Integer Partition Bijection Illustrator

Kris Torres

Friday, June 13, 2014

Contents

1 Focus

The purpose of this project is to model bijections of integer partitions given from Igor Pak's paper on the said topic using Young diagrams. The focus of this project is to understand the fact that random partitions have asymptotically $O\left(\frac{\sqrt{n}}{\log n}\right)$ parts with high probability.

In order to fulfill this, I have to create and implement an `IntegerPartition` class, with its parts in decreasing order, i.e., $Z_1 \geq Z_2 \geq \ldots \geq Z_n$. My initial algorithm for generating a partition of a positive integer n is as follows:

1. Set $i = 1$.
2. Generate a part $Z_i \sim \text{DiscreteUniform}(1, n)$ and add it to the partition.
3. Set $n = n - Z_i$.
4. While $n > 0$,
   (a) Set $i = i + 1$.
   (b) Do STEP 2.
5. Sort the parts in the partition.
6. Reverse the order of the parts in the partition.

The problem with this approach is that this algorithm does not produce "geometric" integer partitions. In other words, the partition $(99, 1) \vdash 100$, which produces a lanky shape, or the partition $(25, 25, 25, 25) \vdash 100$, which produces a rectangular shape, does not resemble the usual hyperbolic limit shape. In order to achieve this property, I used the following algorithm:

1. Set $w = 0$.
2. For $i = 1$, $w < n$, and $i = i + 1$,
   (a) Generate a random number $U_i \sim \text{Uniform}(0, 1)$.
   (b) If $U_i > 0$,
      i. Set $x = e^{-c/\sqrt{n}}$, where c is the square-root of the value of the Riemann zeta function evaluated at $z = 2$. Thus,

$$c = \sqrt{\zeta(2)} = \sqrt{\sum_{k=1}^{\infty} k^{-2}} = \sqrt{\frac{\pi^2}{6}} = \frac{\pi}{\sqrt{6}}$$

      .
      ii. Set $\lambda = 1 - x^i$.
      iii. Set $m = \lfloor -\frac{\log U_i}{\lambda} \rfloor$.

iv. For $j = 0$, $j < m$, and $j = j + 1$,
   A. Add i to the partition.
   B. Set $w = w + i$.

3. Reverse the order of the parts in the partition.

The only setback to this situation is that the integer partition's weight may be greater than the expected target n. In this case, we can clear the integer partition and resample over and over again until the partition's weight is equal to n. This system is known as "waiting-to-get-lucky."

2 Code

2.1 The `IntegerPartition` Class

In order to properly sample an integer partition, the client needs to declare an `IntegerPartition` object, and call either the `randomize` or `randomizeExactly` methods on it with the expected target n passed in as an argument. If the client uses the `randomize` method, then the partition's `weight` will be at least n. If the client uses the `randomizeExactly` method instead, then the partition's `weight` will be equal to n using the "waiting-to-get-lucky" strategy by internally calling the `randomize` function over and over again until achieving success.

Since the integer partition has its parts in decreasing order, its `largestPart` and `smallestPart` have indices `0` and `numberOfParts - 1`, respectively. With a similar approach, any k-th `part` can be accessed. The client can also `print` the integer partition to the console, a *.txt* file, etc.

2.2 The `Table` Class

The purpose of the `Table` class is to serve as the internal workings of the Young diagram. A `Table` object is essentially a two-dimensional vector of colored cells, or to put it more formally, a vector of (row) vectors of colored cells. The `Table` class comes with eight fundamental methods with respect to $\mathbb{Z}^2$:

- The `cut` transformation takes a `Table` $T \subset \mathbb{Z}^2$ and returns a pair of `Tables` $T_1, T_2 \subset \mathbb{Z}^2$, where $T_1 = \{(i, j) \in \mathbb{Z}^2 \mid ai + j \geq c\}$ and $T_2 = \{(i, j) \in \mathbb{Z}^2 \mid ai + j < c\}$.
- The `moveDown` transformation takes a `Table` $T \subset \mathbb{Z}^2$ translates it along the vector $[-1 \;\; 0]^T$.
- Likewise, the `moveLeft` transformation takes a `Table` $T \subset \mathbb{Z}^2$ translates it along the vector $[0 \;\; -1]^T$.
- The `paste` transformation takes a `Table` $T_1 \subset \mathbb{Z}^2$ and overlaps it with another `Table` $T_2 \subset \mathbb{Z}^2$.

- The `shift` transformation takes a `Table` $T \subset \mathbb{Z}^2$ and overlaps and translates each i-th row along the vector $[0 \quad i+1]^T$.
- The `shred` transformation takes a `Table` $T \subset \mathbb{Z}^2$ and returns a pair of `Table`s $T_1, T_2 \subset \mathbb{Z}^2$ where $T_1 = \{(i,j) \in \mathbb{Z}^2 \mid i \text{ even}\}$ and $T_2 = \{(i,j) \in \mathbb{Z}^2 \mid i \text{ odd}\}$.
- The `stretchDown` transformation takes a `Table` $T \subset \mathbb{Z}^2$ and translates each i-th row along the vector $[0 \quad -2i]^T$.
- The `stretchRight` transformation takes a `Table` $T \subset \mathbb{Z}^2$ and translates each j-th column along the vector $[2j \quad 0]^T$.

In addition, the client can also compute the sum and union of multiple `Table`s like so:

- The computation of the `sum` involves taking the columns of all the `Table`s and rearranges them from largest to smallest.
- Likewise, the computation of the `union` involves taking the rows of all `Table`s and rearranges them from largest to smallest.

There are some subtleties within the `Table` class. The cells in a `Table` can also be uncolored or transparent. The transformation of a `Table` input may create a rectangular template for the `Table` output using uncolored cells. Another way to achieve a similar result is to call the `makeRectangular` method on the `Table` input. At the end of each transformation, the `trim` method is internally called, removing any uncolored cells from the `Table` output. If this method was not called, then any immediately succeeding transformations will be way off.

2.3 The `JYoungDiagram` Class

The only functionality of a `JYoungDiagram` object is to display its inner `Table` object using raised-beveled colored panels to represent the cells, ignoring any transparent cells in the process.

3 Application

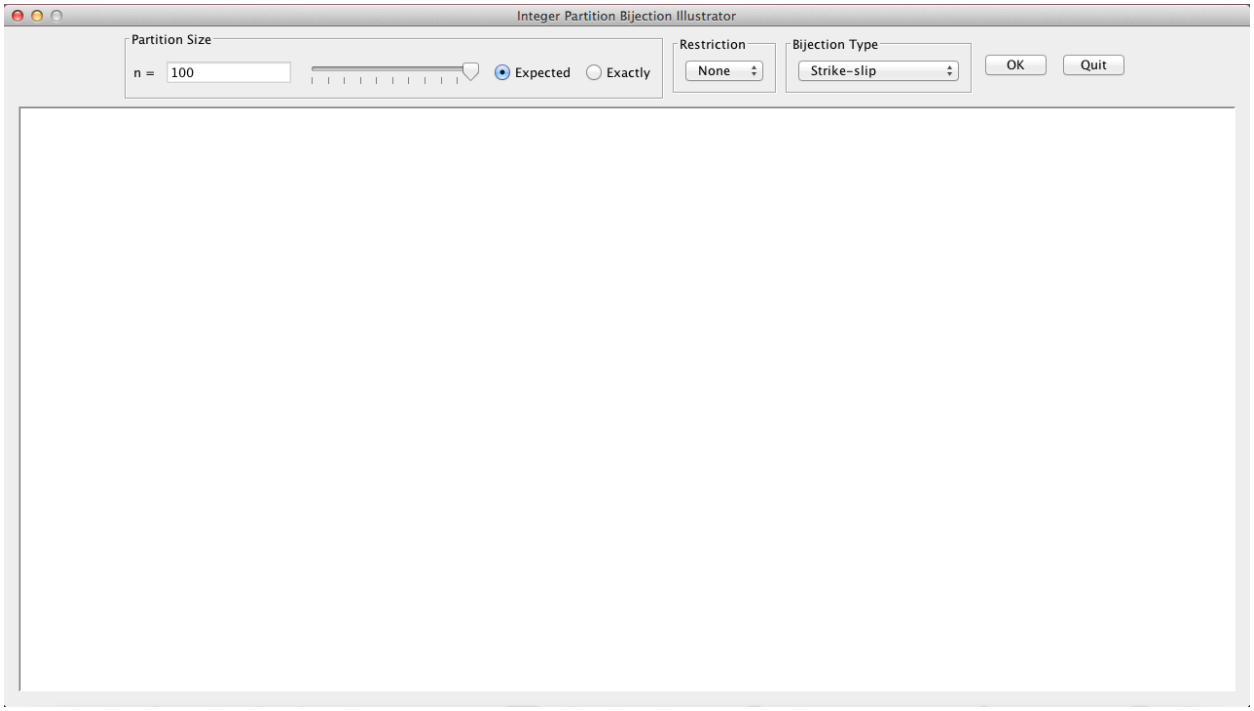

Figure 1: The GUI with an empty display area.

The GUI (graphical user interface) for the application contains a relatively large rectangular area for displaying a bijection of an integer partition. When the program starts running, the display area is initially empty. Above the display area are the controls for setting the partition size (1–100), the sampling mode ("expected" or "exactly" that partition size), restrictions on the parts (no restrictions, even parts only, or odd parts only), and the bijection to display (the strike slip, shred-and-stretch, cut-and-stretch, or Glaisher's bijection).

The client should not attempt to make an empty to make a partition of *exactly* an *odd* positive integer n using *even* parts. Otherwise, instead of displaying a bijection of the "partition," this will pop up:

Figure 2: The error message.

Also, a bijection may output an integer partition with its parts not necessarily in decreasing order. Here is an example:

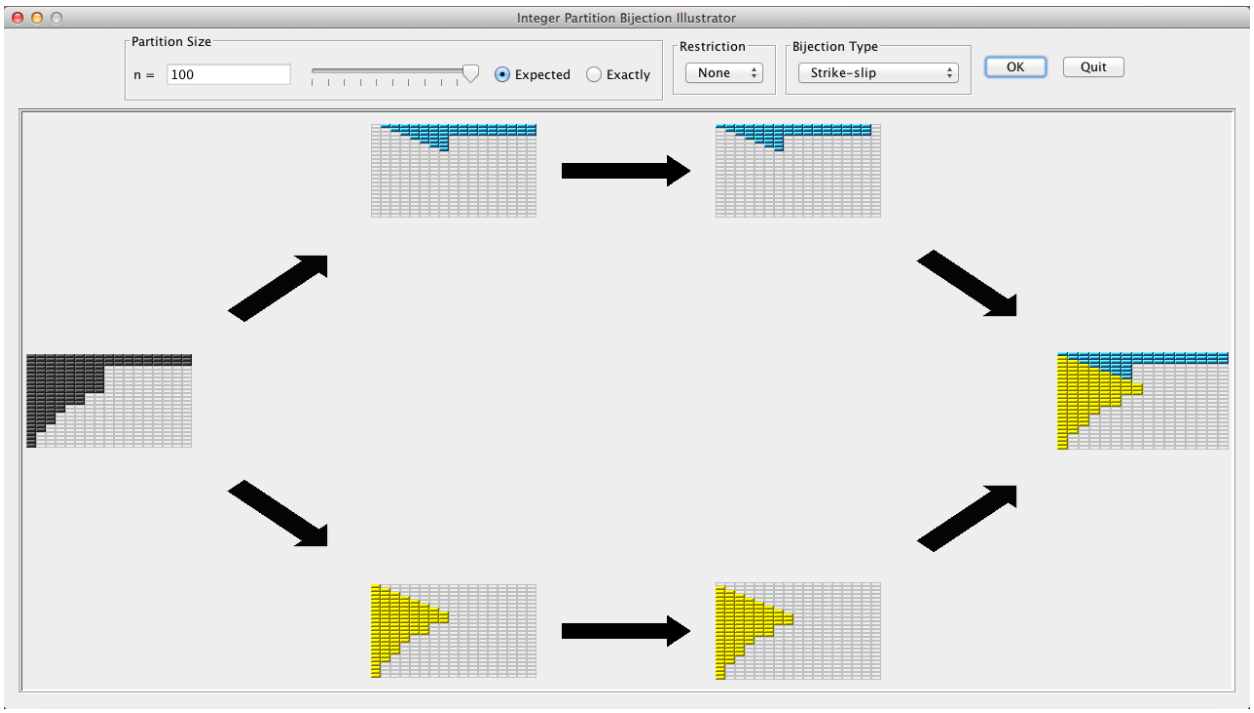

Figure 3: A strike-slip bijection that outputs an *unordered* integer partition.

Finally, to the right of the controls are the *OK* button, which displays a new bijection of a new integer partition, and the *Quit* button, which intuitively quits the program.

3.1 Strike-Slip Bijection

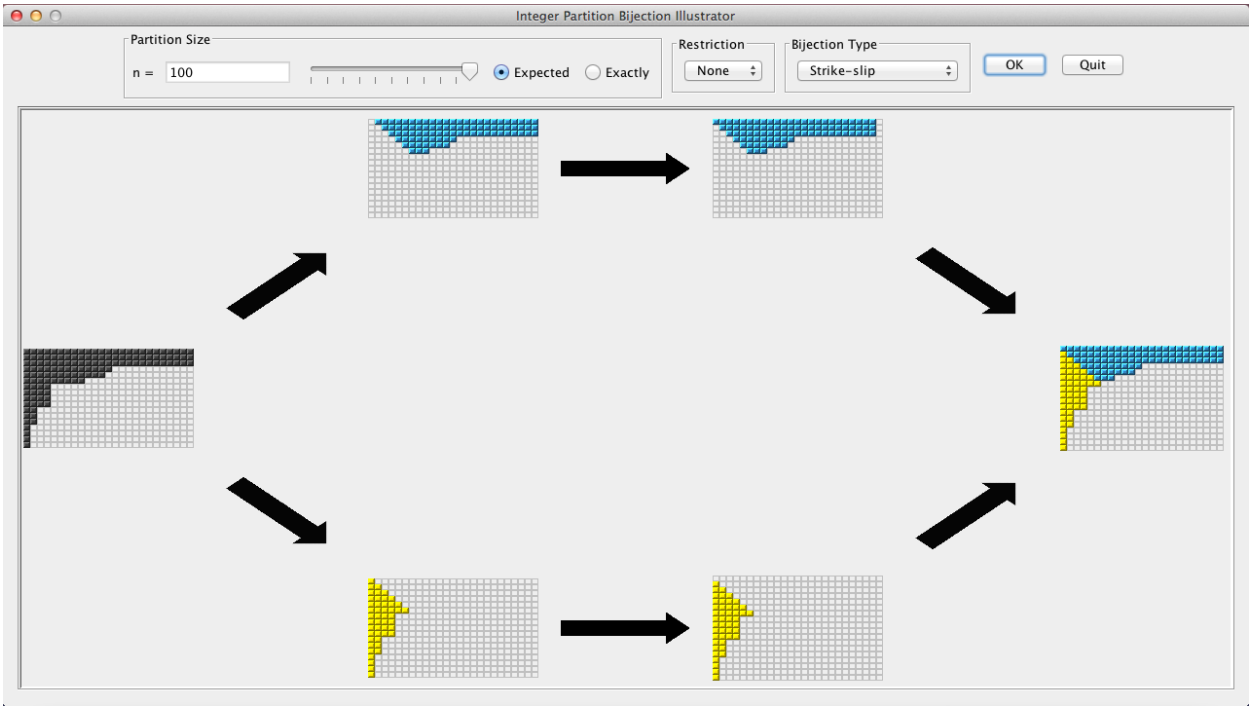

Figure 4: An example of a strike-slip bijection.

The **strike-slip bijection** $L \to R$ is defined as follows:

1. Start with a `Table` $L \subset \mathbb{Z}^2$ and `cut` it along the line $i + j \geq 0$ to obtain `Tables` $U_1, D_1 \subset \mathbb{Z}^2$.
2. Apply the `moveLeft` transformation on U_1 to obtain a `Table` $U_2 \subset \mathbb{Z}^2$.
3. Likewise, apply the `moveDown` transformation on D_1 to obtain a `Table` $D_2 \subset \mathbb{Z}^2$.
4. `paste` D_2 onto U_2 to obtain a `Table` $R \subset \mathbb{Z}^2$.

3.2 Shred-and-Stretch Bijection

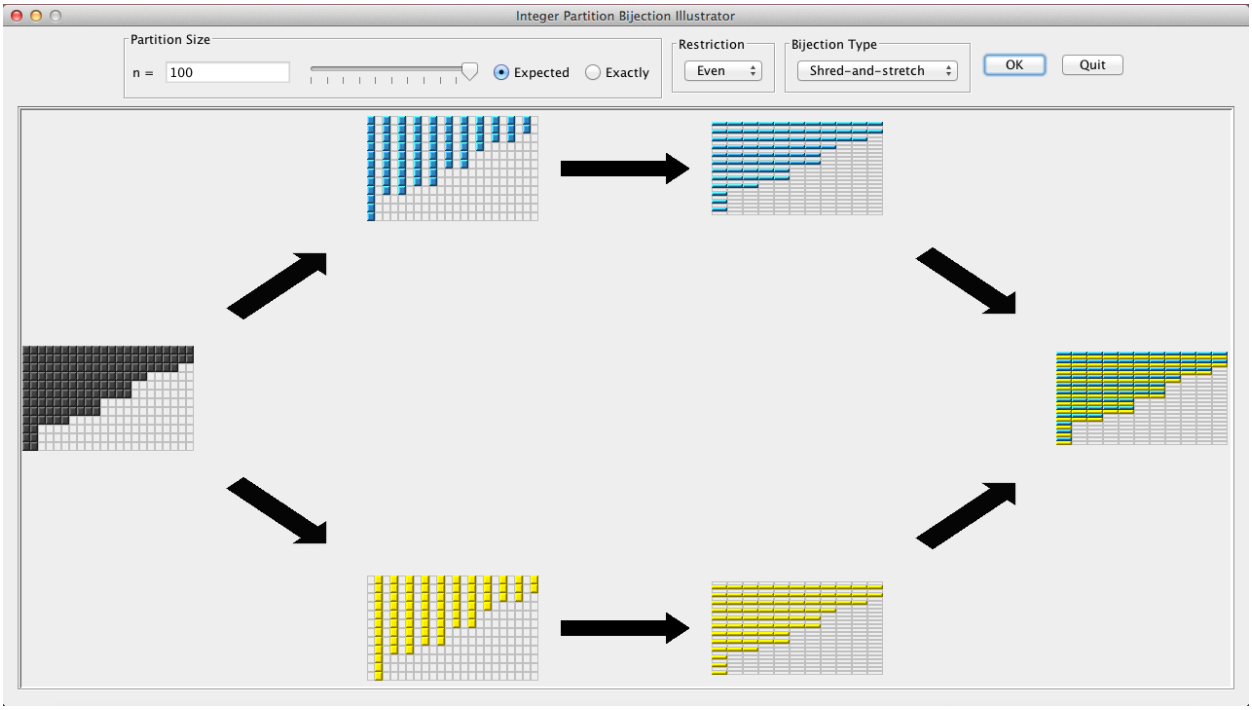

Figure 5: An example of a shred-and-stretch bijection of an integer partition with strictly even parts only.

The **shred-and-stretch bijection** $L \to R$ is defined as follows:

1. Start with a `Table` $L \subset \mathbb{Z}^2$ and `shred` it to obtain `Tables` $U_1, D_1 \subset \mathbb{Z}^2$.
2. Apply the `stretchDown` transformation on U_1 and D_1 to obtain `Tables` $U_2 \subset \mathbb{Z}^2$ and $D_2 \subset \mathbb{Z}^2$, respectively.
3. `paste` D_2 onto U_2 to obtain a `Table` $R \subset \mathbb{Z}^2$.

Note that this bijection works ideally on integer partitions with strictly even parts only.

3.3 Cut-and-Stretch Bijection

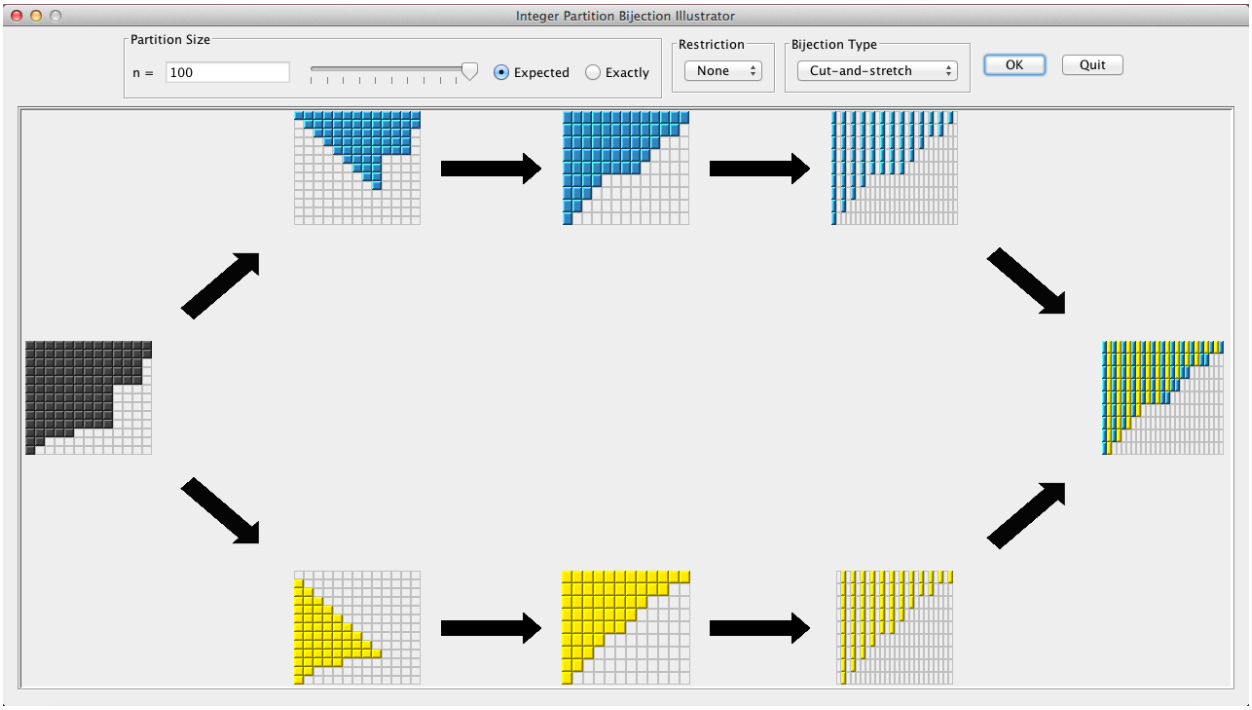

Figure 6: An example of a cut-and-stretch bijection.

The **cut-and-stretch bijection** $L \to R$ is defined as follows:

1. Start with a `Table` $L \subset \mathbb{Z}^2$ and `cut` it along the line $i + j \geq 1$ to obtain `Tables` $U_1, D_1 \subset \mathbb{Z}^2$.
2. Normalize U_1 to the left, i.e., remove any transparent cells from U_1, to obtain $U_2 \subset \mathbb{Z}^2$.
3. Normalize D_1 upwards, i.e., sort the rows of D_1 in decreasing order. Then $D_2 := D_1^T$ (take the transpose of D_1 to obtain $D_2 \subset \mathbb{Z}^2$).
4. Apply the `stretchRight` transformation on U_2 and D_2 to obtain `Tables` $U_3 \subset \mathbb{Z}^2$ and $D_3 \subset \mathbb{Z}^2$, respectively.
5. `paste` D_3 onto U_3 to obtain a `Table` $R \subset \mathbb{Z}^2$.

Note that this bijection works on any integer partition.

3.4 Glaisher's Bijection

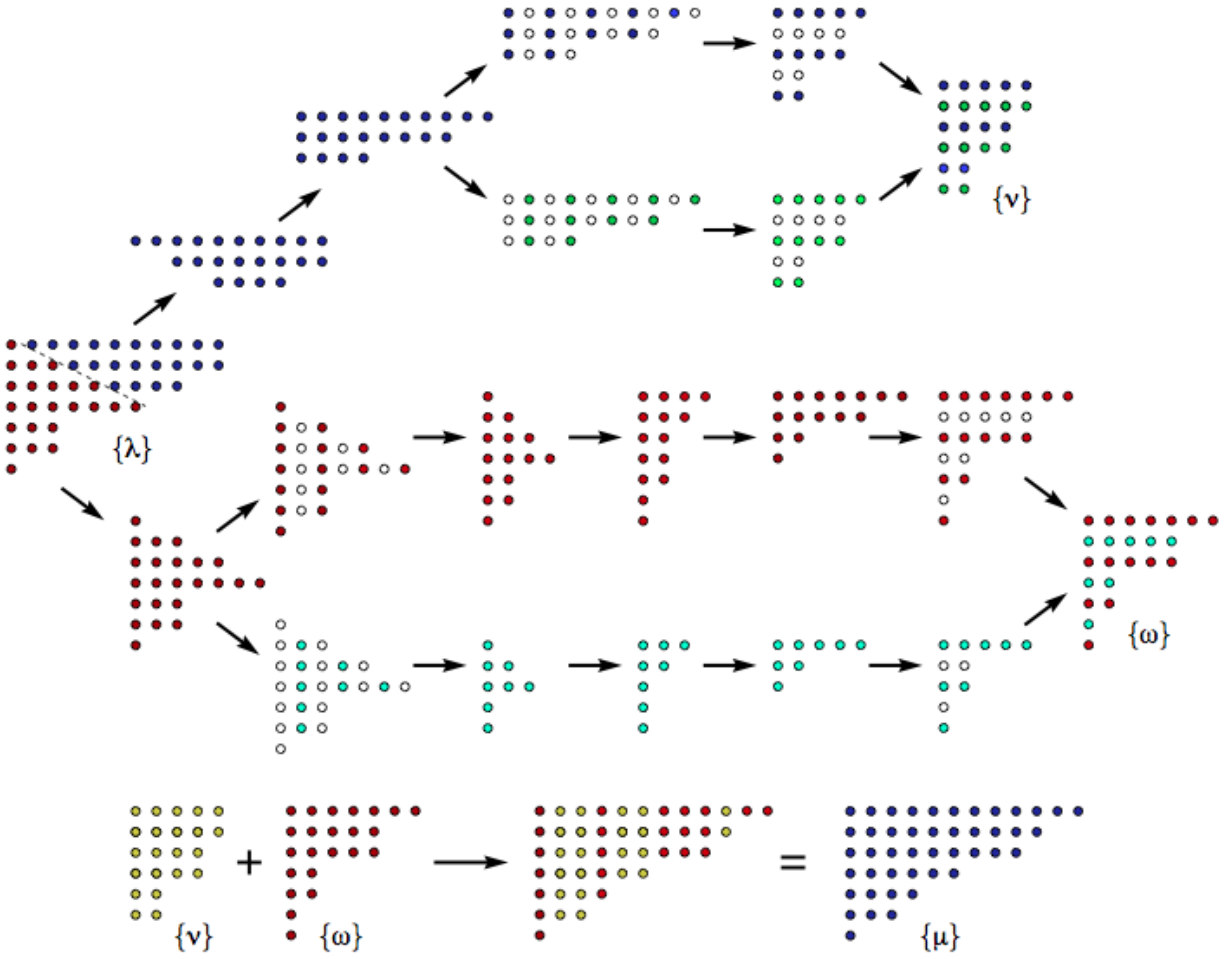

Figure 7: An example of Glaisher's bijection $\varphi : \lambda \to \mu$, where $\lambda = (11, 11, 9, 7, 3, 3, 1) \in \mathcal{L}$ and $\mu = (12, 10, 9, 6, 4, 3, 1) \in \mathcal{R}$.

Glaisher's bijection $L \to R$ is defined as follows:

1. Start with a `Table` $L \subset \mathbb{Z}^2$ and `cut` it along the line $2i + j \geq 0$ to obtain `Tables` $U_1, D_1 \subset \mathbb{Z}^2$.
2. Normalize U_1 to the left to obtain $U_2 \subset \mathbb{Z}^2$.
3. Apply the shred-and-stretch bijection on U_2 to obtain $U_3 \subset \mathbb{Z}^2$.
4. Normalize D_1 upwards. Then `shred` it to obtain `Tables` $D_{U_1}, D_{D_1} \subset \mathbb{Z}^2$.
5. Normalize D_{U_1} and D_{D_1} to the left to obtain `Tables` $D_{U_2} \subset \mathbb{Z}^2$ and $D_{D_2} \subset \mathbb{Z}^2$, respectively.

6. $D_{U_3} := D_{U_2}^T$ and $D_{D_3} := D_{D_2}^T$.

7. Apply the `stretchDown` transformation on D_{U_3} and D_{D_3} to obtain $D_{U_4} \subset \mathbb{Z}^2$ and $D_{D_4} \subset \mathbb{Z}^2$, respectively.

8. `paste` D_{D_4} onto D_{U_4} to obtain a `Table` $D_2 \subset \mathbb{Z}^2$.

9. $R := U_3 + D_2$.

Note that this bijection works on any integer partition.

4 Manifest Destiny (Future Extensions)

I plan to optimize the Integer Partition Bijection Illustrator so that it can handle larger partition sizes. One way I could approach this scenario is to use multithreading. I also plan to make an applet version of this program and embed it in a personal website some day in the future. Finally, I want to add the ability for users to create their own integer partition bijections.

5 Acknowledgements

I would like to thank Prof. Stephen DeSalvo for giving me the opportunity to work on this directed research/senior project and making the topic of integer partition bijections fun and interesting. Also, I would like to give a special thanks to Prof. Igor Pak for letting me use his paper on the aforementioned topic.

6 Works Cited

[1] Arratia, R., DeSalvo, S., Probabilistic divide-and-conquer: A new exact simulation method, with integer partitions as an example, 35 pp., available at `http://arxiv.org/pdf/1110.3856.pdf`.

[2] Pak, I., The nature of partition bijections II. Asymptotic stability, preprint (2004), 32 pp., 19 pictures, available at `http://www.math.ucla.edu/ pak/papers/stab5.pdf`.