Comparing Snowflakes

F. Abiad

1 Introduction

The aim of this code is to compare a large number of snowflakes and see which snowflakes are equal. Each snowflake has 6 legs, as shown in Figure 1.

Figure 1: Profile of a snowflake

Traditional Solution

The traditional comparison would involve taking a snowflake (sf), comparing it through the entire set, then taking the following one, comparing it through the remaining set, and so on. This would involve the following computations

$$\begin{aligned} &N_{sf_1} \in \mathcal{O}(N-1) \\ &N_{sf_2} \in \mathcal{O}(N-2) \\ &\dots \\ &N_{sf_{N-1}} \in \mathcal{O}(1) \\ &N_{sf_N} \in 0 \end{aligned}$$

Then the number of operations to handle the comparison is

$$\begin{aligned} &(N-1)+(N-2)+(N-3)+\ldots+1+0 \\ &(N-1)N-(0+1+2+\ldots+N-1) \\ &(N-1)N-\sum_{i=0}^{N-1} i = (N-1)N-\left[\frac{N}{2}(0+N-1)\right] \\ &(N-1)N-\frac{N}{2}(N-1)=\frac{N}{2}(N-1) \end{aligned}$$

This is of the order $\mathcal{O}(N^2)$.

The proposed solution involves clustering the snowflakes into groups of equal perimeters, then comparing snowflakes within these groups. This is because all identical snowflakes have the same sum of leg lengths (perimeter).

Probability

For 6 legs that can measure 1-6, there are $6^6 = 46,656$ leg configurations, with a perimeter range of [6; 36] of 31 distinct perimeters. Note that the leg lengths are equally probable, but not perimeter, which follows a binomial distribution (rather than normal, as the former is for discrete outcomes). The scenarios are as follows

$$\begin{aligned} &[1,1,1,1,1,1] \rightarrow P=6 \\ &\dots \\ &[6,6,6,6,6,6] \rightarrow P=36 \end{aligned}$$

The most probable perimeter is given as the sum of the number of legs times the expected leg length. The latter can have values of probability $P = \frac{1}{6}$. For equally probable leg lengths, the expected value is thus:

$$E(X) = \mu = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = 3.5$$

Then, the most probable perimeter is $3.5 \times 6 = 21$.

For $6^6 = 46,656$ samples, the occurrence of each perimeter is as follows:

$$\begin{gathered} N_{21} = 4332,\ N_{22},\ N_{20} = 4221, \\ N_{23},\ N_{19} = 3906,\ N_{24},\ N_{18} = 3431, \\ N_{25},\ N_{17} = 2856,\ N_{26},\ N_{16} = 2247, \\ N_{27},\ N_{15} = 1666,\ N_{28},\ N_{14} = 1161, \\ N_{29},\ N_{13} = 756,\ N_{30},\ N_{12} = 456, \\ N_{31},\ N_{11} = 252,\ N_{32},\ N_{10} = 126, \\ N_{33},\ N_{9} = 56,\ N_{34},\ N_{8} = 21, \\ N_{35},\ N_{7} = 6,\ N_{36},\ N_{6} = 1 \end{gathered}$$

So, having a snowflake of perimeter 21 has a probability of $4332/46656 = 0.093$.

The Solution

Say there are $N = 10,000$ randomly generated snowflakes, each with $N_L = 6$ legs, and each leg having an integer length of $L = 1 - 6$. Using the traditional method of comparing one instance with the entire array, this would take $\frac{N}{2}(N-1) = 50,000,000$ comparisons. Clustering snowflakes based on perimeter should reduce the number of operations as the snowflakes are not compared against the entire array.

This method deals with splitting the initial array into 'clusters' of equal perimeter, rather than dealing with the comparison method itself. Thus, the comparison in each cluster would take $\frac{N}{2}(N-1)$ comparisons. For the most populous cluster (see above occurrences), this would be for a perimeter of 21, having a probability of $4332/46,656 = 0.093$. So for 10,000 samples, this would be a cluster of $10,000 \cdot 0.093 = 928$ snowflakes, requiring, $\frac{N}{2}(N-1) = 430,590$ comparisons. Repeating across all clusters then summing, gives the total number of operations required to be $3.3 \cdot 10^6$, which is a fraction of the steps it would take using the traditional way. This discrepancy increases quadratically with sample size; for $N = 1,000,000$ snowflakes, the traditional method would require $5 \cdot 10^{11}$ comparisons, but with the clusters method it would require $3.3 \cdot 10^{10}$ comparisons.

2 The Code

The Setup

Note that all values shown here, and in the code, are randomly generated.

Given a sample size number, snowflakes are randomly generated and stored in an array

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \dots \end{bmatrix} = \begin{bmatrix} 1 & 5 & 6 & 4 & 5 & 1 \\ 2 & 4 & 3 & 4 & 2 & 6 \\ 3 & 1 & 5 & 4 & 1 & 6 \\ & & \dots & & & \end{bmatrix}$$

The total length of legs for a snowflake is referred to as a 'perimeter'.

$$P = \begin{bmatrix} 22 \\ 21 \\ 20 \\ \dots \end{bmatrix}$$

From this, an array listing the most common perimeter and perimeter frequency is made. It is listed from most to least common perimeter value.

$$P_{freq} = (\text{most frequent perimeter}, \text{frequency}) = \begin{bmatrix} (22,4) \\ (21,3) \\ (20,2) \\ \dots \end{bmatrix}$$

An array of the indices of items having the same perimeter is made. The rows here are consistent with the frequency value of the previously shown array

$$P_{idx} = (\text{frequency}, \text{from most to least frequent}) = \begin{bmatrix} 9,6,4,1 \\ 7,5,2 \\ 3 \\ 8 \end{bmatrix}$$

Then an array with perimeter, index, and leg lengths, sorted by perimeter frequency, is made

$$\begin{bmatrix} P_3,\ idx,\ l_1,\ l_2,\ l_3,\ l_4,\ l_5,\ l_6 \\ P_5,\ idx,\ l_1,\ l_2,\ l_3,\ l_4,\ l_5,\ l_6 \\ P_6,\ idx,\ l_1,\ l_2,\ l_3,\ l_4,\ l_5,\ l_6 \\ \dots \end{bmatrix}$$

Based on the last array, an array that notifies when the perimeter changed (1 is when it changed, 0 is when it did not change) when going down, is made.

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \dots \end{bmatrix}$$

Say there are N snowflakes, so the snowflakes array will be N-long, and the change array will be $N-1$ long. Logically, it does not make sense to have the top value a 1, because the topmost perimeter is the most common, so there is a very small chance it changed. However, it was set to 1 for computational purposes. By that logic, the last elements in the array (which are there because they are the least common) should almost always be a 0.

3 The Comparison

First, it is important to understand that snowflakes are identical if they have the same leg lengths. This is regardless of direction, so $[1,2,3,4,5,6] \equiv [6,5,4,3,2,1]$ point to identical snowflakes. This also goes for wraparound arrays, so $[1,2,3,4,5,6] \equiv [3,4,5,6,1,2]$. Combining the two previous points also holds true, it is possible to compare snowflakes using backward wraparound arrays; so $[1,2,3,4,5,6] \equiv [4,3,2,1,6,5]$.

The key point is that all identical snowflakes have the same sum of leg lengths (perimeter). The opposite of this is not necessarily true; if snowflakes have the same perimeter, then it is not imperative that they are identical. The reverse applies; if snowflakes do not have equal perimeters, then they cannot be identical.

To summarize, the snowflakes are compared using the following ways:

1. Forward: $[1,2,3,4,5,6] = [1,2,3,4,5,6]$
2. Forward wraparound: $[1,2,3,4,5,6] \equiv [5,6,1,2,3,4]$
3. Backward: $[1,2,3,4,5,6] \equiv [6,5,4,3,2,1]$
4. Backward wraparound: $[1,2,3,4,5,6] \equiv [4,3,2,1,6,5]$

Two arrays, each consisting of snowflake leg lengths, are compared at once. To account for the wraparound effect, the first array is repeated once, then the code checks if the second is in it; so $a_1 = [1,2,3,4,5,6]$ becomes $a_{1,\ rep} = [1,2,3,4,5,6,1,2,3,4,5,6]$ (the *repeated array*), then the code asks if $a_2 = [2,1,6,5,4,3]$ (the *compared array*) is in the repeated array in the forward direction. It is not, so the code flips a_2 then asks if it is in a_1, which it is.

How does the code check if one array is in another? First, it takes the first element of the compared array and asks where in the repeated array is this element found. Second, it takes the second element of the compared array, and also asks where in the repeated array is this element found, *then subtracts 1 from the solution array.* This latter step is repeated, with the subtraction incremented every step, until all the compared array elements are used.

Example 1: forward wraparound

Taking $a_1 = [1,3,5,6,2,2] \rightarrow a_{1,rep} = [1,3,5,6,2,2,1,3,5,6,2,2]$ and $a_2 = [5,6,2,2,1,3]$, the comparison is as follows:

- Element $a_2[0] = 5$ is found in $a_{1,rep}[2,8]$, then subtract 0 from the array to get $[2,8]$
- Element $a_2[1] = 6$ is found in $a_{1,rep}[3,9]$, then subtract 1 from the array to get $[2,8]$

  ...
- Element $a_2[5] = 3$ is found in $a_{1,rep}[1,7]$, then subtract 5 from the array to get $[8,2]$

Then, the code looks at all answers and sees which elements are common in all. In this case, it is $[2,8]$, which is a non-empty array. Therefore, this tells the code that both snowflakes a_1 and a_2 are identical.

Example 2: backward wraparound

Taking $a_1 = [1,3,5,6,2,2] \rightarrow a_{1,rep} = [1,3,5,6,2,2,1,3,5,6,2,2]$ and $a_2 = [2,6,5,3,1,2]$

- $a_2[0] = 2$ in $a_{1,rep} \rightarrow [4,5,10,11] - 0 = [4,5,10,11]$
- $a_2[1] = 6$ in $a_{1,rep} \rightarrow [3,9] - 1 = [2,8]$

Before going any further, this already looks like there is no match, so flip the array to get $a_{2,flip} = [2,1,3,5,6,2]$, and try again.

- $a_{1,rep}\{a_{2,flip}[0] = 2\} \rightarrow [4,5,10,11] - 0 = [4,5,10,11]$

- $a_{1,rep}\{a_{2,flip}[1] = 1\} \rightarrow [0, 6] - 1 = [11, 5]$
- $a_{1,rep}\{a_{2,flip}[2] = 3\} \rightarrow [1, 7] - 2 = [11, 5]$
- $a_{1,rep}\{a_{2,flip}[3] = 5\} \rightarrow [2, 8] - 3 = [11, 5]$
- $a_{1,rep}\{a_{2,flip}[4] = 6\} \rightarrow [3, 9] - 4 = [11, 5]$
- $a_{1,rep}\{a_{2,flip}[5] = 2\} \rightarrow [4, 5, 10, 11] - 5 = [11, 0, 5, 6]$

The common elements in all the above steps are $[5, 11]$, a non-empty array. Therefore, despite one of them being flipped, both snowflakes a_1 and a_2 are identical.

Example 3: non-identical

Taking $a_1 = [1, 3, 5, 6, 2, 2] \rightarrow a_{1,rep} = [1, 3, 5, 6, 2, 2, 1, 3, 5, 6, 2, 2]$ and $a_2 = [2, 2, 4, 3, 5, 6]$

- $a_{1,rep}\{a_2[0] = 2\} \rightarrow [4, 5, 10, 11] - 0 = [4, 5, 10, 11]$
- $a_{1,rep}\{a_2[1] = 2\} \rightarrow [4, 5, 10, 11] - 1 = [3, 4, 9, 10]$
- $a_{1,rep}\{a_2[2] = 4\} \rightarrow [\,] - 2 = [\,]$

Before going further, there is no common element in the last step, so the end result will not have a common component across all steps, so both snowflakes a_1 and a_2 are different.

Back to the perimeter change array (edited to add more details). The snowflakes array (with perimeter and index) below shows the clusters, whose start is marked in red.

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ \dots \end{bmatrix} \rightarrow \begin{bmatrix} P_3,\ 6,\ l_1 \dots l_6 \\ P_3,\ 12,\ l_1 \dots l_6 \\ P_3,\ 28,\ l_1 \dots l_6 \\ P_3,\ 33,\ l_1 \dots l_6 \\ P_8,\ 20,\ l_1 \dots l_6 \\ P_8,\ 35,\ l_1 \dots l_6 \\ P_2,\ 4,\ l_1 \dots l_6 \\ \dots \end{bmatrix}$$

So the first cluster is

$$\begin{bmatrix} P_3,\ 6,\ l_1 \dots l_6 \\ P_3,\ 12,\ l_1 \dots l_6 \\ P_3,\ 28,\ l_1 \dots l_6 \\ P_3,\ 53,\ l_1 \dots l_6 \end{bmatrix}$$

The array comparison is done by looping through the main snowflake matrix, but segmented in clusters, as follows

$$[l_1 \dots l_6]_{P_3,6} \stackrel{?}{=} [l_1 \dots l_6]_{P_3,12}$$
$$[l_1 \dots l_6]_{P_3,6} \stackrel{?}{=} [l_1 \dots l_6]_{P_3,28}$$
$$[l_1 \dots l_6]_{P_3,6} \stackrel{?}{=} [l_1 \dots l_6]_{P_3,53}$$

When snowflakes are equal according to the comparison algorithm of section 3, an array keeps track of the *indices* of snowflakes that are equal, with every row representing a perimeter in which equality occurs, using -1 as a padding value. This is the final answer:

$$\begin{bmatrix} 6 & 28 & 53 & -1 & -1 \\ 11 & 27 & 64 & 187 & 206 \\ 21 & 238 & 341 & -1 & -1 \\ 43 & 296 & 371 & 421 & -1 \end{bmatrix}$$

This shows that the first perimeter, of unshown value (retrieved by inputting the index value into the snowflake array), has 3 equal snowflakes (of indices 6, 28, 53), the second row has 5, the third has 3, and the fourth has 4 equal snowflakes. Note that no row can have only one valid entry, as this implies there is only one snowflake that is equal to itself.