Analysis of Optimal Die Configurations

Mary Kait Heeren, Dr. Jeremy Becnel
Department of Computer Science, Stephen F. Austin State University

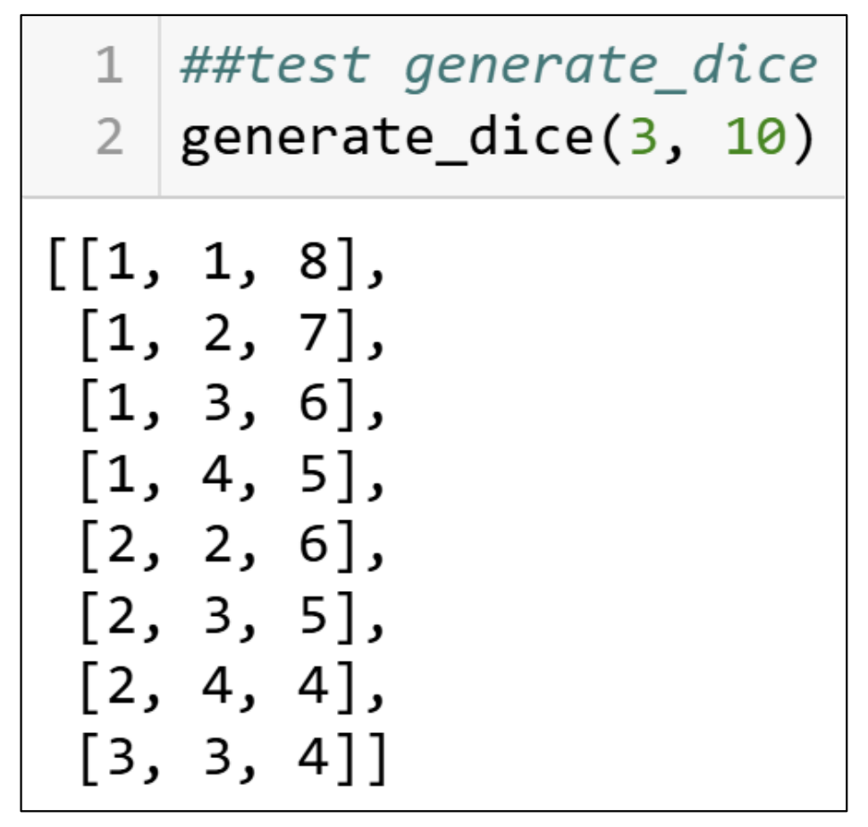
Abstract

This exploration aims to analyze the mathematical intricacies involved in constructing dice with n faces and k pips, with a focus on combinatorial analysis. The initial motivation stemmed from developing a Python program to generate and compare dice configurations. This program is a practical tool for visualizing die combinations, leading to a deeper investigation into related mathematical concepts.

The investigation uses an algorithm to construct all possible dice and compares the die to determine the optimal configuration, which is the configuration with the highest probability of winning. Additionally, we explore properties of the die including the concept of transitivity in dice relationships.

Program Implementation

The **generate_dice** method creates every possible arrangement of pips on a die with a specific number of faces and a total number of pips..



```
##test generate_dice
generate_dice(3, 10)
```

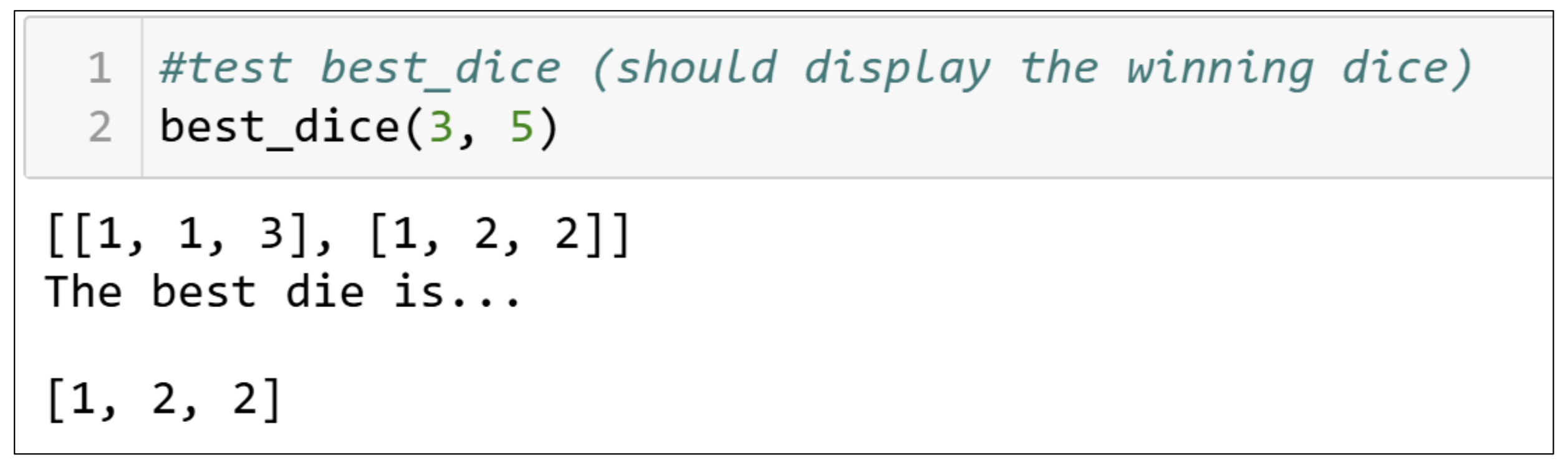
```
[[1, 1, 8],
 [1, 2, 7],
 [1, 3, 6],
 [1, 4, 5],
 [2, 2, 6],
 [2, 3, 5],
 [2, 4, 4],
 [3, 3, 4]]
```

Definition. Given two dice ad with the same number of sides, we say a die A is said to **beat** or another die B if given every pairing of sides (a,b) where a is said from A and B is a side of A, the majority of the pairing have a > b.

Example: Give your example to the right here:

The **best_dice** method takes all the dice configurations created in the above method and compares each pair to see which one performs better. It does this by checking how often one die wins against another in a series of tests. For each pair, it counts the number of wins and ties.

After comparing all the dice, the method looks for the die that has the most wins and the fewest ties. This die is considered the best one based on these comparisons, and the method then selects and shows this top-performing die.



```
#test best_dice (should display the winning dice)
best_dice(3, 5)
```

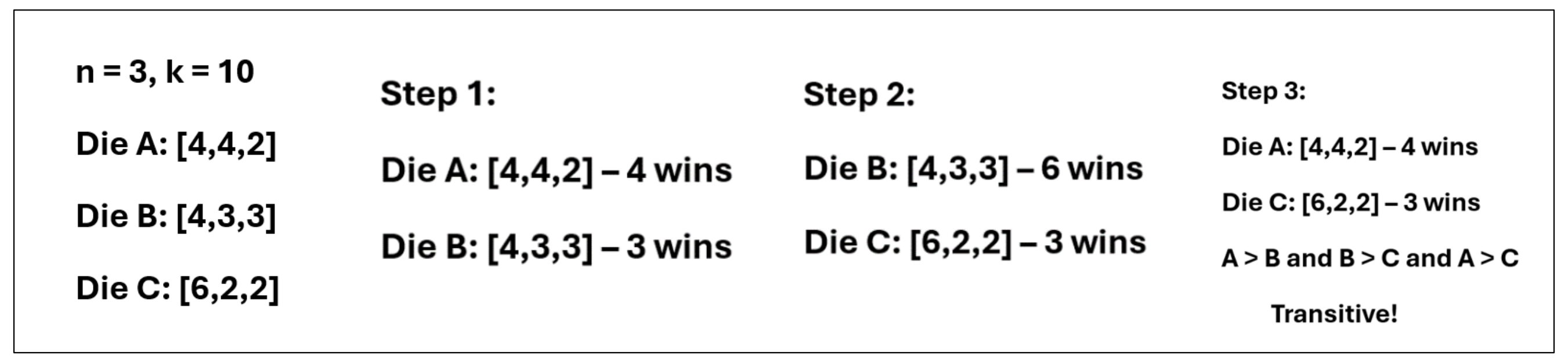
```
[[1, 1, 3], [1, 2, 2]]
The best die is...

[1, 2, 2]
```

Transitivity

A relation is transitive whenever one element is related to a second element, and that second element is related to a third element, then the first element is also related to the third element.

In terms of dice:

- If Die A beats Die B, and Die B beats Die C, then for transitivity to hold, Die A should also beat Die C.



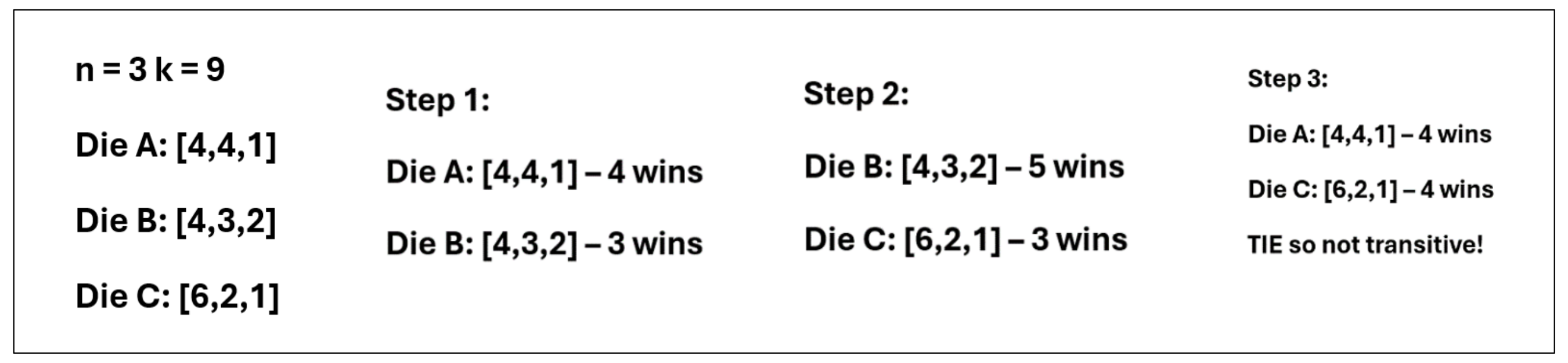
n = 3, k = 10
Die A: [4,4,2]
Die B: [4,3,3]
Die C: [6,2,2]

Step 1:
Die A: [4,4,2] – 4 wins
Die B: [4,3,3] – 3 wins

Step 2:
Die B: [4,3,3] – 6 wins
Die C: [6,2,2] – 3 wins

Step 3:
Die A: [4,4,2] – 4 wins
Die C: [6,2,2] – 3 wins
A > B and B > C and A > C
Transitive!

- As shown above, there are a few scenarios where transitivity remains true.



n = 3 k = 9
Die A: [4,4,1]
Die B: [4,3,2]
Die C: [6,2,1]

Step 1:
Die A: [4,4,1] – 4 wins
Die B: [4,3,2] – 3 wins

Step 2:
Die B: [4,3,2] – 5 wins
Die C: [6,2,1] – 3 wins

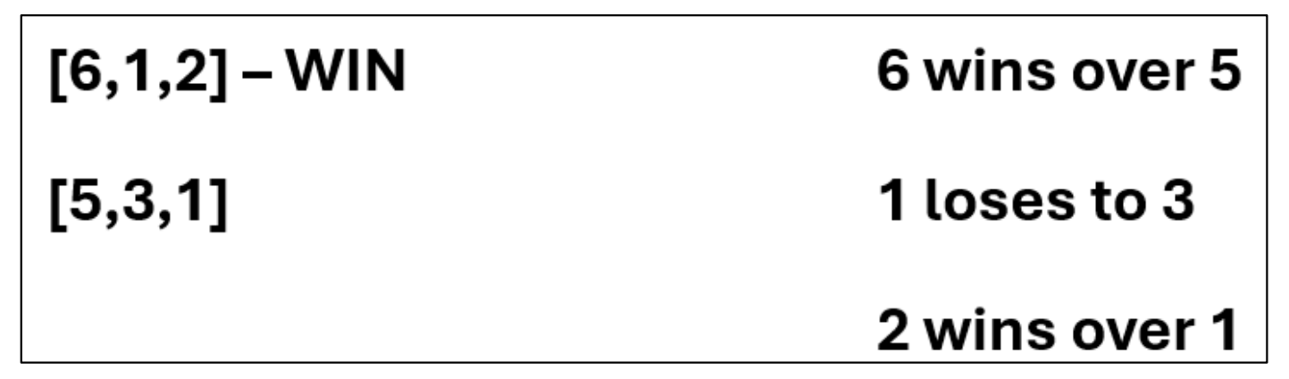
Step 3:
Die A: [4,4,1] – 4 wins
Die C: [6,2,1] – 4 wins
TIE so not transitive!

- Above, is an example where transitivity fails, thus disproving the theory that dice follow transitivity rules for every case.

More Dice Comparisons

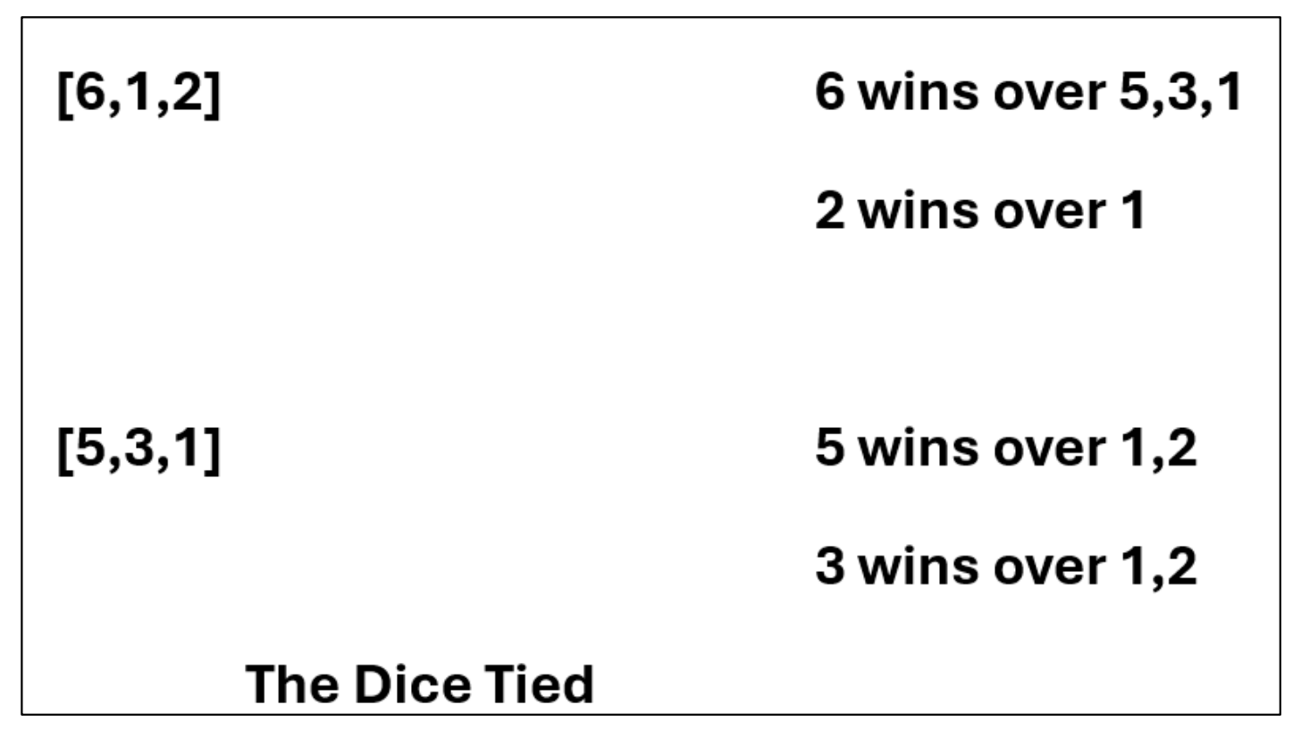
While exploring the transitive properties of dice, I discovered that there were multiple options when comparing dice that cause the results of transitivity to vary. An example of this would be comparing 1 face of the die to 1 face of the other die. This is referred to as the "overall strength" of the die.

Overall Strength Method:



[6,1,2] – WIN   6 wins over 5
[5,3,1]   1 loses to 3
2 wins over 1

Our Dice Comparison Method:



[6,1,2]   6 wins over 5,3,1
2 wins over 1

[5,3,1]   5 wins over 1,2
3 wins over 1,2

The Dice Tied

As shown in the two examples, the same two dice are being compared using two different methods to show that they will provide different comparison results. Upon further analyzation, it can be concluded that neither method shows transitivity amongst all cases.

Bells Triangle Theorem

The Bell Triangle, also known as the Aitken Triangle, is a triangular array of numbers that generates Bell numbers, which count the number of ways to partition a set. The Bell numbers are found in the leftmost diagonal of the triangle. Using the concept of Bell's Triangle, we derived a recursive formula that determines the number of possible die combinations.

$$C(n,k) = \begin{cases} 0 & \text{if } k < n \\ 1 & \text{if } k = n \\ C(n,k-1) + C(n-1,k-1) & \text{if } k > n \end{cases}$$

As an example, lets solve n = 3 and k = 4.

$$C(n,k) = C(n, k-1) + C(n-1, k-1)$$
$$C(3,4) = C(3,3) + C(2,3)$$
$$C(2,3) = C(2,2) + C(1,2)$$
$$C(1,2) = C(1,1) + C(0,1)$$
$$C(0,1) = 0$$

Since C(3,3), C(2,2), and C(1,1) each represent one combination , there are 3 total combinations. We know to stop at C(0,1) because it results in 0 combinations added. To double check, we can write out each combination for n = 3 and k = 4 to verify the correct number of 3 combinations.

[1,1,2], [1,2,1], [2,1,1]

Although the Bell Triangle and the recursive function for dice combinations are not the exact same, they share a common combinatorial theme. Both involve the addition of previous elements in a structured way to build a larger combinatorial object. The Bell Triangle is specifically used for partitioning sets, but the recursive thinking it embodies is relevant to many areas of combinatorics, including the problem of counting dice configurations.

Conclusion

We have developed algorithms to generate and compare dice configurations. We observed that transitivity does not universally hold amongst dice comparisons. We also noted that there are situations **with no maximal or best die.** We utilized the principle of the Bell Triangle to explore and quantify the number of possible dice configurations.

Overall, this research highlights the complexity of dice analysis, which has applications in programming and combinatorics. These results suggest dice's unconventional nature and open the door for further investigation into their behavior and application in different environments.

Contact

Mary Kait Heeren
Department of Computer Science
P.O. Box 9837, SFA Station
Nacogdoches, Texas 75962
heerenmk@jacks.sfasu.edu

Acknowledgements

This research was supported by the College of Sciences and Mathematics as part of the Undergraduate Research Experience at Stephen F. Austin State University. I'd like to extend a thank you to Stephen F. Austin State University and Jeremy Becnel for providing an opportunity to get involved in a Research Experience, as it has been an insightful and rewarding process.
Thank you to the Computer Science department for the workspace and the equipment used throughout the entirety of the project.

References

1. Geeks for Geeks (Aitken's Array or Bell Triangle - GeeksforGeeks)
2. Jupyter Notebook (Project Jupyter | Home)
3. Geeks for Geeks (Transitive Relations: Definition, Properties, and Examples (geeksforgeeks.org))