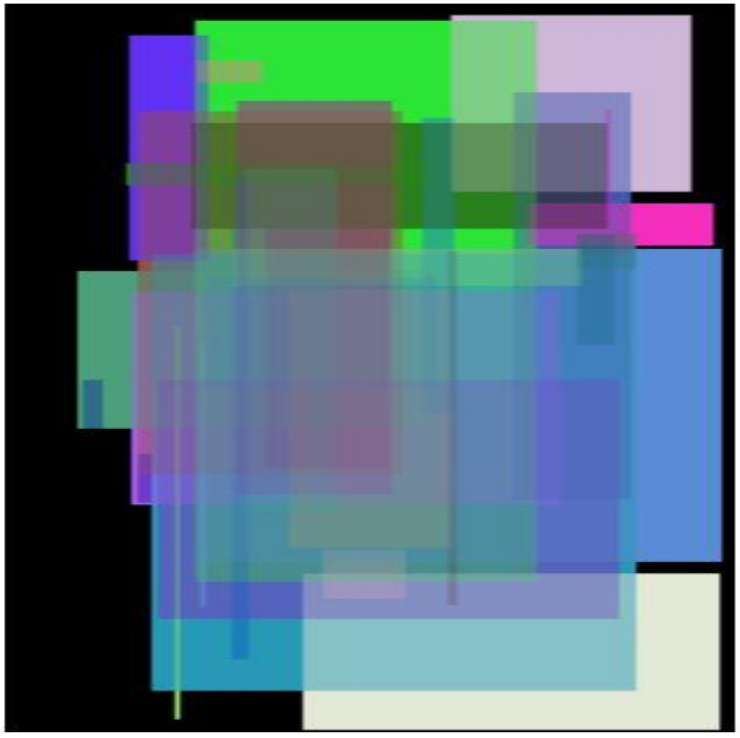
Introduction - The Patchwork Picture

Is it even possible to find an algorithm that can approximate the picture on the left using 32 colored rectangles? A random attempt is on the right.



Introduction - The Patchwork Picture

Here's another simple problem: suppose you want to make a copy of a famous photograph or painting, but you are only able to use colored rectangles. You have rectangles in every size and color. The rectangles are transparent, and if two overlap, the overlapping region will have the average of the two colors.

If that was the whole problem, we'd be done, because we make pixellated versions of images all the time.

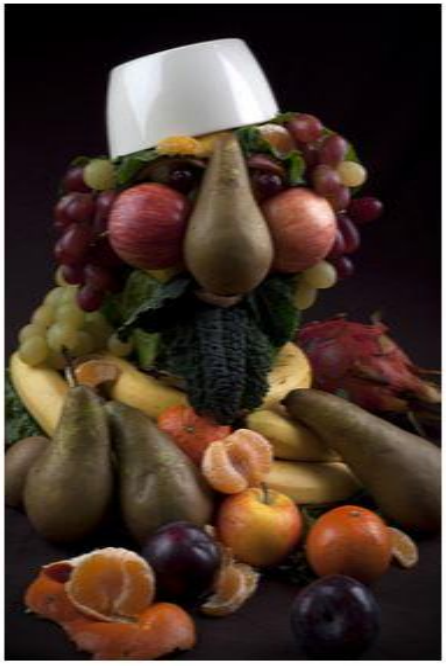
But for this problem, we are only allowed to use 32 rectangles!

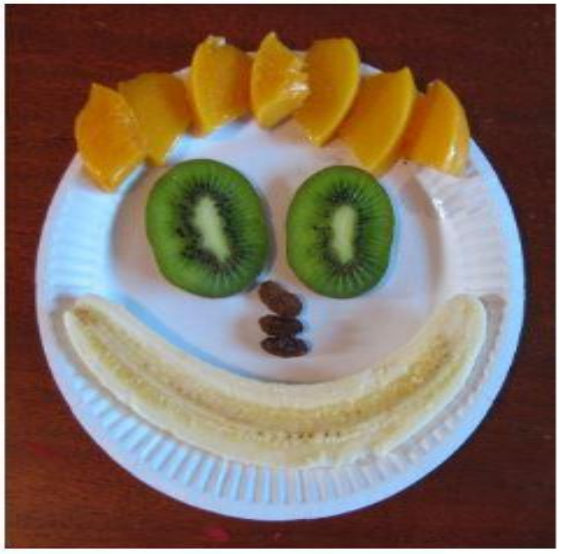
Suddenly, your mind goes blank. There's no obvious solution.

But given any two attempts to copy the picture, we can always determine which one is better, essentially by measuring the difference between the original and the copies.

PATCHWORK: Face = Sum of Fruits?

The Italian painter Arcimboldo enjoyed the puzzle of trying to approximate a human face using vegetables.





This is **not** the sort of problem you expect to give to a computer!

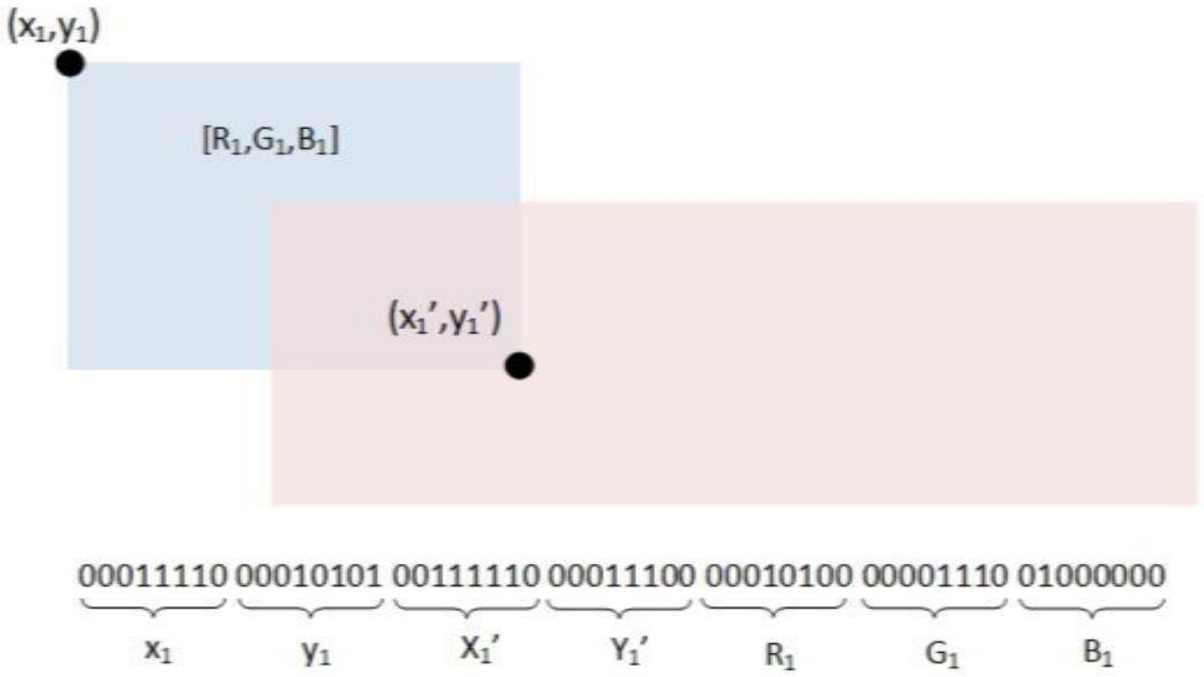
PATCHWORK - The Genetic Information

To make a genetic algorithm for this problem, we want to start by setting up the genetic information.

We assume the image is a **jpg** file containing 256×256 pixels. A candidate solution would be 32 colored rectangles. Each rectangle has a position and a color. Using the **jpg** format, the lower left corner is a pair (xl, yl) between 0 and 255, and (xr, yr) is similar. The color of the rectangle is defined by three integers, r, g, and b, also between 0 and 255. Thus, our numeric representation of a candidate solution is 32 sets of $(xl, yl, xr, yr, r, g, b)$, for a total of 224 integers.

A number between 0 and 255 requires 8 bits to specify, so our genetic information describing one candidate would require $224 * 8 = 1792$ bits.

PATCHWORK: One "Gene"



We have 10 candidates or "chromosomes". Each chromosome describes the color and position of 32 rectangles. The description of a single rectangle is termed a "gene".

PATCHWORK - The Fitness Function

Supposing we have specified a candidate solution x; then we must be prepared to evaluate its fitness $f(x)$.

To evaluate our candidate, we can simply create a 256×256 **jpg** file from the rectangles, and sum the (r, g, b) color differences pixel by pixel:

$$f(x) = \sum_{i=0}^{255} \sum_{j=0}^{255} |r1(i,j) - r2(i,j)| + |g1(i,j) - g2(i,j)| + |b1(i,j) - b2(i,j)|$$

where $r1$ and $r2$ are the reds for original and candidate, and so on.

Note that a perfect solution would have $f(x) = 0$, and that low values of $f(x)$ are better than high ones. We remember that for this problem, we are *minimizing* $f(x)$ instead of maximizing.

PATCHWORK - The Fitness Function

The rest of the procedure is pretty straightforward. We have to generate an initial random population of candidates, say 10 of them.

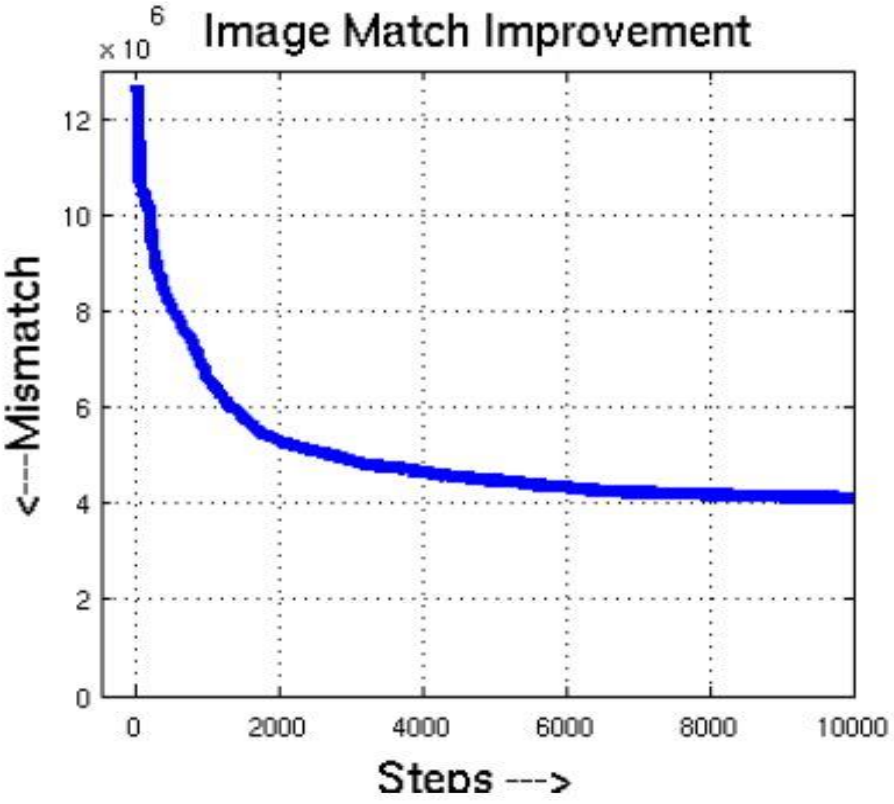
Our generation step will kill the 2 worst candidates, and replace them by the two children produced by breeding the two best candidates. Then we will also pick one candidate at random and hit it with a random mutation.

A typical run of the program might involve thousands or hundreds of thousands of generations. To monitor the progress of the program, we can look at how the fitness function evaluations are changing, or we can examine successive approximations to our picture.

PATCHWORK - Program Outline

```
read the original image "RGB_A";
generate 10 random candidates "B1", "B2", ..., "B10".

for 10,000 steps:

  convert each B to an image RGB_B;
  compare RGB_B to RGB_A to compute score;
  sort candidates, lowest to highest score.

  delete candidates B9 and B10;
  cross two remaining candidates, replacing B9 and B10;
  mutate one candidate of B2 through B8.

end
```

PATCHWORK - The Fitness Function



The graph of the fitness function values suggests that our rate of improvement is leveling off, and that it might be time to take a look at what we've computed!