CURO 2021 Report

Max Schneider & Dr. Weiwei Hu

April 4, 2021

1 Problem Description

What is the optimal way to scramble an image? This is the essential question that we address. Before approaching this question, however, we need to provide a more rigorous description of what an image is, and what it means for an image to be scrambled.

Let $I_{w,h}$ denote an image with width w and height h. Since images are two-dimensional, $I_{w,h}$ can be represented as a matrix with h rows and w columns. Real life image files can be stored in several different formats, but we will make the simplifying assumption that each element in $I_{w,h}$ is a real number in the interval $[0, 1]$. We also introduce the notation $I_{w,h}(x, y)$ as representing the entry at the xth column and yth row of $I_{w,h}$.

A scrambling of $I_{w,h}$ is simply a permutation of the elements in $I_{w,h}$. Therefore, there are $(wh)!$ possible scramblings of $I_{w,h}$. Since it is somewhat of a subjective question to ask how "scrambled" an image is, we need a well-defined metric that can measure how mixed an image is based on some numerical property of the image. Let M denote a function that takes an image of arbitrary size as input and outputs a real number. Then, given an image $I_{w,h}$ we can summarize the problem like so:

1. Determine a viable metric M that is lower when the image is more "mixed" and higher when the image is less "mixed."

2. Determine a permutation P such that $M(P(I_{w,h}))$ is minimized.

Note that it would be equivalent for the metric M to be higher when the image is more "mixed" as long as the algorithm for finding P is then modified in order to maximize $M(P(I_{w,h}))$.

We also stress that we specifically want a practical and efficient method for finding P; i.e. a brute force search over all $(wh)!$ permutations would not be a viable solution. Below are a few relevant criteria in our search for a solution to this problem:

- The solution must be computable in the real world. In other words, a theoretical solution to the problem that is not computable or would take hundreds of years to compute would not be acceptable.
- The solution should be a permutation of the original image. (We are a little loose on this requirement; more on this later.)
- The solution should be reversible; in other words, the original image should be recoverable from the scrambled image.

2 Methodology

2.1 Finding the right metric

We begin this section by introducing a function for measuring how scrambled an image is and justifying how we arrived at this metric. Imagine an image that is just one solid color versus an image that is a perfect black-and-white checkerboard. One would expect the former image to be the least scrambled and the latter image to be the most scrambled, and our metric should reflect this intuition. A naive way to achieve this is to compare every element in the image with the 2, 3, or 4 elements adjacent to it. Such a function metric would be computed like this:

$$M(I_{w,h}) = \sum_{x_1=1}^{w} \sum_{y_1=1}^{h} \sum_{\substack{|x_1-x_2| \\ +|y_1-y_2| \\ =1}} |I(x_1, y_1) - I(x_2, y_2)|.$$

It is easy to see that such a metric is minimized when $I_{w,h}$ is one solid color and maximized when $I_{w,h}$ is a perfect black-and-white checkerboard. However, this metric is flawed because it lacks any information about how the image looks on a global scale. For instance, an image might be skewed on a large scale towards having brighter colors on the left and darker colors on the right, but this metric would not measure this pattern since it only compares adjacent pixels. Therefore, instead of simply comparing adjacent pixels, it makes sense to compare *every* pair of pixels within the image. However, we still place greater importance on pixels that are adjacent than pixels on opposite ends of the image, so we want to weigh pairs of pixels that are closer together more heavily in the summation. This gives us the following metric:

$$M(I_{w,h}) = \sum_{x_1=1}^{w} \sum_{y_1=1}^{h} \sum_{x_2=1}^{w} \sum_{y_2=1}^{h} \frac{|I(x_1, y_1) - I(x_2, y_2)|}{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

The decision to weigh each pair of pixels by the inverse square of their distance rather than simply using the inverse was inspired by the various inverse square laws found in physics.

This is the metric we use from here on out to determine how mixed an image is. Higher values of M indicate greater mixedness and lower values indicate less mixedness.

2.2 An initial solution inspired by physics

Before continuing to our main solution, we will briefly outline an initial solution inspired by processes in physics. We can easily observe particles mixing together in the real world (imagine putting two gases together in one container or dropping food coloring into a water glass), so it is natural to try and simulate these processes in a way that can be applied to our problem.

Suppose we pretend that each pixel in $I_{w,h}$ is an independent particle, and that there exists a gravitation-like force between every pair of pixels. We want to design this force so that pixels with very similar colors are repelled from each other and pixels with very different colors are attracted to each other. The hope is that under the influence of this force, the particles within this system will naturally sort themselves into a state of optimal mixedness.

Let $P_1 = I_{w,h}(x_1, y_1)$ and $P_2 = I_{w,h}(x_2, y_2)$ for some arbitrary coordinates (x_1, y_1) and (x_2, y_2). We want a formula for the force exerted on P_1 by P_2. The force will be in the direction $D = \langle x_2 - x_1, y_2 - y_1 \rangle$. We want the force to have positive magnitude when $|P_1 - P_2|$ is large and negative magnitude when the difference is small; therefore, we can scale the force vector by $|P_1 - P_2| - \frac{1}{2}$. We also want to scale the force vector by the square of the distance between the two pixels to make it consistent with how gravitation works in the real world. This gives us the following expression for the force:

$$C\frac{|P_1 - P_2| - \frac{1}{2}}{|D|^2}\hat{D},$$

where C is a constant.

The implementation of a particle simulation from here is straightforward. Below we have included pseudocode detailing one possible implementation:

```
1: Start with initial image $I_{w,h}$
2: Allocate memory $A$, $V$, and $P$ to store the acceleration, velocity, and
   position of each particle
3: for all integer coordinates $(x, y) \in [1, w] \times [1, h]$ do
4:     $A(x, y) \leftarrow 0$
5:     $V(x, y) \leftarrow 0$
6:     $P(x, y) \leftarrow \langle x, y \rangle$
7:     Set $P(x, y)$ as unmarked
8: end for
9: Allocate $P'$ to be the same size as $P$
10: for every timestep do
11:     $P' \leftarrow P$
12:     for all integer coordinates $(x_1, y_1) \in [1, w] \times [1, h]$ do
13:         $A(x_1, y_1) \leftarrow 0$
14:         for all integer coordinates $(x_2, y_2) \in [1, w] \times [1, h]$ do
15:             $D \leftarrow P'(x_2, y_2) - P'(x_1, y_1)$
16:             $A(x_1, y_1) \leftarrow A(x_1, y_1) + C\frac{|I_{w,h}(x_1,y_1) - I_{w,h}(x_2,y_2)| - \frac{1}{2}}{|D|^2}\hat{D}$
17:         end for
18:         $V(x_1, y_1) \leftarrow V(x_1, y_1) + A(x_1, y_1)$
19:         $P(x_1, y_1) \leftarrow P'(x_1, y_1) + V(x_1, y_1)$
20:         if $P(x_1, y_1)$ has moved outside the image boundary then
21:             Reflect $V(x_1, y_1)$ about the boundary
22:             Move $P(x_1, y_1)$ back within the boundary
23:         end if
24:     end for
25: end for
26: Allocate memory $I'_{w,h}$ for the scrambled image
27: for all integer coordinates $(x_1, y_1) \in [1, w] \times [1, h]$ do
28:     $m \leftarrow w + h$
29:     Allocate coordinate pair $p$
```

```
30:     for all integer coordinates $(x_2, y_2) \in [1, w] \times [1, h]$ do
31:         if $P(x_2, y_2)$ is unmarked and $|P(x_2, y_2) - \langle x_1, y_1 \rangle| < m$ then
32:             $m \leftarrow |P(x_2, y_2) - \langle x_1, y_1 \rangle|$
33:             $p \leftarrow (x_2, y_2)$
34:         end if
35:     end for
36:     $I'_{w,h}(x_1, y_1) = I_{w,h}(p)$
37:     Set $P(p)$ as marked
38: end for
39: Return $I'_{w,h}$
```

2.3 Gradient descent applied to a transport equation

The major flaw in the above approach is that we lack a rigorous argument that M is maximized by the particle simulation algorithm. Instead, we focus from here on out on an algorithm based on gradient descent.

Although gradient descent might seem like a natural approach to this type of problem, it is not immediately obvious how it can be applied to this problem, since it is designed to search continuous sets but the set of permutations of an image is finite and discrete. We get around this issue by mapping the image onto the set $[0, 1]^2$, thus generalizing the image as a continuous function from $[0, 1]^2$ to $[0, 1]$.

There are several ways we could map an image $I_{w,h}$ onto the unit square. Let f denote the function that $I_{w,h}$ is mapped onto. The most naive mapping would be to let $f(x, y) = I_{w,h}(\lfloor 1 + x(w - 1) \rfloor, \lfloor 1 + y(h - 1) \rfloor)$, mapping (x, y) to the interval $[1, w] \times [1, h]$. This mapping is problematic for our purposes, however, because it is not continuous. Furthermore, the derivatives of f with respect to x and y are almost always zero, which can often cause our gradients to equal zero in the algorithm we introduce later. A better solution is to use bilinear interpolation. If $(x, y) = (\frac{n}{w-1}, \frac{m}{h-1})$ for some integers $(n, m) \in [0, w-1] \times [0, h-1]$, then we assign $f(x, y) = I_{w,h}(n+1, m+1)$; otherwise, we assign $(x', y') = (1 + x(w - 1), 1 + y(h - 1))$ and calculate $f(x, y)$ by linearly interpolating between $I_{w,h}(\lfloor x' \rfloor, \lfloor y' \rfloor)$, $I_{w,h}(\lfloor x' \rfloor + 1, \lfloor y' \rfloor)$, $I_{w,h}(\lfloor x' \rfloor, \lfloor y' \rfloor + 1)$, and $I_{w,h}(\lfloor x' \rfloor + 1, \lfloor y' \rfloor + 1)$ based on the fractional part of x' and y'. We will use f to refer to this mapping from here on out.

The next issue at hand is how we can generalize permuting an image in this new context. A natural way to generalize the idea of a permutation to a continuous set is using a transport equation to systematically move every point in the unit square over time. In order for the transport equation to be analogous to a permutation, however, we want the velocity function to satisfy a couple requirements:

1. The velocity function is divergence free. In other words, there is no expanding or compressing anywhere within the unit square.
2. At the boundary of the unit square, the velocity function should be parallel to the boundary line. This ensures that nothing enters or leaves the unit square.

Let $v(x, y) : [0, 1]^2 \to \mathbb{R}^2$ denote our velocity function. Then, let $\theta(x, y, t) : [0, 1]^2 \times [0, \infty) \to [0, 1]$ denote the density at the point (x, y) at time t. We define $\theta(x, y, t) = f(\langle x, y\rangle - v(x, y)t)$. All that is left is to choose a velocity function v that satisfies our requirements. It can be shown that given any parameters $a_1, ..., a_k$,

$$v(x, y) = \sum_{i=1}^{k} a_i \langle \sin(2^i \pi x) \cos(2^i \pi y), -\cos(2^i \pi x) \sin(2^i \pi y)\rangle$$

satisfies both of the aforementioned requirements.

Since the unit square changes with respect time, we need a way to discretize the density field and obtain a final w by h image. To do so, we can simply allocate a final image $I'_{w,h}$ and set $I'_{w,h}(x, y) = \theta(\frac{x-1}{w-1}, \frac{y-1}{h-1}, t)$ for each image coordinate (x, y) and some final time t.

Our task has now been translated to the following problem: find the values of t and $a_1, ..., a_k$ such that $M(I'_{w,h})$ is maximized. The following algorithm outlines how gradient descent can be applied to this problem:

```
1: Start with initial image $I_{w,h}$
2: Initialize the constant $\alpha$
3: Initialize $t$ and $a_i, ..., a_k$ to random values
4: Allocate memory to store $\Delta t$ and $\Delta a_1, ..., \Delta a_k$
5: for every timestep do
6:     Numerically approximate $\frac{\delta M}{\delta t}$ and store the value in $\Delta t$
7:     for all integers $i \in [1, k]$ do
8:         Numerically approximate $\frac{\delta M}{\delta a_i}$ and store the value in $\Delta a_i$
9:     end for
10:    $t \leftarrow t + \alpha \cdot \Delta t$
11:    for all integers $i \in [1, k]$ do
```

```
12:         $a_i \leftarrow a_i + \alpha \cdot \Delta a_i$
13:     end for
14: end for
15: Initialize $I'_{w,h}$ and calculate the values of its entries using the adjusted
    parameters
16: Return $I'_{w,h}$
```

Notice that this is technically "gradient ascent," as we aim to maximize the value of M.

2.4 Improving efficiency and reducing algorithm time complexity

The gradient descent method is a nice solution to the problem because once you have tuned all the parameters, obtaining the final scrambled image only requires a simple function evaluation. However, the process of tuning the parameters can be costly because the function M must be evaluated at each timestep, and M is unfortunately an expensive function to compute. This mainly stems from the fact that M computes a double summation, i.e. every pixel in the image is compared to every other pixel. This means that the time complexity of evaluating M is quadratic with respect to the image size. This problem quickly becomes noticeable as one tries to run gradient descent on increasingly large image files.

A workaround is to define some constant R and redefine M like so:

$$M(I_{w,h}) = \sum_{x_1=1}^{w} \sum_{y_1=1}^{h} \sum_{x_2=1}^{w} \sum_{y_2=1}^{h} \begin{cases} \frac{|I(x_1,y_1)-I(x_2,y_2)|}{(x_1-x_2)^2+(y_1-y_2)^2}, & (x_1-x_2)^2+(y_1-y_2)^2 \leq R^2 \\ 0, & \text{otherwise.} \end{cases}$$

Under this definition, M only compares each pixel to those within a radius R of the pixel, reducing the time complexity to linear with respect to the image size. However, the time complexity is now quadratic with respect to R, meaning that caution must be used when increasing R. It is also worth noting that using this method results in a loss of information about the image's mixedness on a large scale, since pixels are now only compared to other pixels within a close neighborhood.

Note that a similar technique can be used to speed up the particle simulation — instead of comparing every pair of particles, only compare pairs of particles that are less than a distance of R apart.

3 Observations and Conclusions

Following is an evaluation of the strengths and weaknesses of each method:

1. Particle simulation method:
   - Strengths:
     - Outputs a true permutation of the original image
     - Partially based on physical real-world processes
   - Weaknesses:
     - As mentioned before, we have no proof that this algorithm maximizes M.
     - This method is not reversible; we have not found a way to recover the original image from the scrambled image.
     - This method tends to run slower than the gradient descent method.
2. Gradient descent method:
   - Strengths:
     - This method is practical to compute in the real world, enough so that we were able to create demonstration videos of the algorithm in action with minimal computation time.
     - This method is somewhat reversible. Specifically, if you have some final value of t then your final density field will be described by the function $\theta(x, y, t)$. If you are given this field as the starting image, then $f'(x, y)$ gets defined as $\theta(x, y, t)$ and you should be able to obtain the original image by evaluating $\theta'(x, y, -t)$ at each relevant coordinate (x, y). However, in practice we cannot store the entire density field when the algorithm terminates (that would require infinite memory), so instead we sample it in wh places and store the result into an image. Thus, when the image is loaded back up, some information is inevitably lost, and evaluating $\theta'(x, y, -t)$ will really only yield an approximation of the original image.
   - Weaknesses:

- The final image is not necessarily a permutation of the original image since the final image can sample from non-integer coordinates of the original image. However, if a true permutation is desired then the provided algorithm can be modified in order to obtain one.
- The method is not completely reversible; one can only obtain back an approximation of the original image from the scrambled image.
- So far, we have no proof that every possible permutation of an image is achievable using the family of velocity functions we have defined. Thus, it is possible that the method only explores a subset of all permutations of the original image, and that the true global maximum is unreachable by this algorithm.