Napkin Analysis of Paths by K-Integer Nodes is a feasible exact superclass of Mann Whitney U handling ties

July 18, 2024

Dr Keith S. Reid[1] [2]
keithreid@nhs.net
1. Department of Health and Life Sciences, University of Northumbria at Newcastle
2. Positive and Safe Care team, Cumbria Northumberland Tyne and Wear NHS Foundation Trust, Newcastle

The author gratefully acknowledges *Aji Lewis MBE, Peter McMeekin and Mick Hill.*

Abstract

Mann-Whitney U test, a canonical test of the order of two groups, does not handle ties. Typical implementations use approximations fragile to small samples. These problems are troublesome in certain clinical situations where a small range of possible values necessarily forces either small samples or ties. This paper presents Napkin, an exact computationally feasible superset of Mann-Whitney U which handles ties. Call sets S and T. Let q be length in steps of a path across an adapted Young diagram where sloped steps are ties. Generate all pairs of binary numbers of variable length $q \in [1 : n = s + t]$ which write, with 1, the places where elements are present. Binary pairs must meet bitwise OR and binary representations of untied pairs also meet XOR. Partitioning elements over the 1s generates all paths prior to conflating isomorphic paths. Tied elements follow discrete distributions in similar data and deterministically shorten paths. Thus the distribution of tied elements informs likelihood of path lengths. Paths derived from binary pairs of each length have deterministically exact relative likelihoods. In a clinical application the exact test is more sensitive than the approximation. Benefits include exactness, visual beauty and shorter experiments.

Keywords: Mann-Whitney Wilcoxon Young Ties Napkin

1 Introduction

1.1 Aim

This manuscript aims to solve difficulties of the Mann-Whitney U test ("MWU") (Mann and Whitney, 1947). MWU is in common use, a recent literature search in 2024 for "Mann Whitney U" returned over 100,000 results (NLM, 2000). The difficulties are expense, error and ties. An exact computationally feasible superset of MWU is presented. The new proposed test is called Napkin Analysis of Paths by K-Integer Nodes, recursively abbreviated to "Napkin". Napkin is a superset because Napkin handles all possible occurrences of ties, including no ties. MWU remains appropriate when there is an expectation of no ties in the data. Conversely MWU as a subset of Napkin can feasibly be exactly computed.

1.2 Context

The immediate authorial context of Napkin was Shannon-informational analysis. The present author's clinical vocation is using quantitative methods to reduce restraint in psychiatric hospitals. Specifically, a test of questionable information in clinical data sets, the L-test, relied on the MWU, thus inheriting any issues (Reid, 2022; Reid and Price, 2022, 2024). Hence a real clinical data set with ties is used as the demonstrative application of the methods.

1.3 Content

This manuscript proceeds by proof to develop a feasibly computable algorithm called "Napkin". Tables of values are shared, echoing similar tables in the original MWU paper such as Table 1. Computational efficiencies are discussed. Code is presented, in-line, where relevant, and is also in a supplement. A brief discussion considers some implications and utility.

1.4 Terms

1.4.1 Apology

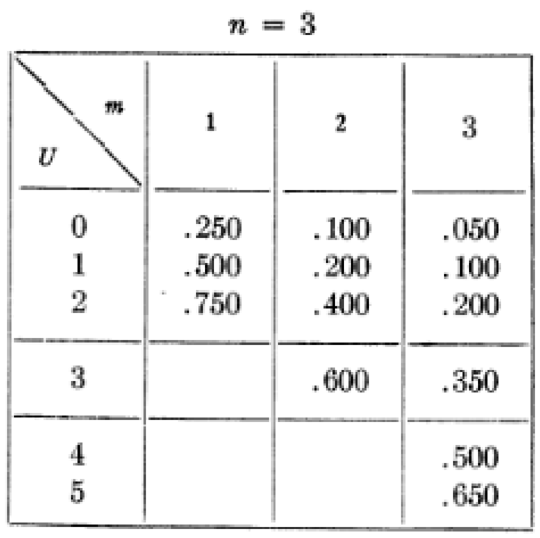
Relevant foundational terms are now defined to aid adoption among clinicians. Similarly, innovations are supported by visual proofs. The novel ideas are in fact only one step on from entry-level definitions. Clear statement of the basic priors seems apt, then, as the only possible preface. There is also a need to define terms such that they align with combinatorial conventions.

1.4.2 Outline of MWU in present combinatorial notation

MWU assesses the probability of a given order of two numeric sets (Mann and Whitney, 1947). The order score is denoted U. Calling sets S and T, any s-element, beating any t-element, adds one U-point. Conventionally S is the smaller set. The count or cardinal number of S is denoted s and the cardinal number of T is denoted t. The combined overall length of the order is $n = s + t$. For consistency call the overall order N.

The ordered U-scores of all orders of two sets of five may be denoted $\mathbb{U}_{s5,t5}$ or more briefly $\mathbb{U}_{5,5}$. Writing pipes around an expression to mean its count or cardinal number, then $|\mathbb{U}_{3,3}|$ is the number of all distinct orders given $s = 3$ and $t = 3$. The actual numeric value of a count itself may be denoted with a prefixed pound or hash #. See Figure 1 and Figure 2 for visual proofs of Equation 1 and Equation 2.

$$|\mathbb{U}_{3,3}| = \#20 \tag{1}$$



$n = 3$

U \ m	1	2	3
0	.250	.100	.050
1	.500	.200	.100
2	.750	.400	.200
3		.600	.350
4			.500
5			.650

Table 1: Reproduced from the original MWU paper. Given $u = 3$; $p = 0.35$ as implied by Equation 1 and Equation 2 and Figure 1 and Figure 2.

1.5 Young Diagrams

Any order N of two sets S, T can be drawn as a Young diagram. In such representations the path of a notional "ant" or automaton crosses a lattice. The lattice has dimensions $s \times t$. The path encodes the order of N by going right for elements in S, and up for those in T (Stanley, 2011).

In accordance with the definition of area, the commutative and distributive laws of addition and multiplication act on the segments of the area, as it is formed by the path. This area subtended in the top left corner is U, which is supported by visual proof and supports useful elaborations and efficiencies (Bucchianico, 1999). This echoes a means of representing binary numbers which goes right for 1 and up for 0 (Knuth, 2005, Ch.7).

1.5.1 Specific paths

The appearance of the squares on a field resembles the titular Napkins on a table and connotes the English idiom "back of a Napkin" meaning elementary and expeditious calculations. The most "S-dominant" distinct path in Figure 1 the singleton path U_9 goes right, only, a total distance of s; then up only a total distance of t. It covers all squares having $U = s \times t$ so $|U_9| = \#1$. If the lattice were a table with napkins, the table is covered.

Conversely the most "T-dominant" distinct path goes up then right and covers no squares such that $|U_0| = \#1$. There are no napkins on the table. Integers may only have integer products so the steps are in whole numbers and only whole squares can be covered, there are no partial napkins. Hence the range of $\mathbb{U}_{s,t}$ must be $[0, 1, ...s \times t]$.

The constraints on the range of Young diagram areas, and the decidability of all possible Young diagrams, when considered as options for each possible area, mean that all paths can be calculated. Table 1 is taken from the founding MWU paper and is reproduced with a minor correction[1]. Note that in the MWU paper the terms m and n are used instead of s and t in the exposition above. The current paper differs from that original usage in order to preserve n as the total number of elements and agree with combinatorial terms using n.

1.6 The pre-existing test statistic in the present notation

Any U-score is more or less extreme compared to all order U scores $[0...s \times t]$ all distinct paths in $\mathbb{U}$. Assuming a one-tailed distribution for demonstrative simplicity, the test probability is the fraction of $|\mathbb{U}_{s,t}|$ equally extreme or less extreme than U. For example in accordance with Figure 1 and Table 1, $\mathbb{U}_{3,3}$ has seven of twenty paths with $U \leq 3$, which gives $p_{MWU} = 0.35$ as stated in Table 1.

[1] $1, 2, 2$ to $1, 2, \mathbf{3}$ in the top row representing MWU's m hence the larger typesetting of 3 here.

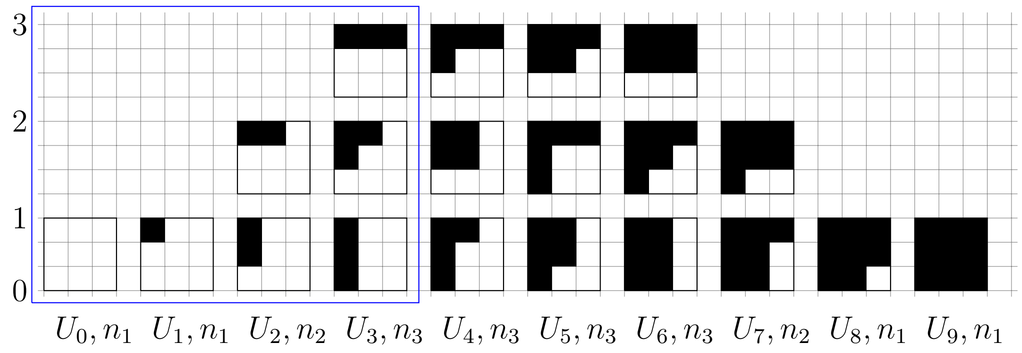


Figure 1: $\mathbb{U}_{3,3}$ has #7 $U \leq 3$ paths out of #20 which gives $p = 0.35$, concording with Table 1 and Figure 2, and Equation 1 and Equation 2

If $U = 3$ and the lattice is of dimensions $|\mathbb{U}_{3,3}|$ then:

$$p = \frac{|\mathbb{U}_{3,3} \leq 3|}{|\mathbb{U}_{3,3}|} = \frac{\#7}{\#20} = 0.35 \tag{2}$$

2 Motivating difficulties

This manuscript duly started by acknowledging, the utility and attestation of MWU. Its usefulness and importance are part of the motivation. In order to proceed it is necessary to clarify the three difficulties raised above, of expense, error and ties.

2.0.1 Expense

MWU is expensive because building all orders is complex. The number of elements in $\mathbb{U}_{s5,t5}$ is $n = s + t = 10$. One "worthlessly" expensive approach proceeds by placing each of the ten "first", tries the remaining nine, etc,. needing over 3.6 million computations (Stanley, 2011, 1.1.4). Thus the first MWU difficulty is combinatorial expense.

Such complexity can be avoided. Per Figure 2, Pascal's triangle, all unique paths across such a lattice can be calculated by simple repeated addition of whole numbers which is computationally cheap. Each cell contains the count of its paths from self to top. This is the same as the number of unique paths, which is the same as the number of standard Young diagrams. That provides the denominator of the test statistic in a computationally cheap way.

1

1 **1**

1 **2** **1**

1 **3** **3** **1**

1 **4** **6** **4** **1**

1 **5** **10** **10** **5** **1**

1 **6** **15** **20** **15** **6** 1

7 **21** **35** **35** **21** 7

... 28 **56** **70** **56** 28 ...

... 84 **126** **126** 84 ...

... ... 210 **252** 210 ... ...

... ... 462 462 ... ...

... ...

Figure 2: Pascal's Triangle: numbers sum their "parents" diagonally above

Discerning that $|\mathbb{U}_{3,3}| = \#20$ requires 9 additions. Discerning that $|\mathbb{U}_{5,5}| = \#252$ requires 25 additions, and generally it is $s \times t$ such additions that are required, mitigating combinatorial explosion for the value in Equation 1 which is also the denominator of Equation 2. Calculating only the U scores at the extreme end (lower end in Figure 1) and summing their cardinal number, as in Figure 1, simplifies counting the numerator. Young diagrams can be drawn "to order" in this way as shown in sections ss.2.1.4-2.1.8, which sets the scene for consideration of error in s2.1.9.

2.1 Isomorphism

Before drawing all Young diagrams to order it is convenient to note a feature of sets of Young diagrams implied in the founding MWU approach and in Pascal's triangle. It is also intuitively true and implied by the use of Pascal's triangle. This manuscript calls the property isomorphism. Isomorphic paths are not distinct, in the sense that distinct was used above in discussing singleton paths U_0 and $U_{s\times t=9}$. This implies a property of sets of Young diagrams, paths, and orders making up $\mathbb{U}_{st}$ which Napkin calls isomorphism. Working through the MWU paper, this appeared to be the approach taken, seemingly necessary to achieve the values in the tables. Orders which are represented by isomorphic paths, i.e. which describe the same outline of one Young diagram, were treated as the same in that founding MWU paper and only counted once. This is supported by deduction, example and with reference to Pascal's triangle.

2.1.1 Isomorphism supported by deduction

Elements in S and T and thus the internal order of N are indistinguishable except by set membership and order. They are furthermore only only distinguishable inasmuch as order affects U of N.

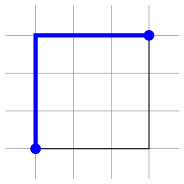


Figure 3: $[\uparrow, \uparrow, \uparrow] > [\rightarrow, \rightarrow, \rightarrow]$

2.1.2 Isomorphism supported by example and contradiction

Figure 3 and Figure 4 are isomorphic. Given the 4 different ways to achieve the vertical and horizontal limbs of the path, i.e. $[3], [2,1], [1,2], [1,1,1]$, there would be 16 ways to achieve each singleton if isomorphism were not applied. They all isomorphically collapsed into a single path, Young diagram and distinct order. Isomorphic paths Figure 3 and Figure 4 together are counted only *once* in twenty in $\mathbb{U}_{s3,t3}$ when operators do not consider any ties. Hence they are counted just once in MWU. Again this concords with the Table 1 statistic that $p = \frac{\#1}{\#20} = .050$ of $\mathbb{U}_{s3,t3}$ have $U = 0$, and the other MWU probabilities in tables.

2.1.3 Isomorphism supported by reference to Pascal's triangle

Treating isomorphic paths as one countable entity is also implied by Pascal's triangle. The cardinal number of paths from each node in Pascal's triangle is the same whether one pauses at each number or jumps each straight line in one leap. In that sense festinating and leaping are isomorphic styles of travel if they travel over the same path.

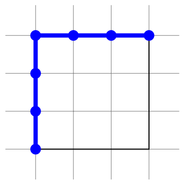


Figure 4: [↑>↑>↑>→>→>→]

2.1.4 Build isomorphic U_0

Thus, Figure 3 and 4 represent isomorphic paths mapping onto the only Young diagram that has no area in $\mathbb{U}_{s3,t3}$. It represents T entirely beating S, such that S is 4th, 5th, 6th, and T is 1st, 2nd, 3rd. The path across the grid goes $\uparrow, \uparrow, \uparrow, \rightarrow, \rightarrow, \rightarrow$ and U is $U = 0$. We label elements in S as $s_\alpha, s_\beta, s_\gamma$. Likewise T elements as $t_\alpha, t_\beta, t_\gamma$. This path is $s_\alpha > s_\beta > s_\gamma > t_\alpha > t_\beta > t_\gamma$. In fact there is always one way to form a Young diagram with $U = 0$ means that the value of $\mathbb{U}$ at $U = 0$ is always $n = 1$, as noted by Bucchianico (1999). Normal and binomial are very small at that value, as shown in the comparison at Figure 10.

2.1.5 Build isomorphic U_1

Figure 5 shows the area cut off by the path of the only distinct path that has area of $U = 1$ without ties. It represents one S element beating one T such that S is 3rd, 5th, 6th, and T is 1st, 2nd, 4th. The path across the grid goes $\uparrow, \uparrow, \rightarrow, \uparrow, \rightarrow, \rightarrow$. U is 1. The path is $s_\alpha > s_\beta > \quad t_\alpha > s_\gamma > \quad t_\beta > t_\gamma$. Again this forces a mismatch with the approximations in Figure 10, as discussed in more detail at s2.1.9.

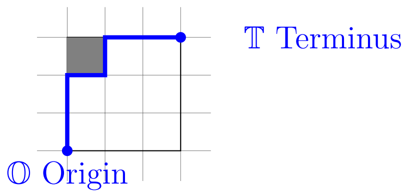


Figure 5: Only Young diagram of $U = 1$ in $\mathbb{U}_{s3,t3}$ in grey subtended by a path

2.1.6 Build isomorphic U_2

Figure 6 shows the two paths that have area of $U = 2$. The algorithm for iterating through all such paths is that one starts with the left-most arrangement of the tiles or “napkins”. We look for “donor” napkins that can be moved right and “receptors” that can accept them. Donors are always the bottom of a column and rightmost in a row. Receptors are any cell which won’t stick out below its leftmost neighbour or the bottom of the “table”. Young diagrams keep all arrangements in the corner. For a given U this generates all arrangements at least once. For U_0 and U_1 it generates only one diagram because there is no receptor or donor and the end state is the beginning state.

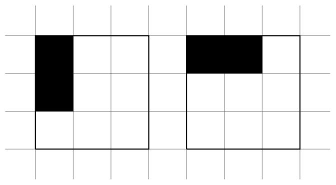


Figure 6: Two Young diagrams of $U = 2$ in $\mathbb{U}_{3,3}$ in black

2.1.7 Build isomorphic U_3

Figure 7 and Figure 8 show the three paths that have area of $U = 3$ in $\mathbb{U}_{s3,t3}$. They are formed, beginning with the first upright bar, by moving some squares from “donor” squares to “receptor” squares. Figure 7 illustrated how the middle path was built from the left hand path. The grey cell moves in a decidable way from donor to receptor.

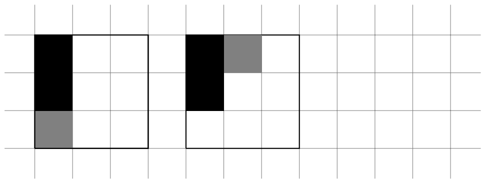


Figure 7: Progression from completely vertical leftmost $U = 3$ in $\mathbb{U}_{s3,t3}$ to the “more horizontal” middle shape

Figure 8 illustrates how the right path was built from the middle path, again in a decidable way. The implementation is simple and code has been retained available on request. It amounts to some condition statements about a vector of integers with length s and limits $0, t$; and the rules about columns not sticking out below leftmost neighbours.

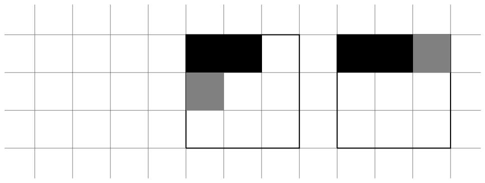


Figure 8: Progression from intermediate middle shape to last shape with no donors

2.1.8 Build isomorphic $s5t8u8$

For a more interesting and asymmetrical example, see Figure 9 all paths generated by Napkin with $U = 8$ assuming no ties in $s5t8$. In this case the Young Diagrams are “upside down”. The present author understands this to be known as the French style of Young diagram. It is employed here because it affords the benefit of aiding an intuitive sense that the stack cannot stick out over the base. Another intuitive benefit is that transformations are only allowed to happen by "pushing" a napkin right. Not all lines between diagrams are included, some would have crossed over clumsily.

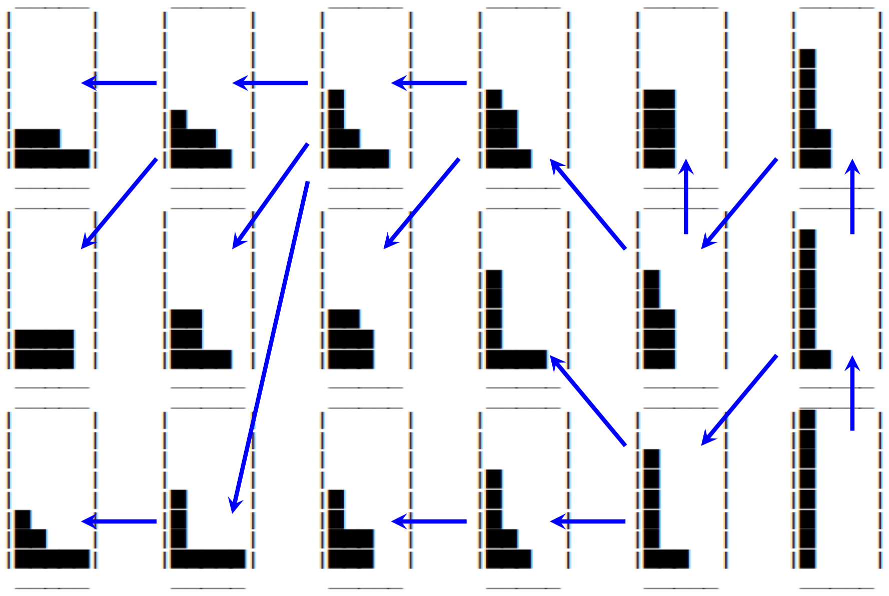


Figure 9: Generating all possible Young diagrams with $s5t8U8$ if ties never occur, progression is by starting with the leftmost stack possible and knocking off top Napkins rightward

2.1.9 Error

The normal distribution denoted here Φ and the binomial distribution denoted $\mathbb{B}$ are parametric, see Figure 10 for an attempt to show them at scale with $\mathbb{U}_{5,5}$. $\mathbb{U}$ the distribution of orders is not parametric. It is "stepped" see Figure 1 where $\mathbb{U}_{3,3}$ is so stepped as to have a ledge of width four at the apex. The result of the De Moivre-Laplace Theorem ("DMT") however is that Φ and $\mathbb{B}$ become more similar as n grows (Ross, 2018, p.219). The parametric nature of Φ makes calculation simple.

Thus, intending to mitigate expense by using approximation, MWU's founding paper proved $\mathbb{U}$ approaches $\mathbb{B}$ which in turn approaches Φ due to DMT. The fit of $\mathbb{U}_{s,t}$ to $\mathbb{B}$,

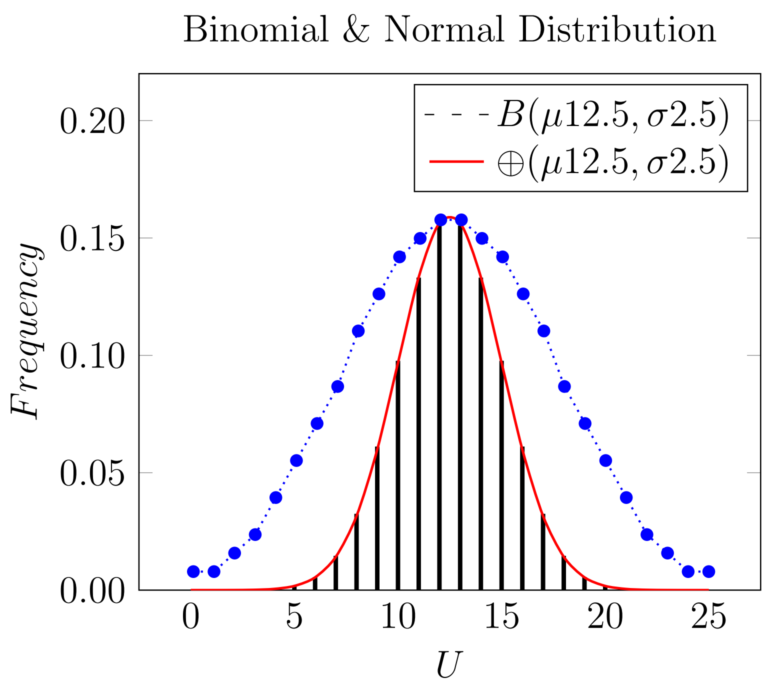


Figure 10: $\mathbb{U}_{s5,t5}$ is exact in blue; neither binomial black bars nor normal red line match blue

and the goodness of these approximations improves as set size grows. This means poorer approximation for small samples like $\mathbb{U}_{s5,t5}$. Figure 10 shows that for $\mathbb{U}_{5,5}$, $\mathbb{U}$ is quite different to the binomial and normal. Binomial $\mathbb{B}$ is the black bars, only existing at whole numbers. Normal Φ is the red line. They both noticeably mismatch the blue line. Online MWU calculators, including the current author's calculator of choice, ban smaller groups than $\mathbb{U}_{s5,t5}$ to prevent unacceptable approximation (Lowry, 2022).

Thus the second issue is error, which arises in an attempt to reduce expense by approximating. The Napkin method for calculating Young diagrams with out ties is practicable on domestic hardware obviating the need for approximation.

2.2 Ubiquity of exceptions in clinical work

Ties are not dealt with by MWU's founding paper. Some analysts approximate ties' effect (Marx et al., 2016). Some ignore them (McGee, 2018). Put another way, MWU as currently implemented proceeds as if expecting with certainty there are not ties. This implies zero expectation of $n-1$ tied elements, $n-2$ elements, down to 1 tied elements all having zero frequency. The commensurate discrete distribution is charted at Figure 11.

This manuscript described above how small samples break approximation's limits. Sadly, either ties or small samples can lead to MWU being imperfect. Now it may be proven proven that for certain clinical rating scales, exceptions are provably ubiquitous.

Some rating scales in clinical work have ranges of possible scores in the region of ten (Busner and Targum, 2007) (Dickens et al., 2020) (Dack et al., 2012). By the pigeonhole principle a ten-point rating scale measuring eleven persons must have ties (Borowski and Borwein, 2002, p.428). Alternatively at $n \leq 10$ samples suffer approximation error. Thus ties *or* small samples *must* occur given such rating scales. There appears to be a difficult situation where given certain scales all circumstances are exceptions to the safe use of MWU.

3 Methodology

Napkin Analysis of Paths using K-Integer Nodes, ("Napkin"), proceeds with adapted Young diagrams as follows.

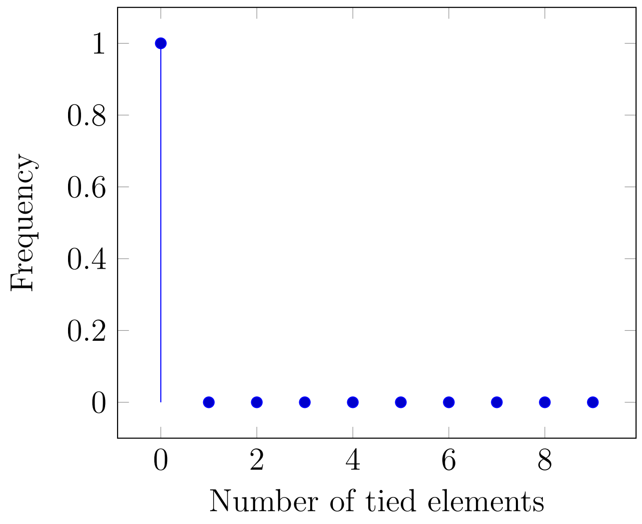


Figure 11: Implied expectatation distribution of ties in MWU

3.1 Tied elements as sloped Young steps

The first is to tackle ties by considering tied elements. The second is to draw steps in the path featuring ties as diagonal steps. See Figure 12 the actual path of some human rights related data. See Figure 14 all possible paths of length three steps, which is soon denoted $q3$ in $U_{5,5}$; or may be denoted $s5t5q3$, etc.. See Figure 18 where after some development it seems that there are three distinct paths with $U = 1$ if tied elements can exist.

3.2 Tied elements as shortening the isomorphic path

The third is to recognise that tied elements shorten the number of steps in a path in a way which is decided by the number of tied elements. The fourth is to suppose that tied elements follow a distribution. The fifth is to use that distribution, and isomorphism within set of paths generated with a certain number of tied elements, to model the likelihoods of path lengths. It is important to recognise that, just as isomorphism exists in the MWU distribution expecting no tied elements, yet separate isomorphisms pertain within each group of paths which have path distance q as defined later. That model is used to weight the contributions of different path lengths to an overall distribution which accounts for ties.

3.3 Which distribution?

A way forward is offered by modelling tied elements rather than ties. If the attack defines each tied element as each tied to one and only one element, then the occurrence of tied elements has useful constraints.

3.3.1 Tied elements τ where $0 \leq \tau < n$

In this scheme a singleton element in an order of $n = 1$ is never tied, neither to itself nor to null. The fastest possible rate of increase from that base case is one per increase in length n. Call the total number of tied elements, over N, from both sets, τ. By induction τ is then a whole number from 0 to $n - 1$. This is consistent with Figure 10; note the graph there does not allow for $\tau = 10$.

3.3.2 Deducing toward Poisson

Elements are either tied or not so a plausible discreet distribution should be used. When building distributions to assess extremity of order, one cannot make assumptions about order. Occurrence of tied elements is an aspect of order. So tied elements cannot be assumed to interact in any way. Therefore tied elements are assumed to occur independently. Such necessary assumption of independent occurrence with a space meets Poisson distribution assumptions, which the attack now takes up as a weak assumption.

3.3.3 A soft pragmatic approach

The current author takes a "weak" assumptive position on using Poisson, though four reasons are given at s3.3.2 above and ss3.3.4-3.3.6. A strong position is taken that there is some apt useful discrete distribution. Proceeding with Poisson on that basis, the only parameter needed for Poisson is λ an expected number of tied elements. That λ can be gained from literature or counted. In the application case below the expectation of τ is 3 so $\lambda = 3$. Denote the Poisson tied element distribution $\mathbb{E}$, because T, t and e are taken. Conceivably other distributions could be used, for example if grouping of tied elements were plausible one might reach for the negative binomial.

3.3.4 Nice behaviour at limits

Poisson has one nice feature that it covers the MWU method if $\lambda = 0$, indeed it behaves as drawn in Figure 11. It also tends towards a plausibly symmetric distribution of tied elements around a central tendency once there are many tied elements.

3.3.5 Nice computation

Poisson has another nice feature that in a given problem, λ and of course e the universal constant do not change, so they can be represented by an unknown irrational real constant z which is cancelled out in these calculations, which are concerned with relative weight, see Equation 3 for the exact probability calculation. This reduces the calculation of each relative probability of each count of tied elements to one exponent and one factorial as simplified in Equation 3. Furthermore, the previous term of x requires calculation of the factorial $x - 1$, so each term only requires one multiplication for each additional factorial term, see Equation 3.

3.3.6 Unrefuted by present data

Furthermore, *a priori* fitting of $\mathbb{E}_{\lambda 3}$ to the current data does not refute it, see Figure 13, discussed at section 4.1.

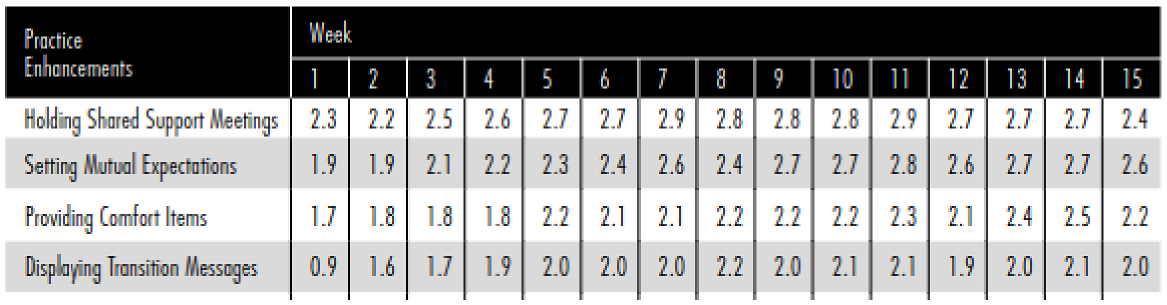
$$p(Poiss) = \frac{\lambda^k e^{-\lambda}}{k!} = z\frac{\lambda^k}{k!}; \qquad k! = k(k-1)!; \qquad z \in \mathbb{R} \tag{3}$$

3.3.7 Parsimony

All that is needed to build the appropriate truncated Poisson distribution is λ which is available the data, and n which is trivially available too. In some sense the author is encouraged that the information for $\mathbb{E}_\lambda$ is easily inherent in the data, leaving less moving parts.

4 Application

Safewards is a suite of interventions for reducing conflict and containment on wards (Bowers, 2014). Carter et al adapted and implemented Safewards *inter alia* measuring take-up of ten interventions (Carter et al., 2019). The first row records take-up of meetings, see Table 2 taken from the relevant published proceedings.



Practice Enhancements	Week														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Holding Shared Support Meetings	2.3	2.2	2.5	2.6	2.7	2.7	2.9	2.8	2.8	2.8	2.9	2.7	2.7	2.7	2.4
Setting Mutual Expectations	1.9	1.9	2.1	2.2	2.3	2.4	2.6	2.4	2.7	2.7	2.8	2.6	2.7	2.7	2.6
Providing Comfort Items	1.7	1.8	1.8	1.8	2.2	2.1	2.1	2.2	2.2	2.2	2.3	2.1	2.4	2.5	2.2
Displaying Transition Messages	0.9	1.6	1.7	1.9	2.0	2.0	2.0	2.2	2.0	2.1	2.1	1.9	2.0	2.1	2.0

Table 2: First four rows of ten rows of data from Carter et al 2019

Carter's data allows comparison between the first $[2.3, 2.2, 2.5...]$ and last $[...2.7, 2.7, 2.4]$ five weeks. Call the first five weeks S and the last T. Again for simplicity allow that "take up" may only go up, and have a one-tailed test. At Figure 12 is seen the path of the data in the top row, put in order for each, having a U score of 4.5.

$$S\ [2.3, 2.2, 2.5, 2.6, 2.7]$$

$$T\ [2.9, 2.7, 2.7, 2.7, 2.4]$$

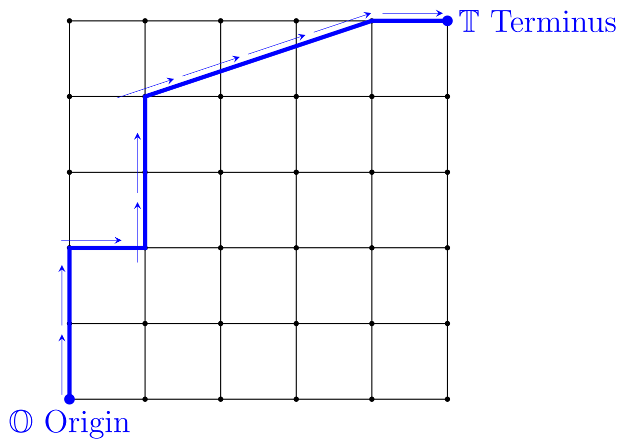


Figure 12: Carter et al data represented as lattice path, note ties are diagonal

4.1 Attempting to refute $\mathbb{E}_{\lambda 3}$

In this $U_{5,5}$ problem, perhaps due to a decision to round all values to $1s.f.$, there are three tied elements over the resultant N, and all are values of 2.7. It has been deduced that the distribution of counts of tied elements follows $\mathbb{E}$. That invites an estimate of λ.

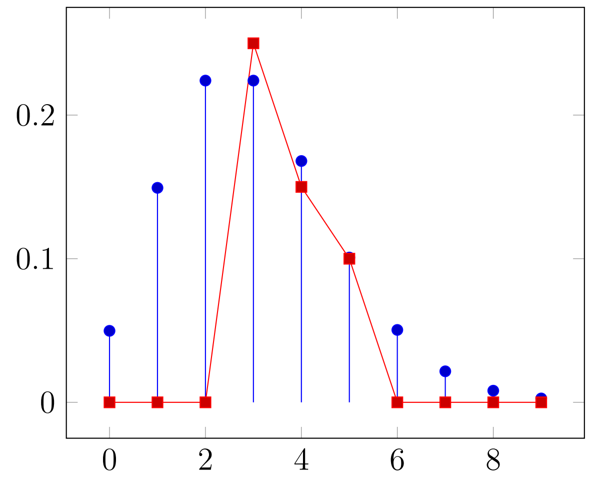


Figure 13: Poisson distribution $\mathbb{E}_\lambda 3$ vs real distribution of tied elements in Carter's ten rows of data

Figure 13 depicts counts of tied elements in ten rows from Carter's table (Carter et al., 2019). Three is the modal value. Plug in $\lambda = 3$ arriving at $\mathbb{E}_\lambda 3$ the distribution. Blue is $\mathbb{E}_\lambda 3$. Red is the number of tied elements in Carter's ten rows. Visually they match reasonably well, not refuting the deduction. Analysis proceeds based on $\mathbb{E}_\lambda 3$.

5 Presence-pairs

5.1 Generally

Any path such as at Figure 12 has steps. Any step has at least one element from S or T. Binary numbers can be built of 1 and 0 digits. A pair of binary numbers describing N, may write where any step has any element present in S or T using 1. Then the pair of numbers representing S and T between them must have at least 1 from either group over each step in their length. Length of S or T binary numbers we denote q. Due to the pigeonhole principle, tied elements deterministically shorten the path and $q = n - \tau$. Each step-digit has a 1 in either S or T, or there is nothing to be in the step. Viable paths must have either an s OR a t at every step, as in "inclusive OR". The number of steps in the shortest path under isomoprphism is reduced when two steps have the same angle. The length of strings q is any number up to n, which we know to be $s + t$. All possible binary number-pairs cover all paths.

5.2 Specifically

Such notation of S in Carter's overall presence in the order over the seven steps in Carter's path gives the binary number 1101110. For example this means the last step does not have an s element in it hence the end 0. The presence of T over the order is notated by the binary number 0010011.

$$\begin{array}{lcccccccl}
S\ [& 2.2 \mid & 2.3 \mid & \mid & 2.5 \mid & 2.6 \mid & 2.7 & \mid &] \rightarrow 1101110 \quad q = 7 \\
T\ [& \mid & \mid & 2.4 \mid & \mid & \mid & 2.7\ 2.7\ 2.7 & \mid 2.9 &] \rightarrow 0010011 \quad q = 7
\end{array}$$

5.3 Elaborating

For clarity, when they are superimposed, $\substack{S \\ T}\ \substack{1101110 \\ 0010011}$, the observer can see that there is a 1 from either all the way through from left to right, thus meeting bitwise-OR, the condition that each binary has a 1 OR the other does. In T, for Carter's path, $s = 5$ are "pigeonholed" together in $k = 3$ positions and share positions because $k < s$.

5.3.1 Further shortening under isomorphism

Further shortening under isomorphism happens in Carter's path. Though it was generated by $q7$ binary numbers it looks like it has five steps. The first and third steps of the apparent $q5$ simplest isomorphic path are made of two successive steps with the same angle - a vertical t-step.

5.3.2 Special case of τ_0

When $\tau = 0$, then $q = n$ and the pair of paths meets XOR, as in "exclusive or". This is what happens when there are no tied elements expected. Isomorphism plays out over all paths generated by $q10$ bit strings, which is the MWU case seen above. This leads to generation of all Manhattan Young diagrams. It is is the special circumstances in which MWU is exact. Such pairs are a subset of pairs which meet OR because logically any XOR pair meets OR.

6 Algorithmic generation of all binary pairs

6.1 Napkin-based and Q-notation

A reasonably efficient and interesting method for generating all OR-pairs re-uses the "napkins on a table" method. It interprets the heights as a different notation, namely Knuth's Q-notation, and cycles between that and left-shifting the strings, which can be discussed in more detail in papers focussed on algorithms. For a discussion of Q-notation see (Knuth, 2005, Ch.7). This also appears to be a an (n, k) solution, as in how to generate all bitstrings of length k and having k 1s. It is available on request. It generates all such orders in a different pattern from those in Knuth's comprehensive treatment.

6.2 Tree of appending $\begin{smallmatrix} S & 1 \\ T & 0 \end{smallmatrix}$, $\begin{smallmatrix} S & 1 \\ T & 1 \end{smallmatrix}$ or $\begin{smallmatrix} S & 0 \\ T & 1 \end{smallmatrix}$

A better one uses decision trees appending only $s1$ and $t1$; or $s0$ and $t1$; or $s1$ and $t0$; within the constraints of j, k, q, which is available on request, or at the section "Code". It would be able to take the q_7 pattern $\begin{smallmatrix} S & 1101110 \\ T & 0010011 \end{smallmatrix}$ and append $\begin{smallmatrix} S & 0 \\ T & 1 \end{smallmatrix}$ to make $\begin{smallmatrix} S & 11011100 \\ T & 00100111 \end{smallmatrix}$, without breaking limits on j, k or q. It only builds pairs of strings that work, which is efficient. The pairs of binary numbers when plotted make a pleasing Sierpińskioid pattern, named Triangular OR-wise Binary Integer Algorithm Sierpińskioids, ("TOBIAS") triangles. Code for this is included and here is an example of a TOBIAS.

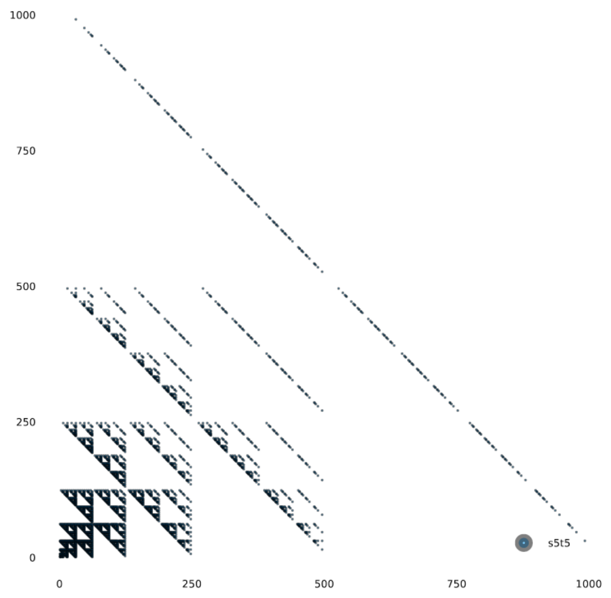


Figure 14: A Cartesian representation of the pairs which meet bitwise OR; Triangular OR-wise Binary Integer Algorithm Sierpińskioids; the same nodes on binary axes are topologically similar but the angles change.

6.3 Partition: share out the elements

6.3.1 Procedure

Once all binary number-pairs are made, "partition" or share s or t between the j or k 1 digits. Once the 1s are in place the elements can be shared between them. In T where $k = 3$ the $s5$ can be shared as $[3, 1, 1]$,$[2, 2, 1]$ etc.... or $[1, 1, 3]$. Partitioning is well attested in the literature on the mathematics underlying the types of combinations that arise in such statistics (Stanley, 2011, 1.73) (Knuth, 2005, Ch.7). The generation of presence-pairs and then partitioning above generates all paths of starting length q.

6.3.2 Isomorphism within q-class

Call such a group made from binary pairs with length q a q-class of paths. All paths are in a q class between 1 and n, so progressing q from 1 to n builds all paths. Isomorphic paths occurs within a q-class. See, above that in $s5t5q10$ there were #16 ways to form the singleton $U = 0$ Young diagram and the same repetition occurs in other q-families. q_1 for example, regardless of how likely it is, has all elements in one step and is always a single straight line bisecting the lattice and always has $U = \frac{s \times t}{2}$. The function in the code called denoding removes extraneous nodes with no information, which sit on a straight line. It looks for successive steps with the same angle and deletes the node between them. Removing these leaves the simplest isomorphic path. If it is already on the stack for the q-family do not add it. Some paths appear in on more than one q class. For the distribution within a q-class stick to one q value. The q-class stacks other than q_0 will generally have some duplicates which need to be discarded due to isomorphism. This occurred in MWU which is the special case τ_0.

6.3.3 All paths collapsed under isomorphism is checkable

Each arising path can be added to the stack of paths if it is a new distinct paths under isomorphism within paths generated by a given q. If the observer wants to work out all paths in total, have one stack, and for $s5t5$ they will get the answer 5321 in McMeekin Hill, see Table 3.

6.3.4 Benefit

The point is that all of this is decidable, therefore exact, and very much quicker than iteratively working out all possible paths including ties. Each q-class has its own distribution of U scores. They need to be added with the right weights informed by $\mathbb{E}$. This allows all paths to be generated, for example in the case of $\mathbb{U}_{s5,t5}$ all distinct paths under the assumption of three steps, Figure 15. Note that $U = 0$ is not part of the $s5t5q3$ distribution and so an exact test of stochastic order must take account of that.

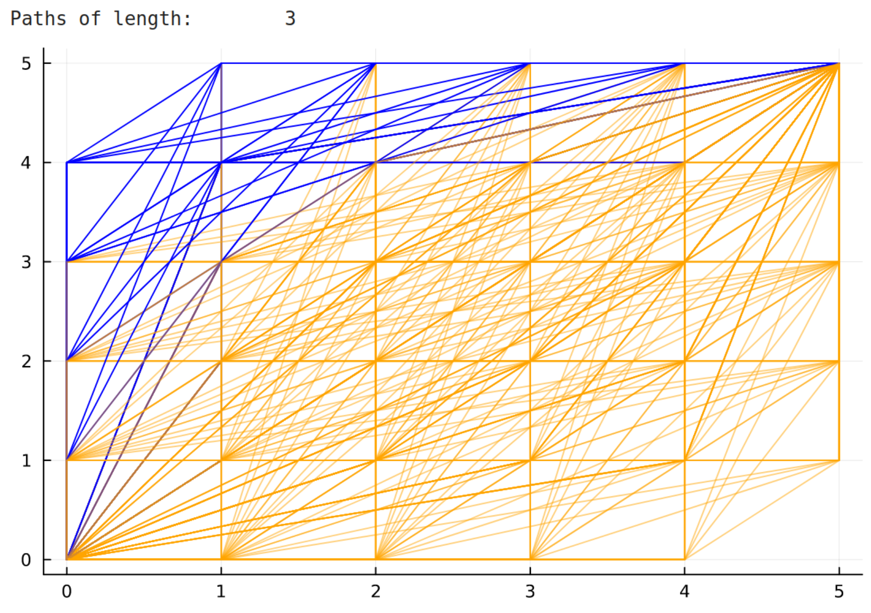


Figure 15: All paths in $s5t5q3$, i.e. all paths across a 5×5 lattice with three steps.

6.4 Apply Poisson weightings

$\mathbb{E}$'s prediction of likelihoods of numbers of tied elements τ is equivalent to predicting relative proportions of families of paths built assuming different $q \in [1 : n]$. That is because tied elements deterministically shorten paths, due to the pigeonhole principle. Predictions of one predicts the other. Hence if $\mathbb{E}$ predicts that, e.g. 20% of paths will have $\tau = 4$ tied elements, it equivalently predicts that those 20% form paths which have, in $\mathbb{U}_{5,5}$, length $q = n - \tau = q6$. Call such a group a q-class of paths. Each q-class has it own local distribution of paths. Some have $U' \leq U$. Correctly weighting each q-class using $\mathbb{E}$ gives an exact distribution. This gives an exact expectation of the correct weighting of paths and a reliable probability which handles ties. It is computationally cheap enough to avoid approximations.

7 Weighted Truncated Poisson

This section applies Poisson to the different q-classes. $\mathbb{E}$ is a discrete distribution informing expectations $\tau = [0, 1, 2...9]$ per row. The most common value for τ is 3 in Carter's data and this was chosen as λ in Figure 13. Poisson extends into infinity on the x axis. But $\mathbb{E}$ is only defined in the range $[0, 1..n - 1]$. There cannot be ten tied elements in ten elements. Therefore $\mathbb{E}$ is a truncated Poisson distribution.

Truncation means each set of paths with the same τ gets a bigger slice of the total probability. In $s5t5\lambda3$ they are all inflated by being divided by 0.998.... That is the asterisked cumulative probability of $\tau = 9$ in Figure 16, the limit of truncation is given by n.

About 0.224 of the paths should have 7 steps, or $\tau = 3$ tied elements, see this is the figure under Exact Probability. Each path family has a fixed number of possible orders some of which are $U \leq 4.5$. The probability *within* $s5t5q7$ paths is 0.0322... and if we knew that only $\tau = 3$ i.e. $q = 7$ was possible, that would be the overall probability. The various probabilities with the different q-path families must be weighted using the weighting in Figure 15. They are presented here in decimal form but the discrete nature of all underlying structures means that they are rational numbers and this adds to the exactness, particularly in language such as Julia which allows rational typing. Section 8 and 9 may be safely ignored by those disinterested in further algorithmic efficiency.

Tied elements	Exact Probability	Cumulative	Weighting	Weighted
$\tau = 0$	0.0497...	0.0497...	$\frac{0.0497...}{*0.998...}$	0.0498...
$...\tau = 1$	0.149...	0.199...	$\frac{0.149...}{*0.998...}$	0.149...
$...\tau = 2$	0.224...	0.423...	$\frac{0.224...}{*0.998...}$	0.224...
$...\tau = 3$	0.224...	0.647...	$\frac{0.224...}{*0.998...}$	0.224...
$...\tau = 4$	0.168...	0.815...	$\frac{0.168...}{*0.998...}$	0.168...
$...\tau = 5$	0.100...	0.916...	$\frac{0.100...}{*0.998...}$	0.100...
$...\tau = 6$	0.0504...	0.966...	$\frac{0.0504...}{*0.998...}$	0.0504...
$...\tau = 7$	0.0216...	0.988...	$\frac{0.0216...}{*0.998...}$	0.0216...
$\tau = 8$	0.00810...	0.996...	$\frac{0.00810...}{*0.998...}$	0.00811...
$\tau = n - 1 = 9$	0.00270...	*0.998...	$\frac{0.00270...}{*0.998...}$	0.00270...

Figure 16: Poisson $\lambda = 3$ distribution up to $\tau = n - 1 = 9$ the maximum in Carter's data, showing unrounded first three figures, with differences at τ_8 and τ_0

8 Checking

Collapsing all q-classes given s, t together builds all isomorphically distinct diagonal paths across a lattice of dimensions s, t. A visual check of how many paths with ties there are generates Table 3. McMeekin Hill is the vanity name for that additive structure, for the current author's doctoral advisors and in homage to Pascal's Triangle.

Note, the first instantiation of this structure was made by calculating all paths using presence-pairs, not by addition. This was very inefficient, analogous to generating all Young diagrams to build Pascal's triangle, instead of adding pairs of numbers. It gives #5321 paths across $s5t5$. In Table 3, 5321 is five along and five down in bold, echoing the position of $|\mathbb{U}_{5,5}| = \#252$ in Pascal's triangle. This has led to a peer reviewed "integer list" and visual check for the current author which supports these methods. This is registered under the present author's name on the peer-reviewed Online Encyclopedia of Integer Series ("OEIS") (Reid, 2023).

						1						
					1		**1**					
				1		**3**		**1**				
			1		**6**		**6**		**1**			
		1		**10**		$\underline{\mathbf{17}}$		**10**		**1**		
	1		**15**		**39**		$\underline{\mathbf{39}}$		**15**		**1**	
1		**21**		**76**		$\underline{\mathbf{111}}$		$\underline{\mathbf{76}}$		**21**		1
	28		**135**		**266**		$\underline{\mathbf{266}}$		**135**		28	
...		222		**566**		**757**		**566**		222		...
	...		1100		**1876**		**1876**		1100		...	
...		...		4197		**5321**		4197		...		...
	...		...		13469		13469		...		...	
...		...		...		...		...		...		...

Table 3: McMeekin Hill gives $|\mathbb{U}|$ for lattices across paths allowing ties; this is rotated 45 degree anticlockwise from Figure 17. 266, 111, 17, 29 and 76 are underlined for orientation

8.1 General pattern

Similarly to the huge simplification offered by the recurrent addition of two parent nodes in Pascal's triangle, there is a pattern in McMeekin Hill. The pattern is that all "visible" steps are allowed to be the next step onward to Terminus. Just as the numbers in Pascal's triangle are sums of two parent nodes, the numbers in McMeekin Hill are the sum of all *visible* nodes. The step for any node is only those that can be "seen", given that some nodes occlude each other. The originating node in question may be denoted (x, y). What defines the options onward from there is that any allowed next step is a coprime pair (w, v), for width and vertical displacement, fitting in the lattice. These are all also the simplest fraction $\frac{w}{v}$. All fractions of this type are co-prime.

8.1.1 Example from perspective of #266

Figure 18 shows displacement from $(x, y) = (0, 0)$ in an $s3t4$ lattice. Table 3 implies #266 possible routes across an $s3t4$ lattice. See that #266 on McMeekin Hill can go up directly to #39. It cannot up 2, to the node two above which has #6 paths home, because that node "cannot be seen" it is occluded. Not occluded 1, 10, 39, etc. Adding all the nodes it *can* see gives its value, echoing Pascal's triangle. Pascal's triangle's geometry only allows the two parents above to be "seen" but McMeekin Hill allows all directly visible ancestors to be seen.

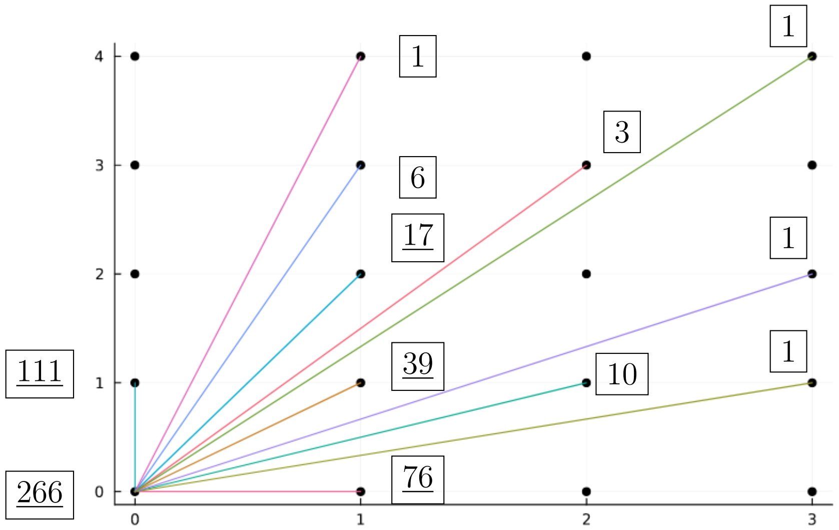


Figure 17: From "266" occlusion excludes paths home; this is rotated 45 degrees clockwise relative to Table 3; 266, 111, 17, 39, 76 are underlined to aid orientation. All the lines are vectors (w, v) which arise in the Stern Brocot tree

8.2 Generate non-occluded paths

It is now convenient to generate all non-occluded paths. This procedure may generate all shortest steps between any two nodes. Vectors from node displaced in width and vertically by (w, v) where w and v, being co-prime, are generated as pairs in the the Stern Brocot tree. Clearly the vectors cannot leave the lattice either.

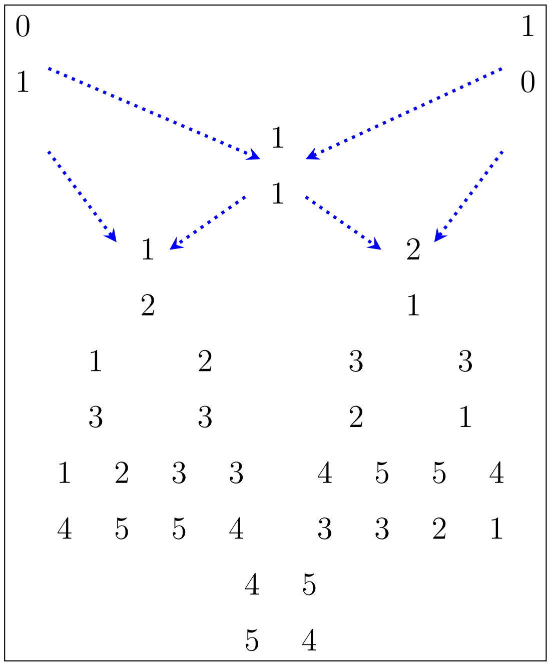


Figure 18: Stern Brocot tree grows by mediant additions and generates all vectors (w, v)

9 Efficiencies

Generating the Stern Brocot tree or McMeekin Hill requires simple addition with a recursion. Further secondary optimisations are now laid out such as exploiting hypercurves, cellular automata and lattice recursions. For the standard Young diagrams above it was noted that generating all unique young diagrams (or isomorphic paths) in a distribution only at the tail from $U' = [0, 1, 2...U]$ gave the numerator efficiently. Then Pascal's triangle gave the denominator. Certain efficiencies which may have a similar effect in tied situations. The series of possible U in tied situations is not $U' = [0, 1, 2...U]$ but is $U' = [0, 0.5, 1, 1.5, 2.5...U]$. It is possible to make these in an efficient way to gain the numerator, with a q-family of paths.

9.0.1 Exploiting Intermediate Value Theorem

Given that paths start at Origin and Traverse to Terminus, *any and all* two paths with the same U must cross. This may feel intuitively true but is proven as the Intermediate Value Theorem (Borowski and Borwein, 2002, p.291). Thus if we can make one prototypical "index" path or curve sweeping past the corner with a given U', all paths with the same U' must cross that index path. Because such paths are made of individual steps, any crossing step is between two nodes. The crossing point between two nodes must the shortest paths between those two nodes due to Euclidean axioms.

Any paths will serve as the one which others cross, the index curve. One may identify all individual steps that cross the index curve, if their value in terms of $y = mx + c$ in the lattice treated as a Cartesian graph equals that value. Smooth paths with a given U may be built with parametric functions.

Hypercurves are quadrants of a hypercircle. A circle has a defining function based on the power of 2. If we generalise using the power a in place of 2 to a, then the "circle" becomes more "boxlike" as a increases, and more flattened and diamond-like as a decreases to 1.

9.0.2 Hypercurves as index curves

Hypercurves give a pleasing line and hug the corner. The right a value gives the right value of a if the curve is scaled correctly, see Figure 18. All lines generated between any node and any visible next node can be assessed as to whether they cross the parametric function. Call the parametric function the "index" function.

9.0.3 Minimum space U implied by crossing line

In Figure 18 all shortest paths with $U = 1$ are caught in this way, in orange, due to the Intermediate Value Theorem. Some such candidate lines which cross the hypercurve can never make U and are discarded. They are "bust" and already subtend more than U. See Figure 19 where the orange lines can make $U = 1$ but the horizontal red line crosses the index but is bust.

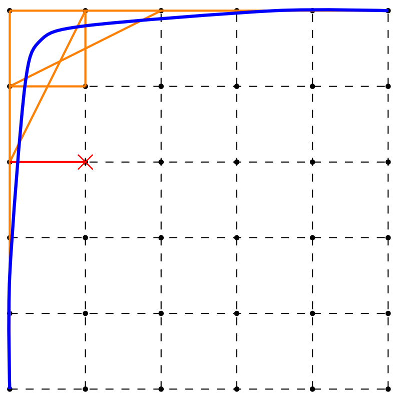


Figure 19: In this schematic if the blue hypercurve has $U = 1$; orange lines cross it and can be part of paths with $U = 1$; the red dashed line is “bust” due to minimum viable $U = 2$

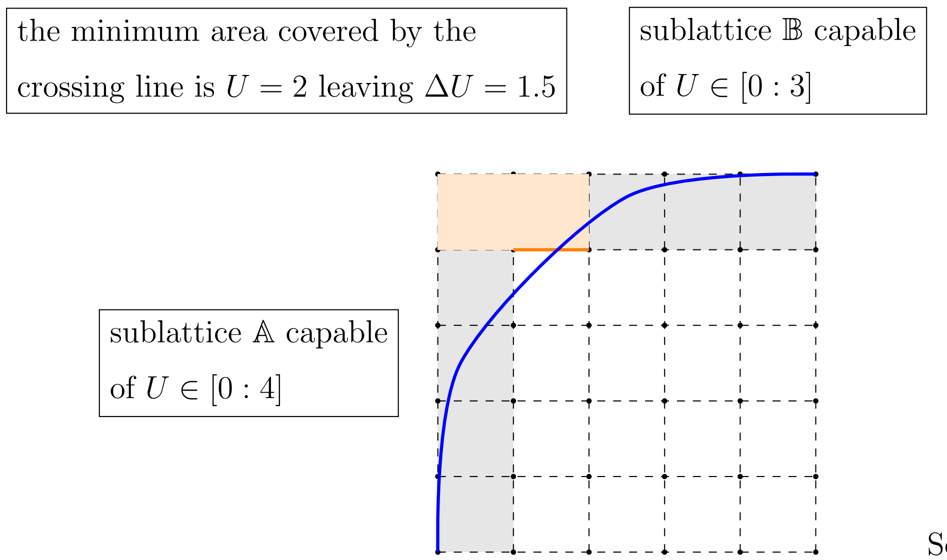
9.0.4 Automata partition residual space ΔU

Any crossing line which is not bust remains declared in the lattice. It has a residual area ΔU which must be provided. Alternately, lines that are bust have a negative value for $DeltaU$ The crossing line has two loose ends, before and after the crossing line.

These can be called $\mathbb{A}$ “Alice” and $\mathbb{B}$ “Bob” according to negotiation conventions and solved by treating them as negotiating automata. $\mathbb{A}$ and $\mathbb{B}$ can negotiate all possible solutions providing starts and ending which meet U exactly. A total path which is a viable solution arising from such as negotiation goes from Origin through any A line and then the “crossing line” and then the B line then on to Terminus. Again this is a fairly decidable problem and indeed the same calculations keep appearing and can be cached, or saved for reference, speeding things up.

The added constraint of only working within a q-class speeds this up. The numerator and of the q-class can be decided using presence pairs and partitions. Then all paths crossing the hypercurve can be caught, and the ones which are not bust can be negotiated with the remain steps using q shared between $\mathbb{A}$ and $\mathbb{B}$. This can be achieved recursively when the two sublattices themselves are subject to crossing lines and automata. It is just an efficiency; calculating all $s5t5$ paths then weighting by Poisson without this efficiency remains rapid on old and obsolete domestic laptops which break with naive methods.

Furthermore, each lattice can be analysed as a lattice, and some point the choice degenerate to a few basic solution which can be cached such as $U = 0$ or its compliment $U(s \times t) - 1$, particularly at Figure 22.


the minimum area covered by the crossing line is $U = 2$ leaving $\Delta U = 1.5$
sublattice $\mathbb{B}$ capable of $U \in [0:3]$
sublattice $\mathbb{A}$ capable of $U \in [0:4]$

Section 8 and 9 may have been safely ignored by those only interested in the answer.

Figure 20: In this schematic seeking to build $U = 3.5$ paths if the blue hypercurve has $U = 3.5$; one orange line crosses it $U = 3.5$; the earlier automaton $\mathbb{A}$ and $\mathbb{B}$ can meet the residual $+U = 1.5$ by sharing the burden

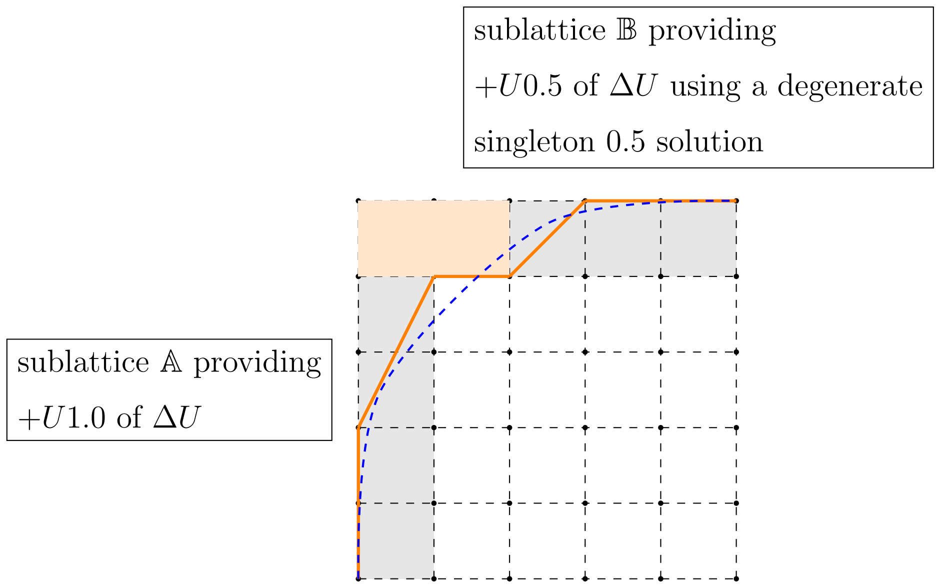

sublattice $\mathbb{B}$ providing $+U0.5$ of ΔU using a degenerate singleton 0.5 solution
sublattice $\mathbb{A}$ providing $+U1.0$ of ΔU

Figure 21: $\mathbb{A}$ and $\mathbb{B}$ may meet residual $+U = 1.5$ meeting $q5$

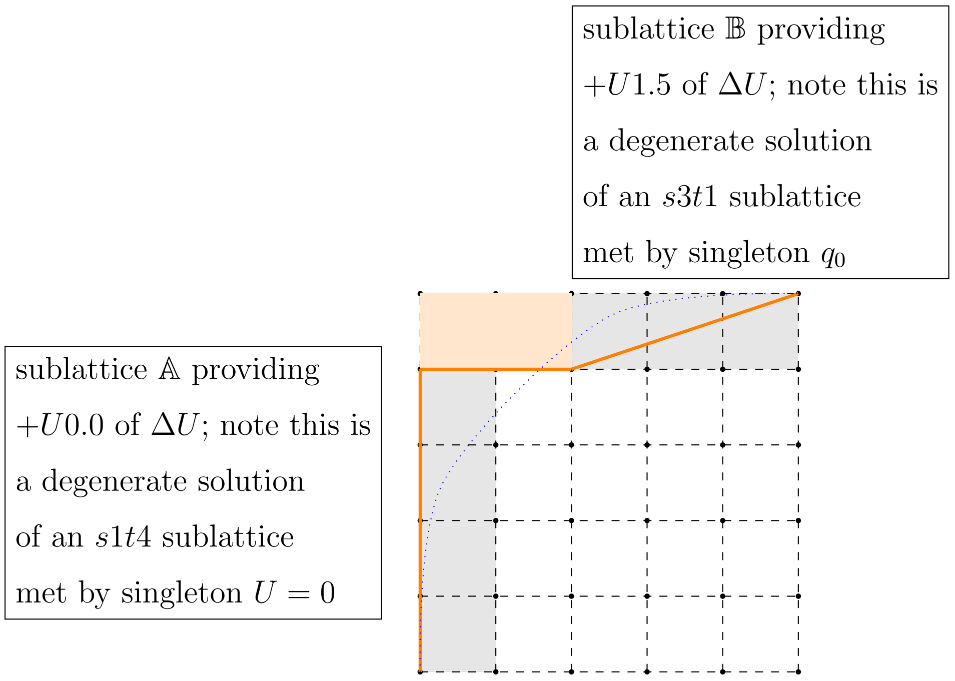

sublattice $\mathbb{B}$ providing $+U1.5$ of ΔU; note this is a degenerate solution of an $s3t1$ sublattice met by singleton q_0
sublattice $\mathbb{A}$ providing $+U0.0$ of ΔU; note this is a degenerate solution of an $s1t4$ sublattice met by singleton $U = 0$

Figure 22: $\mathbb{A}$ and $\mathbb{B}$ may meet residual ΔU in a solution that meets $q3$ and also relies singletons which can be cached

9.1 Using MWU, assuming no tied elements

Section 8 and 9 may have been safely ignored by those only interested in the answer. Plugging Carter's data into a current online statistical engine gives one-tailed $p = 0.0582$ (Lowry, 2022). That exceeds conventional $\alpha = 0.05$, so the test of take up is "not significant".

9.2 Napkin

Using the present method called Napkin Analysis of Paths using K-Integer Nodes "Napkin" the one-tailed probability is $p = 0.040$, below conventional α. The result may seemed moved into "significance" by this more exact procedure. The story is *post hoc* and a boundary case was chosen for demonstration. The assumption that take-up of meetings only goes up is a fiction.

9.3 Implications

In conclusion, MWU is the subclass of Napkin which does not expect ties, weighting all expectations to the class of Young diagrams with no slopes. Future testing of order may be more exact and handle ties. Experiments can stop earlier, if a more powerful model relaxes power calculations. In the game-theoretical terms of the present author's vocation, called "adapanomics", this maximises value and information, leading to less cost and potential harm to subjects. Future L-tests may use Napkin. It arose from adapanomics's concern for thoughtful numeric analysis and relieves L-test of an inherited approximation.

10 Discussion

In summary a desire to escape some issues inherited by L-test from MWU has led to an interesting and ultimately productive foray into the underlying methods, underpinning the hoped-for benefits.

11 Postscript

The splitting up or exclusion of paths is called "way of cutting paths" or "*nasin pi kipisi nasin*" in Toki Pona a language aimed at simplicity, and is abbreviated to Napkin for that reason also.

12 Code

This demonstrative code was coded on an iPhone using Pythonista. Much of the author's own code base in is Julia. Writing at again gave the same outcome.

12.1 Libraries

These in built sets of functions and objects are standard in Python.

```
# s.0  LIBRARIES

from collections import Counter
from itertools import permutations
from itertools import compress
from math import factorial
import math
import matplotlib.pyplot as plt
import numpy             as np
```

12.2 Classes

These data objects represent the representation of any order as a path. In turn the path is represented by a pair binary numbers. Any 1 in such binary numbers denotes the presence of any element. All paths must have steps. All steps must have an element.

```
# s.1 CLASSES


class PairST:
        def __init__(self, pair, st):
                self.spart = pair[0]
                self.tpart = pair[1]
                self.s     = st[0]
                self.t     = st[1]
                self.q     = len(pair[0])
                self.j     = self.spart.count('1')
                self.k     = self.tpart.count('1')

        def tell(self):
                print('q', self.q, 's', self.s, 't', self.t,
                'spart', self.spart, 'tpart', self.tpart,
                'j', self.j, 'k', self.k)

class QTally:
        def __init__(self, q, U):
                self.q = q
                self.U = U
```

12.3 Configuration

In this implementation these configurable values pertain to teh csize or cardinal number of S and T. They also pertain to l meaning λ for the expectation of tied elements in in $\mathbb{E}$. This implementation handles $found_U$ as hard-wired variable.

```
# s.2 CONFIGknuth_art_2005

def get_found_U():
        found_U = 2
        return found_U

def get_l():
        l = 3
        return l

def get_s():
        s = 2
        return s

def get_t():
        t = 5
        return t
```

12.4 Model

This section named following the Model-View-Control architectural paradigm holds the working combinatorial parts.

```
# s.3 MODEL

def build_onewise_parts(binary, partitions):
  onewise_parts = []
  for int_partition in partitions:
    this_onewise   = []
    this_partition = int_partition.copy()
    for i in binary:
      if i == '0':
        this_onewise.append(0)
      else:
        next_ = this_partition.pop(0)
        this_onewise.append(next_)
    onewise_parts.append(this_onewise)
  return onewise_parts

def count_ustat(path):
  origin   = [0,0]
  terminus = path[-1]
  #print("origin", origin, "terminus", terminus)
  ustat    = 0
  edges    = []
  for i in range(0, len(path)-1):
    first = path[i]
    other  = path[i+1] # "next" is reserved
    #print("first", first, "other", other)
    box_x = other[0]    - first[0]
    box_y = terminus[1] - other[1]
    tri_x = other[0]    - first[0]
    tri_y = other[1]    - first[1]
    box   = box_x * box_y
    tri   = tri_x * tri_y * 0.5
    inc   = box + tri
    ustat = ustat + inc
  return ustat

def denode(before):
  long_xs = before[0]
  long_ys = before[1]
  if len(long_xs) == 2:
    return before
  else:
    long_nodes = [list(x) for x in zip(long_xs, long_ys)]
    long_x_gaps = [long_xs[i] -
    long_xs[i-1] for i in range(1,len(long_xs))]
  long_y_gaps = [long_ys[i] -
  long_ys[i-1] for i in range(1,len(long_ys))]
  long_xy_gaps = [list(xy) for xy in
  (zip(long_x_gaps, long_y_gaps))]
  gap_gcds      = [math.gcd(xy[0], xy[1]) for
  xy in long_xy_gaps]
  coprimes = [[long_xy_gaps[i][0]/gap_gcds[i],
  long_xy_gaps[i][1]/gap_gcds[i]]
  for i in range(len(gap_gcds))]
  diff_from_next = [coprimes[i] != coprimes[i+1]
  for i in range(len(coprimes)-1)]
  node_path = [[0,0]] # origin of any path
  terminus = long_nodes[-1] # terminus of any path
  inters = long_nodes[1:-1]
  assert len(diff_from_next) == len(inters),
  'inters or diff bools wrong'
  keepers = list(compress(inters, diff_from_next))
  for keeper in keepers:
    node_path.append(keeper)
  node_path.append(terminus)
  path_xs = [xy[0] for xy in node_path]
  path_ys = [xy[1] for xy in node_path]
  after   = [path_xs, path_ys]
return after
```

```
def get_combs(s_xor_t, current=None):
  if current is None:
    current = []
  current_sum = sum(current)
  if current_sum == s_xor_t:
    yield current
    return
  for n in range(1, s_xor_t - current_sum + 1):
    yield from get_combs(s_xor_t, current + [n])

def get_parts(s_or_t):
  combs = get_combs(s_or_t)
  parts = list(combs)
  return parts

def rel_poisson(k,l):
  weight = (l**k)/factorial(k)
  return weight

if __name__ == "__main__":
  s           = get_s()
  t           = get_t()
  l           = get_l()

  st          = (s, t)
```

```
n           = s+t
tmax        = n-1

s_parts     = get_parts(s)
t_parts     = get_parts(t)

tau_range   = range(n)
rel_freq    = [rel_poisson(k,l) for k in tau_range]
rel_sum     = sum(rel_freq)
normed_freq = [x/rel_sum for x in rel_freq]
print(normed_freq)
plt.plot(normed_freq, linestyle ='dashed')
plt.scatter(range(n),normed_freq)
plt.show()
plt.cla()

print(sum(normed_freq))
assert math.isclose(sum(normed_freq), 1.00, rel_tol=1e-5),
'mistruncated'

options = [['0','1'],['1','0'],['1','1']]
q = 1
register = options.copy()
while q < n:
  for qstr in [x for x in register if
  len(x[0]) == q]:
    for option in options:
      new_s = qstr[0]+option[0]
      new_t = qstr[1]+option[1]
      new   = [new_s,new_t]
      if new_s.count('1') <= s:
        if new_t.count('1') <= t:
          register.append(new)
  q=q+1

noded_q_paths = []
for pair in register:
  pair_st = PairST(pair, st)
  if pair_st.j > 0:
    if pair_st.k > 0:
      j_parts = [x for x in s_parts
      if len(x)==pair_st.j]
      k_parts = [x for x in t_parts
      if len(x)==pair_st.k]
      s_owp = build_onewise_parts(pair_st.spart, j_parts)
      t_owp = build_onewise_parts(pair_st.tpart,
      k_parts)

      for this_s_owp in s_owp:
        for this_t_owp in t_owp:
          appendage = [pair_st.q,
          this_s_owp, this_t_owp]
          noded_q_paths.append(appendage)

denoded_q_paths = []
for this_q_path in noded_q_paths:
  s_owp = this_q_path[1] # note space for q
  t_owp = this_q_path[2] # note soace for q
  s_distal = np.cumsum(s_owp)
  t_distal = np.cumsum(t_owp)
  s_path   = [0]
  t_path   = [0]
  for i in s_distal:
    s_path.append(i)
  for i in t_distal:
    t_path.append(i)
  noded   = [s_path, t_path]
  denoded = denode(noded)
  s_path  = denoded[0]
  t_path  = denoded[1]
  this_q  = this_q_path[0]
  appendage = [this_q, s_path, t_path]
  denoded_q_paths.append(appendage)

print("This many denoded q_paths:", len(denoded_q_paths))
```

```
unique_denoded_q_paths = []
for i in denoded_q_paths:
  if i not in unique_denoded_q_paths:
    unique_denoded_q_paths.append(i)

print("This many unique denoded q_paths:", len(unique_denoded_q_paths))

q_max = 0
all_qtallies = []
unique_denoded_q_paths.sort()
for i in unique_denoded_q_paths:
  this_q     = i[0]
  if q_max < this_q:
    q_max = this_q
  #print(this_q)
  this_path = [list(x) for x in zip(i[1],i[2])]
  ustat      = count_ustat(this_path)
  #print("ustat", ustat)
  qtally     = QTally(this_q, ustat)
  #print(qtally.q, qtally.U)
  all_qtallies.append(qtally)
  #plt.plot(i[1],i[2])
  #plt.show()
  #plt.cla()
  for qband in range(1,q_max+1):
    #print(qband)
    right_q = [x.U for x in all_qtallies if x.q == qband]
    right_q.sort()
    mixed_pdf = Counter(right_q)
    print("mixed_pdf of ", qband, "is", mixed_pdf)
    someUs     = list(mixed_pdf.keys())
    freqs      = list(mixed_pdf.values())
    plt.scatter(someUs, freqs)
    plt.show()
    plt.cla()

    found_U = get_found_U()
    leq = [x for x in right_q if x <= found_U]
    print(len(right_q), "total")
    print(len(leq), "less than or equal")
    print("local q probability", len(leq)/len(right_q))
```

13 Test Driven Development

```
# TESTS

def test_build_onewise_parts():
```

```
s_parts = '110'
t_parts = '101'
j_parts = [[1, 1]]
k_parts = [[1, 2], [2, 1]]
s_onewise_parts = build_onewise_parts(s_parts, j_parts)
assert isinstance(s_onewise_parts, list)
assert s_onewise_parts == [[1,1,0]], 'wrong onewise'
t_onewise_parts = build_onewise_parts(t_parts, k_parts)
assert isinstance(t_onewise_parts, list)
assert t_onewise_parts == [[1,0,2],[2,0,1]], 'wrong onewise'

s_parts = '1'
t_parts = '1'
j_parts = [[2]]
k_parts = [[3]]
s_onewise_parts = build_onewise_parts(s_parts, j_parts)
assert isinstance(s_onewise_parts, list)
assert s_onewise_parts == [[2]], 'wrong onewise'
t_onewise_parts = build_onewise_parts(t_parts, k_parts)
assert isinstance(t_onewise_parts, list)
assert t_onewise_parts == [[3]], 'wrong onewise'

s_parts = '0011'
t_parts = '1111'
```

```
    j_parts = [[1,2],[2,1]]
    k_parts = [[1,1,1,1]]
    s_onewise_parts = build_onewise_parts(s_parts, j_parts)
    assert isinstance(s_onewise_parts, list)
    assert s_onewise_parts == [[0,0,1,2],[0,0,2,1]], 'wrong onewise'
    t_onewise_parts = build_onewise_parts(t_parts, k_parts)
    assert isinstance(t_onewise_parts, list)
    assert t_onewise_parts == [[1,1,1,1]], 'wrong onewise'
    print("passed build onewise parts")

def test_count_ustat():
    this_path   = [[0,0],[1,0],[2,2],[3,4]]
    xs = [xy[0] for xy in this_path]
    ys = [xy[1] for xy in this_path]
    #plt.scatter(xs,ys)
    #plt.show()
    #plt.cla()
    ustat       = count_ustat(this_path)
    # box + triangle per edge
    expected    =  (4+0)   + (2+1)  +  (0+1)
    assert ustat == expected

    this_path   = [[0,0],[0,1],[0,2],[2,3],[6,5]]
    xs = [xy[0] for xy in this_path]
    ys = [xy[1] for xy in this_path]
```

```
  #plt.scatter(xs,ys)
  #plt.show()
  #plt.cla()
  ustat         = count_ustat(this_path)
  expected      =   0   + 0   + 5   + 4
  assert ustat == expected

  this_path     = [[0,0],[1,1],[2,2],[3,3],[4,4]]
  xs = [xy[0] for xy in this_path]
  ys = [xy[1] for xy in this_path]
  #plt.scatter(xs,ys)
  #plt.show()
  #plt.cla()
  ustat         = count_ustat(this_path)
  expected      = 3.5 + 2.5 + 1.5 + 0.5
  assert ustat == expected
  print("passed ustat")

def test_denode():
  ref_befores = [
  [[0,1],          [0,1]],
  [[0,4],          [0,4]],
  [[0,8],          [0,10]],
  [[0, 1, 2, 4], [0, 1, 2, 4]],
  [[0, 1, 3, 4], [0, 1, 3, 4]],
[[0, 2, 3, 4], [0, 2, 3, 4]],
[[0,0,0,0,1,2,3],[0,1,2,3,3,3,3]],
[[0,0,1,2,3],[0,3,3,3,3]],
[[0,0,1,2,3],[0,2,3,3,3]],
]
ref_afters = [
[[0,1],           [0,1]],
[[0,4],           [0,4]],
[[0,8],           [0,10]],
[[0, 4], [0, 4]],
[[0, 4], [0, 4]],
[[0, 4], [0, 4]],
[[0,0,3],[0,3,3]],
[[0,0,3],[0,3,3]],
[[0,0,1,3],[0,2,3,3]],
]
refs = zip(ref_befores,ref_afters)
for ref in refs:
  before = ref[0]
  after  = ref[1]
  answer = denode(before)
  #plt.scatter(before[0],before[1])
  #plt.plot(after[0], after[1])
  #plt.scatter(after[0], after[1])
  #plt.show()
```

```
      #plt.cla()
      assert answer == after, "wrong compression"
    print("passed denode")

def tests():
    test_build_onewise_parts()
    test_count_ustat()
    test_denode()
    print("passed all tests")

#tests()

"""

    # these graphs are called TOBIAS graphs, triangular
    # Ordinal Binary and Serpienskioid graphs after
    # Dr Philip T Robinson
    # draw a nice graph
    for entry in register:
      if len(entry[0])%2==0:
        plt.scatter(str(entry[0]),
        str(entry[1]),c='orange',s=3)

      else:
      #  plt.scatter(str(entry[0]),
 #  str(entry[1]),c='blue')
    plt.scatter(str(entry[0]),
    str(entry[1]),c='blue',s=3)
plt.tick_params(axis='x', labelrotation = 270)
plt.show()

plt.cla()

n_reg = [x for x in register if len(entry[0])==n]

for entry in n_reg:
  if len(entry[0])%2==0:
    plt.scatter(str(entry[0]),
    str(entry[1]),c='orange',s=3)

  else:
 #  plt.scatter(str(entry[0]),
 #  str(entry[1]),c='blue')
    plt.scatter(str(entry[0]),
    str(entry[1]),c='blue',s=3)
plt.tick_params(axis='x', labelrotation = 270)
plt.show()

plt.cla()
```

```
sub_st = [x for x in register if len(entry[0])<=s]

for entry in sub_st:
  if len(entry[0])%2==0:
    plt.scatter(str(entry[0]),
    str(entry[1]),c='orange',s=3)

  else:
  #  plt.scatter(str(entry[0]),
  #  str(entry[1]),c='blue')
    plt.scatter(str(entry[0]),
    str(entry[1]),c='blue',s=3)
plt.tick_params(axis='x', labelrotation = 270)
plt.show()
"""
```

References

Borowski, J. and J. M. Borwein (2002, July). *Mathematics* (2nd ed.). Collins Dictionary of. London, England: Collins.

Bowers, L. (2014, August). Safewards: a new model of conflict and containment on psychiatric wards. *J. Psychiatr. Ment. Health Nurs. 21*(6), 499–508.

Bucchianico, A. (1999). Combinatorics, computer algebra and the Wilcoxon-Mann-Whitney test. *Journal of Statistical Planning and Inference 79*(2), 349–364.

Busner, J. and S. D. Targum (2007, Jul). The clinical global impressions scale: applying a research tool in clinical practice. *Psychiatry (Edgmont) 4*(7), 28–37.

Carter, S., E. Johnstone, and S. Canning (2019, October). Enhancing CAMHSs safety program with Safewards creating safer spaces for work and recovery. *Violence in Clinical Psychiatry. Eds: Callaghan, Patrick and Oud, Nico and Nijman, Henk and Palmstierna, Tom and Duxbury, Joy. Proceedings of the 11th European Congress on Violence in Clinical Psychiatry. 23–26 October 2019; Oslo 11th*, p. 112.

Dack, C., J. Ross, and L. Bowers (2012, September). The relationship between attitudes towards different containment measures and their usage in a national sample of psychiatric inpatients. *J. Psychiatr. Ment. Health Nurs. 19*(7), 577–586.

Dickens, G. L., T. Tabvuma, K. Hadfield, and N. Hallett (2020). Violence prevention climate in general adult inpatient mental health units: Validation study of the vpc-14. *International Journal of Mental Health Nursing 29*(6), 1101–1111.

Knuth, D. E. (2005). *The art of computer programming.* Upper Saddle River, NJ: Addison-Wesley.

Lowry, R. (2022). Mann Whitney U. `http://www.vassarstats.net/utest.html`. The candidate's statistical engine of choice, accessed up to submission.

Mann, H. B. and D. R. Whitney (1947, March). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics 18*(1), 50–60.

Marx, A., C. Backes, E. Meese, H.-P. Lenhof, and A. Keller (2016). Edison-wmw: Exact dynamic programing solution of the Wilcoxon–Mann–Whitney test. *Genomics, Proteomics & Bioinformatics 14*(1), 55–61.

McGee, M. (2018, July). Case for omitting tied observations in the two-sample t-test and the Wilcoxon-Mann-Whitney test. *PLoS One 13*(7), e0200837.

NLM, U. (2000). Pubmed central (PMC). `https://www.ncbi.nlm.nih.gov/pmc/`. Accessed: 11th April 2024.

Reid, K. (2022, 11). 16 A friendly accessible description of the ‘l-test’ – measuring (dis)information in incomplete incident reporting. In *Faculty of Clinial Informatics meeting June 2022 proceedings*, Volume 29, pp. A10.1–A10.

Reid, K. and O. Price (2022, August). PROD-ALERT: Psychiatric restraint open data-analysis using logarithmic estimates on reporting trends. *Front Digit Health 4*, 945635.

Reid, K. and O. Price (2024, May). PROD-ALERT 2: replicating and extending psychiatric restraint open data analysis using logarithmic estimates of reporting trends provisionally accepted. *Frontiers in Psychiatry*.

Reid, K. S. (2023, Apr). A362242 triangle read by rows: T(n,k) is the number of lattice paths from (0,0) to (k,n-k) using steps (i,j) with i,j gte 0 and gcd(i,j)=1. `https://oeis.org/A362242`. vanity name: McMeekin Hill.

Ross, S. M. (2018, September). *First course in probability, a books a la carte edition* (10th ed.). Pearson.

Stanley, R. P. (2011, December). *Cambridge studies in advanced mathematics enumerative combinatorics: Series number 49: Volume 1* (2nd ed.). Cambridge, England: Cambridge University Press.