore,

$| < \tau(1,1,4) > | = \binom{3}{2\,1}\binom{1}{1}\binom{1}{1} = 3.$

Similar calculations show that

$| < \tau(1,2,4) > | = \binom{3}{1\,1\,1}\binom{1}{1}\binom{1}{1} = 6$

$| < \tau(1,3,3) > | = \binom{3}{1\,1\,1}\binom{2}{1}\binom{1}{1} = 12$

$| < \tau(2,2,3) > | = \binom{3}{2\,1}\binom{2}{1}\binom{1}{1} = 6$

In the following calculations $u = 2$ and $X_2 = \{1^2, 2^1\}$.

$| < \tau(1,3) > | = \binom{3}{1\,1\,1}\binom{2}{2}\binom{2}{2} + \binom{3}{1\,2}\binom{2}{1}\binom{1}{1} = 12$

$| < \tau(2,2) > | = \binom{3}{1\,1\,1}\binom{3}{2}\binom{2}{2} + \binom{3}{1\,2}\binom{3}{1}\binom{1}{1} = 18 + 9 = 27$

In the following calculation $u = 3$ and $X_2 = \{1^3, 1^1\,2^1, 3^1\}$.

$| < \tau(2) > | = \binom{3}{111}\binom{3}{3}\binom{3}{3} + \binom{3}{1\,2}\binom{3}{2}\binom{2}{1\,1} + \binom{3}{3}\binom{3}{1}\binom{1}{1} = 6 + 18 + 3 = 27.$

Further,

$| < A_+ > | = | < (1,1,4) > | + | < (1,2,4) > | + | < (1,3,3) > | + | < (2,2,3) > | = 15$

Therefore $|\{P \in V | C(P) \geq 10\}| = 3+6+12+6+12+27+27+15 = 108$.

The computation of the above example as done by the algorithms is illustrated step-by-step in Section 3.3.1.

3.1 Finding slice anchor points

By Theorem 2.6 we know that the set of anchor points is connected and from Theorem 2.2 we know that they can be found by searching neighbors. Hence, a simple algorithm can be designed to find all anchor points. Our algorithm begins by searching diagonal points (all coordinates are equal) and their neighbors for a "seed" anchor point, once that seed is found we simply start looking at the seed's neighbors for new anchor points. The neighbors of the new anchor points are then searched in turn. This process continues until all anchors are found. While searching for neighbors, if we restrict the search to neighbors with monotonically increasing coordinate values, we are left with A_+, a set of slice anchor points (another set of slice anchor points can be found by restricting the search to neighbors with monotonically decreasing coordinate values). For sake of brevity we omit the details of our algorithm to find slice anchor points

3.2 Counting Points in Slice

Anchors2PointsSlice (Algorithm 2) illustrates the algorithm used to compute the points with cost $\geq T$ in V using only A_+. The counting process begins with the initial call to *Anchors2PointsSlice*(A_+, n, $\begin{bmatrix} 1 & 1 \\ k & U \end{bmatrix}$), where k is the cardinality of A_+. *Anchors2PointsSlice* is a wrapper around the main Algorithm *Anchors2PointsSliceRecur* (Algorithm 3). *Anchors2PointsSliceRecur* uses *MakeGroups* (Algorithm 1). The *Make groups* algorithm generates recursive "groups" within the list of anchor points such that the first column in every generated group is identical. The effect of such grouping is that with each level of recursive grouping we reduce a dimension of our virtual space. This process is illustrated in Figure 2 where groups are generated in the sequence $r_6, r_5, \ldots, r_1$. Each recursive call to *MakeGroups* returns smaller groups. Eventually, when the groups reach a single column, *Anchors2PointsSliceRecur* begins to return and all counting is done when *Anchors2PointsSliceRecur* is unrolling back.

Below are some definitions and formulae used in the algorithms.

Definition 3.6. Suppose $part_U$ is a set of all the partitions of U. For any $E \in part_U$ with $E = s_1^{p_1} s_2^{p_2} \ldots s_r^{p_r}$ we define the following constants:

$$E_n = \sum_{i=1}^{r} p_r \tag{1}$$

$$mult_E = \binom{E_n}{p_1, \ldots, p_r} \tag{2}$$

$$perm_E = \binom{u}{\underbrace{s_1, s_1 \ldots}_{p_1} \cdots \underbrace{s_r, s_r \ldots}_{p_r}} \tag{3}$$

Let,

$$F_U = \sum_{E \in part_U} mult_E \times perm_E \times D_F \qquad D_F = \begin{cases} 1 & E_n = 1, 2 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$R_U = \sum_{E \in part_U} mult_E \times perm_E \times D_R \qquad D_R = \begin{cases} 1 & E_n = 2, 3 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$Q_{j,E_n} = \begin{cases} 0 & E_n < 3 \\ \dfrac{1}{(E_n - 3)!} & E_n = 3 \\ \dfrac{1}{(E_n - 3)!} \displaystyle\prod_{m=0}^{E_n - 4} (j - E_n + 3 + m) & \text{otherwise} \end{cases} \tag{6}$$

Algorithm 1 *MakeGroups*(A_+, r)

1: $\{A_+$ is the $k \times U$ matrix of k lexicographically sorted anchor points.$\}$
2: $\{r$ is a 2×2 matrix where $[rs\ cs; re\ ce]$ define the upper left hand corner (rs, cs) and (re, ce) define the lower right hand corner of the space in A_+ within which groups are to be made$\}$
3: $[rs\ cs; re\ ce] \Leftarrow r$
4: $ri \Leftarrow rs$
5: **while** $ri \leq re$ **do**
6: Record $[ri\ cs + 1]$ as beginning of group
7: $fe \Leftarrow A_+(ri, cs + 1)$
8: **while** $ri \leq re$ **do**
9: **if** $fe = A_+(ri, cs + 1)$ **then**
10: $rf \Leftarrow ri$
11: $ri \Leftarrow r1 + 1$
12: **end if**
13: **end while**
14: Record $[rf\ ce]$ as end of group
15: **end while**
16: return all groups

3.3 Examples

In this section we illustrate the technique of counting points using two examples.

Algorithm 2 *Anchors2PointsSlice*(A_+, n, r)

1: {A_+ is the $k \times U$ matrix of k slice anchor points}
2: {n is the max value of any element in an anchor point}
3: {r is a 2×2 matrix where [rs cs; re ce] define the upper left hand corner (rs, cs) and (re, ce) define the lower right hand corner of the space in A_+ that is currently being processed}
4: $result$ = Anchor2PointsSliceRecur(A_+, n, r)
5: **for all** p, slice anchor point in A_+ **do**
6: $result$ = $result$ + permutation degree of p
7: **end for**
8: return $result$

3.3.1 Example 1

Consider costs [2, 3, 4, 9, 12] with $U = 3$ and $T = 10$. A_+ is obtained using the procedure described in Section 3.1. The process begins with a call to *Anchor2PointsSlice* (A_+, 4, $\begin{bmatrix} 1 & 1 \\ 4 & 3 \end{bmatrix}$), where $rs = 1$, $cs = 1$, $re = 4$, $ce = 3$ and,

$$A_+ = \begin{bmatrix} 1 & 1 & 4 \\ 1 & 2 & 4 \\ 1 & 3 & 3 \\ 2 & 2 & 3 \end{bmatrix}.$$

In this initial call $re = 4$ is the number of slice anchor points and $ce = 3$ is U. Each row of A_+ is the vector for a slice anchor point. Figure 2 shows the counting of points using the slice anchor points in A_+. The *makegroups* algorithm generates recursive groups within the list of slice anchor points such that the first column in every generated group is identical. The effect of such grouping is that with each level of recursive grouping we reduce a dimension of the virtual space.

For each group returned by *MakeGroups*, *Anchors2PointsSliceRecur* computes rp in line 21. We now show the computation of $rp_1, \ldots, rp_7$.

Computing rp_1

Here, $femax = 4, u = 1, k = 0, part_u = \{1^1\}$. Now for each $E \in part_u$ we evaluate $mult_E$(Equation (2)) and $perm_E$ (Equation (3)). For $E = 1^1$, $E_n = 1$ (Equation (1)) and,

$$mult_E = \binom{1}{1}$$

$$perm_E = \binom{1}{1}$$

Algorithm 3 *Anchors2PointsSliceRecur*(A_+, n, r)

1: {A_+ is the $k \times U$ matrix of k slice anchor points}
2: {n is the max value of any element in an anchor point}
3: {r is a 2×2 matrix where $[rs\ cs; re\ ce]$ define the upper left hand corner (rs, cs) and (re, ce) define the lower right hand corner of the space in A_+ that is currently being processed}
4: {If the final value of a summation variable is smaller than its initial value, let that summation be zero.}
5: **if** S is empty **then**
6: return 0
7: **end if**
8: $[rs\ cs; re\ ce] \Leftarrow r$
9: **if** $ce > cs$ **then**
10: $r1 = makegroups(S, r)$
11: **for all** ri such that ri is a 2×2 group specification in $r1$ **do**
12: $result = result + Anchors2PointsSliceRecur(A_+, n, ri)$
13: **end for**
14: **end if**
15: $femax$ =max(cs^{th} column from rows rs to re in A_+)
16: $u = ce - cs + 1$
17: $k = n - femax - 1$
18: {F_u, R_u and Q_{j,E_n} below are from Equation (4), (5) and (6) respectively.}
19:

$$fsum = 1 + kF_u + \frac{(k-1)k}{2} R_u + \sum_{E \in part_U} \left[mult_E \times perm_E \times \sum_{j=2}^{k-1} \left(\frac{(k-j)(k+1-j)}{2} Q_{j,E_n} \right) \right]$$

20: d = denominator of multinomial coefficient of $A_+(re, 1), A_+(re, 2), \ldots, A_+(re, cs-1)$
21: $rp = \frac{U!}{d \times u!} \times fsum$
22: return $result + rp$

Hence using Equation (4),

$$F_1 = (1 \times 1 \times 1) = 1 \tag{7}$$

Similarly using Equation (5),

$$R_1 = (1 \times 1 \times 0) = 0 \tag{8}$$

Hence from *Anchors2PointsSliceRecur* line 19

$$fsum = 1 + 0 + 0 + 0 \tag{9}$$

The last term in Equation (9) is 0 since in that term the final condition of the summation (-1) is less that the initial value (2). In line 20, d is the denominator of the multinomial coefficient of 1,1, which is the denominator of $\binom{2}{2}$. Hence $d = 2!$ From *Anchors2PointsSliceRecur* line 21

$$rp_1 = \frac{3!}{2!1!} fsum = 3$$

Computing rp_2

Similar to rp_1, here $femax = 4, u = 1, k = 0, part_u = \{1^1\}$. However, $d = 1!1!$. Hence,

$$rp_2 = \frac{3!}{1!1!1!} fsum = 6$$

Computing rp_3

Here, $femax = 3, u = 1, k = 1, part_u = \{1^1\}$. $mult_E$ and $perm_E$ identical rp_1.

$$fsum = 1 + 1 + 0 + 0 \tag{10}$$

Hence,

$$rp_3 = \frac{3!}{1!1!1!} fsum = 12$$

Computing rp_4

This computation is similar to rp_3 except that $d = 2!$. Hence,

$$rp_4 = \frac{3!}{2!1!} fsum = 6$$

Computing rp_5

In this case, $femax = 3, u = 2, k = 1, part_u = \{1^2, 2^1\}$ and $d = 1!$. Since there are two paritions of u,

$$F_2 = \left[\binom{2}{2}\binom{2}{1,1}1\right] + \left[\binom{1}{1}\binom{2}{2}1\right] = 3 \tag{11}$$

$$R_2 = \left[\binom{1}{1}\binom{2}{1,1}1\right] + \left[\binom{1}{1}\binom{2}{1}0\right] = 2 \tag{12}$$

Hence,

$$fsum = 1 + 1 \cdot 3 + 0 \cdot 2 + 0 = 4$$

$$rp_5 = \frac{3!}{1!2!} fsum = 12$$

Computing rp_6

Here, $femax = 2, u = 2, k = 2, part_u = \{1^2, 2^1\}$ and $d = 1!$. Using Equations (11) and (12),

$$fsum = 1 + 2 \cdot 3 + 1 \cdot 2 + 0 = 9 \tag{13}$$

Hence,

$$rp_6 = \frac{3!}{1!2!} fsum = 27$$

Computing rp_7

Finally, here $femax = 2, u = 3, k = 2, part_u = 1^3, 1^1 2^1, 3^1$ and $d = 1!$. So,

$$F_3 = \left[\binom{3}{3}\binom{3}{1,1,1}0\right] + \left[\binom{2}{1,1}\binom{3}{1,1}1\right] + \left[\binom{1}{1}\binom{3}{3}1\right] = 7 \tag{14}$$

$$R_3 = \left[\binom{3}{3}\binom{3}{1,1,1}1\right] + \left[\binom{2}{1,1}\binom{3}{1,1}1\right] + \left[\binom{1}{1}\binom{3}{3}0\right] = 12 \tag{15}$$

$$fsum = 1 + 2 \cdot 7 + 1 \cdot 12 + 0 = 27$$

Hence,

$$rp_7 = \frac{3!}{1!3!} fsum = 27$$

Line 4 in Algorithm 2 returns the sum of all rp_i, $1 \leq i \leq 7$ and in line 6, the p-degree of each slice anchor point is added. The overall number of points $\geq T$, 108, is returned in line 8. The p-degree of each point in A_+ is shown in the rightmost column in Figure 2.

We can see that Equations (4) and (5) are independent of k and can be reused in computing rp of a column from Figure 2

3.3.2 Example 2

Now, we present another example similar to that in Section 3.3.1, but with $T = 8$. This example has fewer slice anchor points and exercises computation of Q_{j,E_n} (Equation (6)). Figure 3 illustrates this computation.

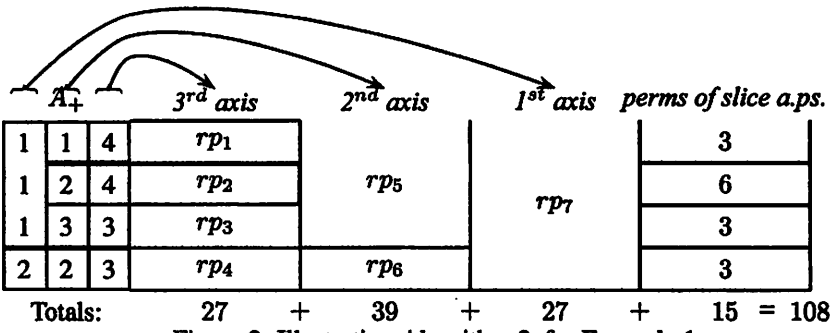

Figure 2: Illustrating Algorithm 2, for Example 1

Consider costs [2, 3, 4, 9, 12] with $U = 3$ and $T = 8$. The process begins with a call to *Anchor2PointsSlice*(A_+, 2, $\begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix}$),where

$A_+ = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 2 & 2 \end{bmatrix}$.

Computing rp_1

Here, $femax = 3$, $u = 1$, $k = 1$, $part_u = \{1^1\}$. $fsum$ in this case is identical to Equation (9). Hence,

$$rp_1 = \frac{3!}{2!1!} fsum = 6$$

Computing rp_2

Here, $femax = 2$, $u = 1$, $k = 2$, $part_u = \{1^1\}$ and $fsum$ is again identical to Equation (9). Hence,

$$rp_2 = \frac{3}{1!1!1!} fsum = 18$$

Computing rp_3

Here, $femax = 2$, $u = 2$, $k = 2$, $part_u = \{1^2, 2^1\}$ and $fsum$ is identical to Equation (13). Hence,

$$rp_3 = \frac{3!}{1!2!} fsum = 27$$

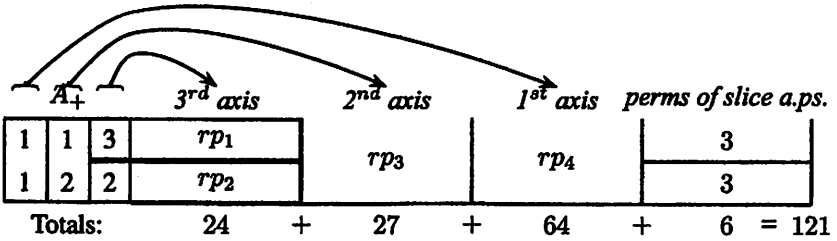

Figure 3: Illustrating Algorithm 2, for Example 2

Computing rp_4

Here, $femax = 1$, $u = 3$, $k = 3$, $part_u = \{1^3, 1^1 2^1, 3^1\}$. Using F_3 and R_3 from Equations (14) and (15) respectively,

$$\begin{aligned} fsum &= 1 + 3 \cdot 7 + 3 \cdot 12 + \left\{ \binom{3}{3} \binom{3}{1,1,1} \left[\frac{(3-2)(3+1-2)}{2} \frac{1}{(3-3)!} \right] \right\} \\ &\quad + \left\{ \binom{2}{1,1} \binom{3}{1,2} \left[\frac{(3-2)(3+1-2)}{2} \cdot 0 \right] \right\} \\ &\quad + \left\{ \binom{1}{1} \binom{3}{3} \left[\frac{(3-2)(3+1-2)}{2} \cdot 0 \right] \right\} \\ &= 1 + 21 + 36 + 6 + 0 + 0 \\ &= 64 \end{aligned}$$

Hence,

$$rp_4 = \frac{3!}{3!} fsum = 64$$

The rightmost column of Figure 3 shows the p-degree of each point in A_+. Adding up all rp_i, $1 \leq i \leq 4$ and the p-degrees, give us 121, the total number of points with cost ≥ 8.

4 Complexity

The computational complexity of *Anchors2PointsSliceRecur* is dominated by line 19. Both F_U and R_U requires the computation of all unrestricted paritions of U. The number of unrestricted paritions of U, $P(U)$ is given by [2] as

$$P(U) \approx \frac{e^{2\pi\sqrt{2U/3}}}{4U\sqrt{3}}$$

Assuming that each paritions can be generated in constant time, since $U! = O(U^U)$ the complexity of line 19 in *Anchors2PointsSliceRecur* is

$$\begin{aligned} &= O\left(U\frac{e^{2\pi\sqrt{2U/3}}}{4U\sqrt{3}}\right) + \left[O\left(\frac{e^{2\pi\sqrt{2U/3}}}{4U\sqrt{3}}\right)(UnO(U^U))\right] \\ &= O(nU^U e^{2\pi\sqrt{2U/3}}) \end{aligned} \tag{16}$$

A_+ has fewer points than A by a factor of $U!$ hence in the worst case the number of anchor points is $a = \frac{U(n-1)^{(U-1)}}{O(U^U)}$.

The overall complexity with constant U and worst case a,

$$\begin{aligned} &= O\left(n\log n + \frac{U(n-1)^{2(U-1)}}{U^{2U}} + n\right) \\ &= O(n^{2U}) \end{aligned} \tag{17}$$

Hence the *asymptotic* complexity does not change. This is to be expected since the reduction is based on U and we let U be the constant.

However, with constant n, the overall worst case complexity is,

$$\begin{aligned} &= O\left(\begin{array}{l} U^3 + \frac{U(n-1)^{(U-1)}}{U^U}3^U U^2 + \frac{U^2(n-1)^{2(U-1)}}{U^2 U} + \\ \frac{U(n-1)^{(U-1)}}{U^U}U\log\left(\frac{U(n-1)^{(U-1)}U}{U^U}\right) + nU^U e^{2\pi\sqrt{2U/3}} \end{array}\right) \\ &= O\left(\frac{n^{2U}}{U^{2U}}\right) \end{aligned} \tag{18}$$

4.1 Comparison with Brute Force Method

The brute force method is one in which we test each point in V for $\geq T$. The complexity of this approach is always $O(Un^U)$. We can conclude that while in the worst case (Equations (18) and (17)) our algorithms exhibit worse scalability than brute force whereas in the best case we do significantly better. Our future work will include an average case complexity analysis in which we hope to show that our approach works much better than brute force in the average case. This is primarily because the worst case, as outlined above, occurs only for a very narrow range of T.

4.2 Comparison with Generating Functions

A classic method of counting points in the U-dimensional space is using generating functions. Though our problem requires counting points $\geq T$, when using generating functions it is easier to calculate points $\leq T$, so that is what we will do in this section. We begin with explaining this method and then compare it with our algorithms.

Let the costs be $[1, 2, 5]$, $U = 2$ and $T = 3$. This first step is to create a generating function *for each axis* from the costs. Such a generating function is,

$$g(x) = x + x^2 + x^5 \tag{19}$$

Since there are two such axes we multiply the generating function twice and divide the result by $(1 - x)$ [1] in order to accumulate coefficients. We then differentiate the result T times, evaluate at $x = 0$ and divide by $T!$ to get the number we want. Multiplying generating functions had the effect of creating a multinomial expansion of the generating function. Such an expansion has the property that for each term in the expansion, the coefficient of a given term in x happens to be the number of ways to get the exponent of that x. Multiplying the generating function by $\frac{1}{1-x}$ has the effect of accumulating coefficients [1]. Now, in order to get the coefficient of the x term with an exponent of T we differentiate T times. We then evaluate the result at $x = 0$ to cancel all terms with exponents greater than T. Finally to negate the effect of repeated derivative on the coefficient of x we divide the end result by $T!$. Hence we evaluate,

$$\begin{aligned} N &= \frac{d^T}{dx^T} \frac{(x + x^2 + x^5)^2}{1 - x} \Big]_{x=0} \frac{1}{T!} \\ &= \frac{d^3}{dx^3} \frac{(x + x^2 + x^5)^2}{1 - x} \Big]_{x=0} \frac{1}{3!} \\ &= 3 \end{aligned} \tag{20}$$

We can now say that there are 3 ways that $T \leq 3$, they are $(1, 1)$, $(1, 2)$, $(2, 1)$. Equivalently there are $6 = 3^2 - 3$ ways such that the sum is ≥ 4. So, the required probability of $T \geq 4$ is $\frac{6}{9} = 0.66$. There are two computationally expensive steps in the evaluation of Equation (20): Evaluating the T^{th} derivative and computing $T!$. We will now look at how we can eliminate or reduce one or both of these steps.

If we try to prevent explicit computation of $T!$ we must compute $\frac{d^T}{dx^T}$ in T steps, $\frac{d}{dx}, \frac{d^2}{dx^2}, \ldots, \frac{d^T}{dx^T}$ and at each step divide the result by $n, n-1, n-2, \ldots, 1$. Division of $\frac{d^T}{dx^T}$ by $T!$ or division of each successive derivative by $n, n-1, n-2, \ldots$ is required to cancel the effect of multiplication of the coefficient of x by it's exponent in each successive derivative.

If we try to prevent successive computation of derivatives then we must compute $T!$. While the gamma function $\Gamma(T) = \int_0^\infty x^T e^{-x} dx$ can be used to exactly calculate $T!$, $T! = \Gamma(T + 1)$, it involves at least T integration steps. An approximation to $T!$ can be obtained using T or fewer steps by using Sterling's approximation as $T! \approx \sqrt{2\pi T} \left(\frac{T}{e}\right)^T$. In either case, the factorial value is large for even modest values of T, for example, 100! contains 158 digits. Given that T represents the sum of costs, finding the factorial is constrained to costs that can be handled by the precision of the system on which the computation is being carried out. It is unlikely that most commonly used general-purpose systems will be able to compute the factorial of large cost values (in the order to several hundreds or thousands) without overflow.

Hence we cannot eliminate both these computationally intensive steps simultaneously. So the choice is between finding the T^{th} derivative, one at a time or calculating $T!$. Both of these are choices severely constrain using this generating functions technique for large values of T (several hundreds or thousands).

Though the computational requirement of this method is sensitive to T, it is fairly immune to U, the number of users. This is the case since in Equation (20), U is the exponent of the generating function and does not affect the computational effort to find a derivative. Here we can see a substantial reduction in complexity due to the slice algorithms. Shen and Marston [3] consider a restricted case of our problem with usual die consisting of sequentially numbered face values (costs). Their techniques are based multinomial expansions and they claim a running time of $O(U^2, n^2)$. The paper does not give any details of the dynamic programming used.

All algorithms described in our paper are available as MATLAB™code.

References

[1] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison Wesley, Reading, Massachusetts, 1988.

[2] G. H. Hardy and S. Ramanujan. Asymptotic Formulae in Combinatory Analysis. *Proc. London Math. Soc.*, 17:75–115, 1918.

[3] Zhizhang Shen and Christian M. Marston. A Study of a Dice Problem. *Appl. Math. Comput.*, 73(2-3):231–247, 1995.

[4] Richard P. Stanley. *Enumerative Combinatorics*, volume 1 of *Cambridge Studies in Advanced Mathematics 49*. Cambridge University Press, 1997.