Vision as Inverse Graphics

Machine learning techniques towards a program-based model for scene understanding

Vinjai Vale

May 21, 2017
MIT PRIMES CS conference

Scene Understanding

Recognize objects and components

Answer questions about scene

Why is scene understanding hard

Scene understanding is easy for humans.

What is this a picture of?

What is the man doing?

What is the man’s jersey number?

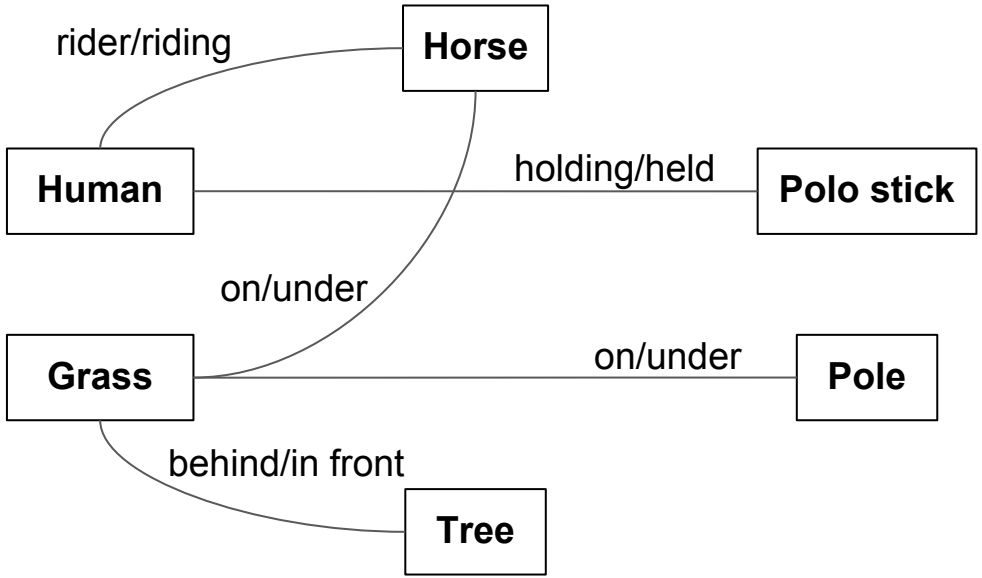
How fast is the horse going?

These questions are second nature for humans to answer, but are really difficult for a computer.

Scene Understanding

Scene: Polo

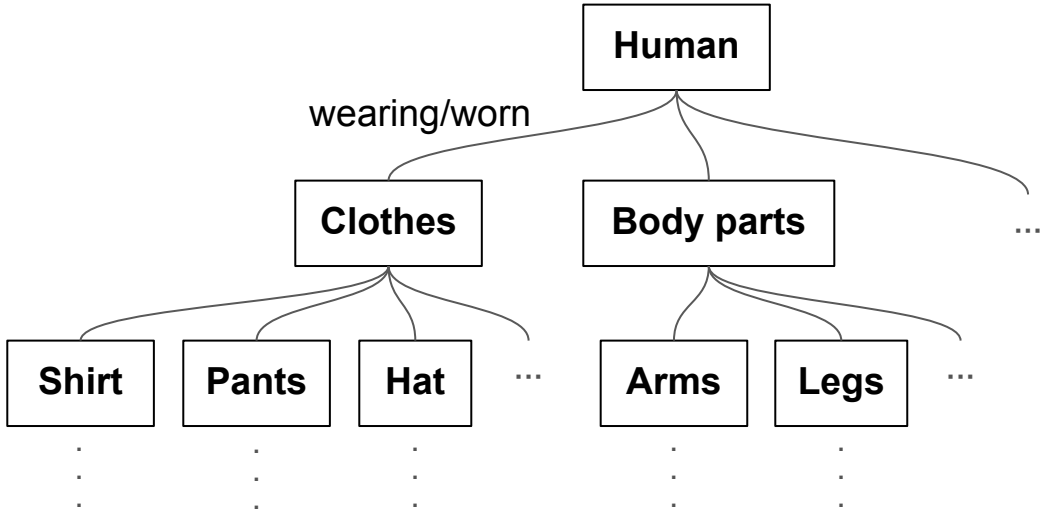
Primary Objects



Scene Understanding

Object: Human

Components (Secondary/Tertiary Objects)



Current approaches

Classify objects and *infer* the scene context based on the types of objects present

Types of objects present: Horse, Human, Stick, Grass, Tree —> Scene is Polo

Top-down approach; gets the gist of the scene

Google image captioner

Image source: Google paper “Show and Tell: A Neural Image Caption Generator,” CVPR 2015

Another example

Computer: A zebra, of course!

Human: A horse in a zebra costume, of course!

Image source: Google Images

Current approaches lack **compositionality**

A *compositional representation* is one where complicated objects or scenes are represented by putting together simpler parts.

Compositionality is second nature to humans, but not to computers!

Alternative approach

Goal: accomplish scene understanding by creating an **abstraction** that includes information about the following:

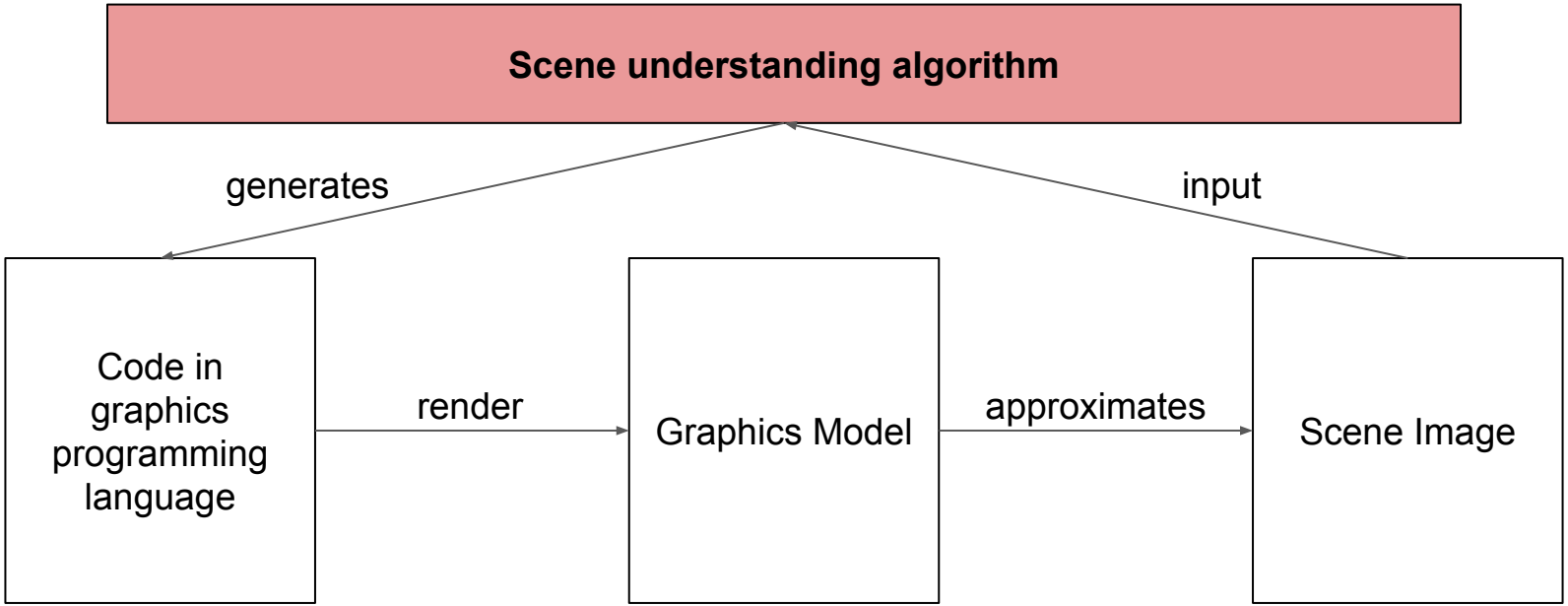
- The type of each object and each component
- How each component is related to the object that it is a part of, and vice versa (thereby encoding the specifics of each object)

Vision as inverse graphics

Alternate paradigm:

Analyze an image by attempting to synthesize it

Program-based model



Reduction of the problem

How can we create an algorithm that generates programs of **lines and circles** to match given scenes of lines and circles?

The scene will consist of one to two black lines and circles drawn on a white 100 pixel by 100 pixel binary raster canvas.

MetaPost (MP)

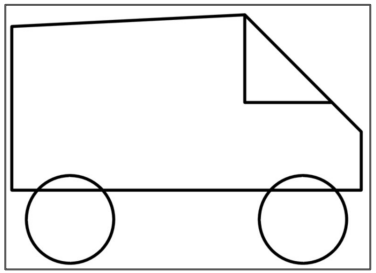
MetaPost (MP) is a graphics programming language similar to LaTeX's Ti*k*Z — except it renders PNGs six times faster from command line.

Images you can draw with lines and circles range from:

quite simple...

```
draw (38,78) -- (80,35);
draw fullcircle scaled 6 shifted (57,91) withcolor black;
```

...to more complex



```
draw (20,40) -- (80,40) -- (80,50) -- (60,70) -- (20,68) -- cycle;
draw (75,55) -- (60,55) -- (60,70);
draw fullcircle scaled 15 shifted (30,35) withcolor black;
draw fullcircle scaled 15 shifted (70,35) withcolor black;
```

Representation of MP programs

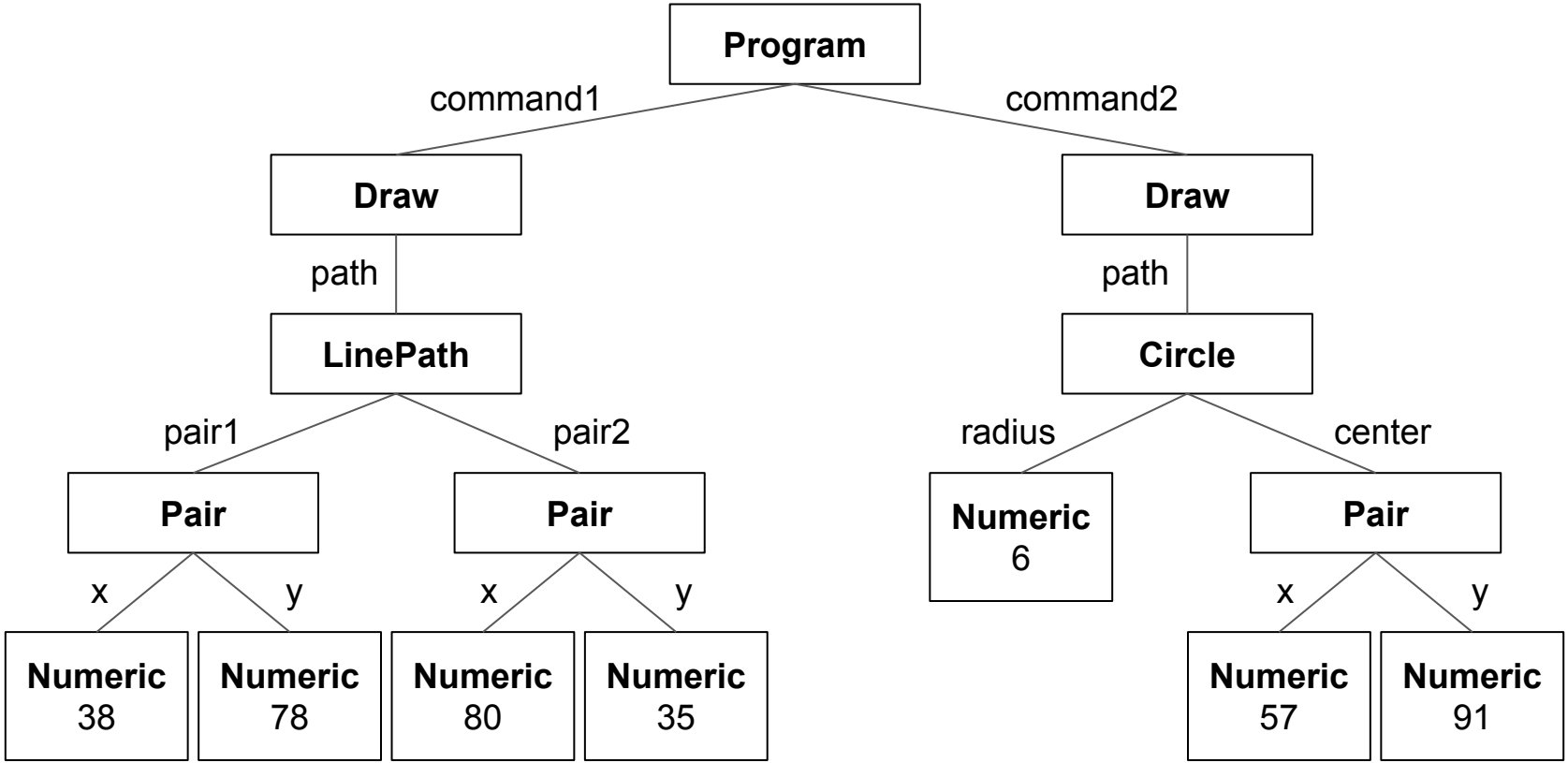
```
draw (38,78) -- (80,35);
draw fullcircle scaled 6 shifted (57,91) withcolor black;
```

Pure code offers no compositional structure

Syntax tree

Code:

```
draw (38,78) -- (80,35);
draw fullcircle scaled 6 shifted (57,91) withcolor black;
```

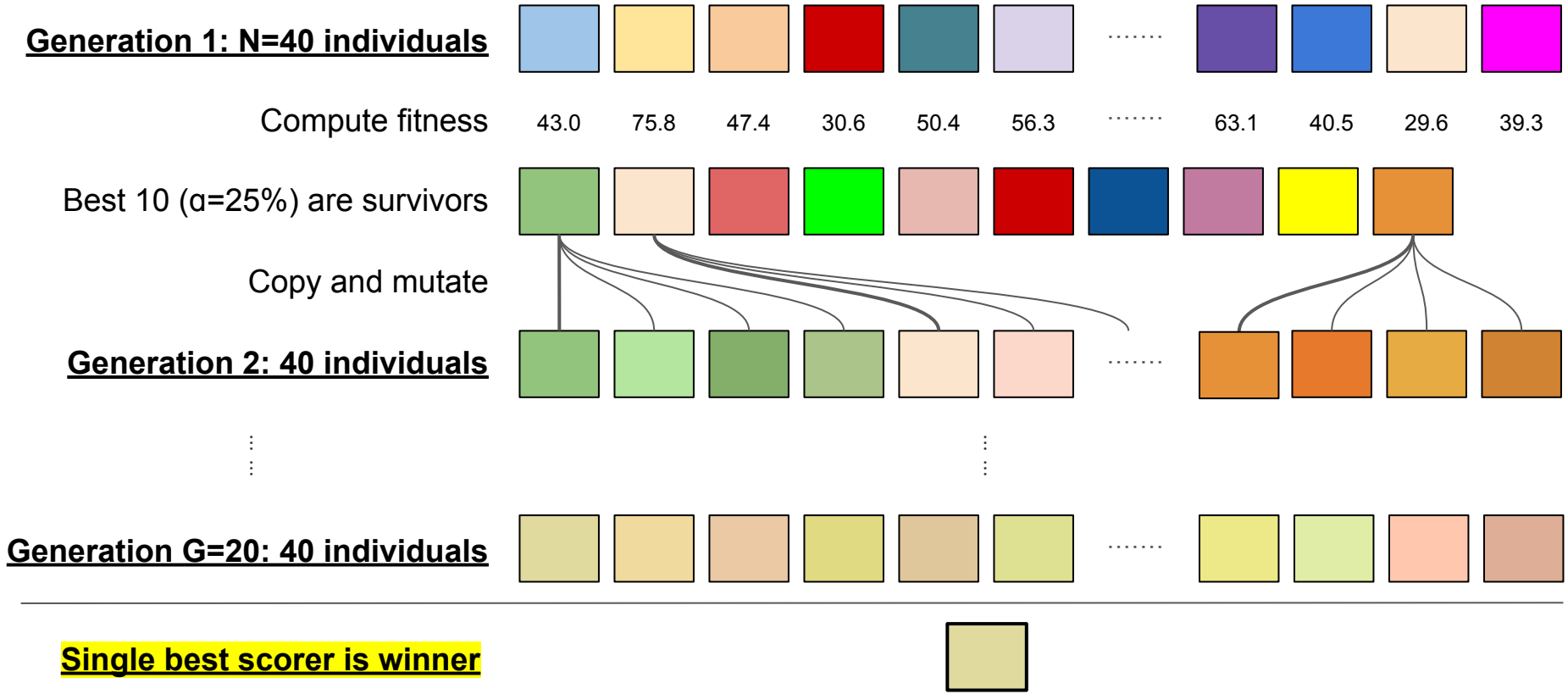


Search Problem

There are ~10^{16} different programs just consisting of two lines; how do we search through such a large possible space?

Technique inspired by evolution: ***genetic algorithm***

Genetic algorithm



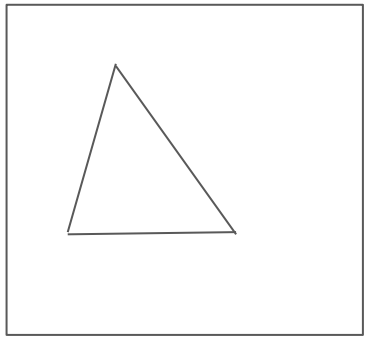
Fitness as a distance metric

We determine fitness by calculating the **distance** between a candidate and the goal scene image. Fitness and distance are inversely proportional.

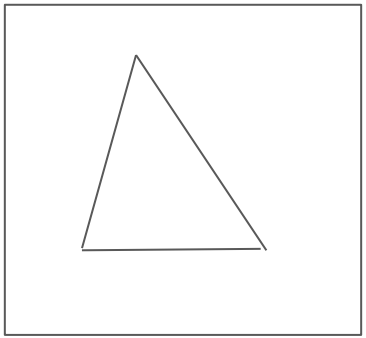
Naive pixelwise distance

$$\texttt{pixelwiseDist}(A_1, A_2) = \sum_{\substack{0 \le i < m \\ 0 \le j < n}} (A_1[i][j] - A_2[i][j])^2$$

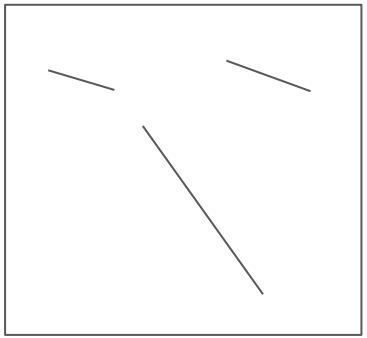
Downside of pixelwise distance



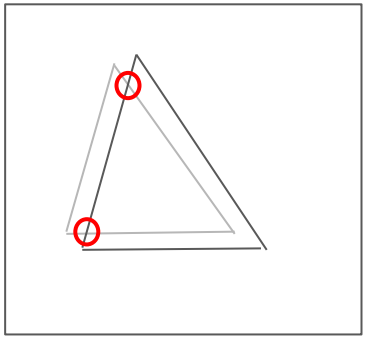
scene



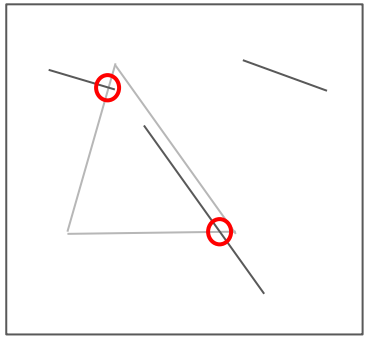
candidate 1



candidate 2



Does not account for proximity



Hausdorff distance

For two sets of points P and Q:

$$\texttt{hausdorffDist}(P,Q) = \max\{\sup_{p\in P}\inf_{q\in Q} d(p,q), \sup_{q\in Q}\inf_{p\in P} d(p,q)\}$$

Farthest distance between each point in P and its closest point in Q, and vice versa.

Desired mutation function

1. From any starting point, the mutation function should be able to hone in on the goal scene image, like gradient descent

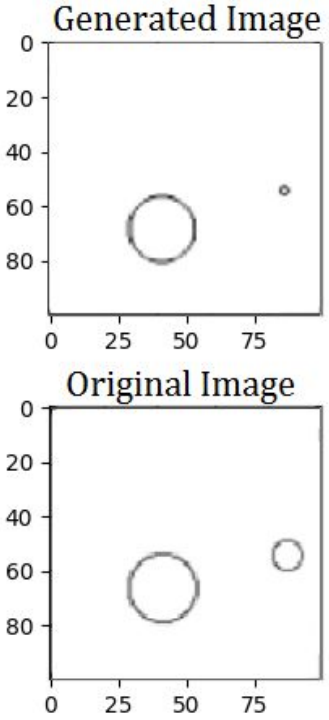
2. The mutation function should be able to jump out of a local minimum within a few generations of falling into it, which is something gradient descent on its own cannot do.

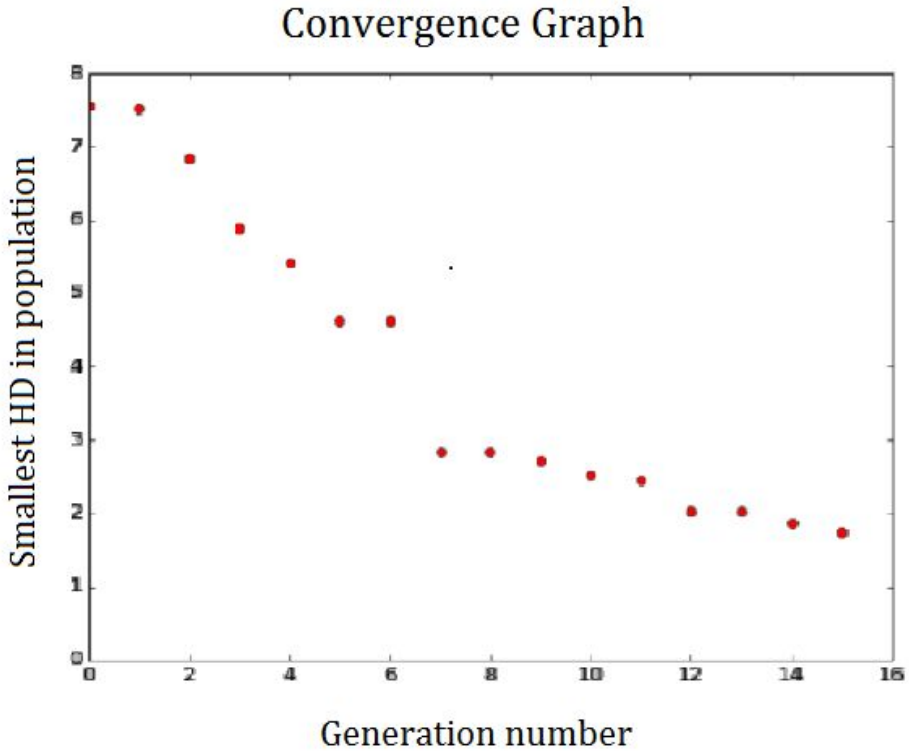
3. The mutation function should be able to bridge the gap between near (low distance) and exact (zero distance).

Mutation

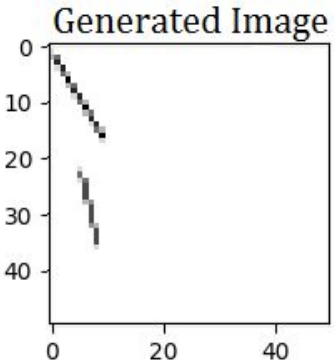
- Trickles down recursively to each Numeric
- New value of Numeric chosen from Gaussian distribution that narrows over time

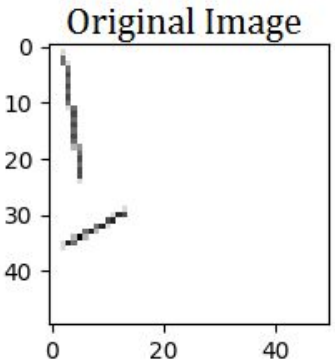
Moderate initial success

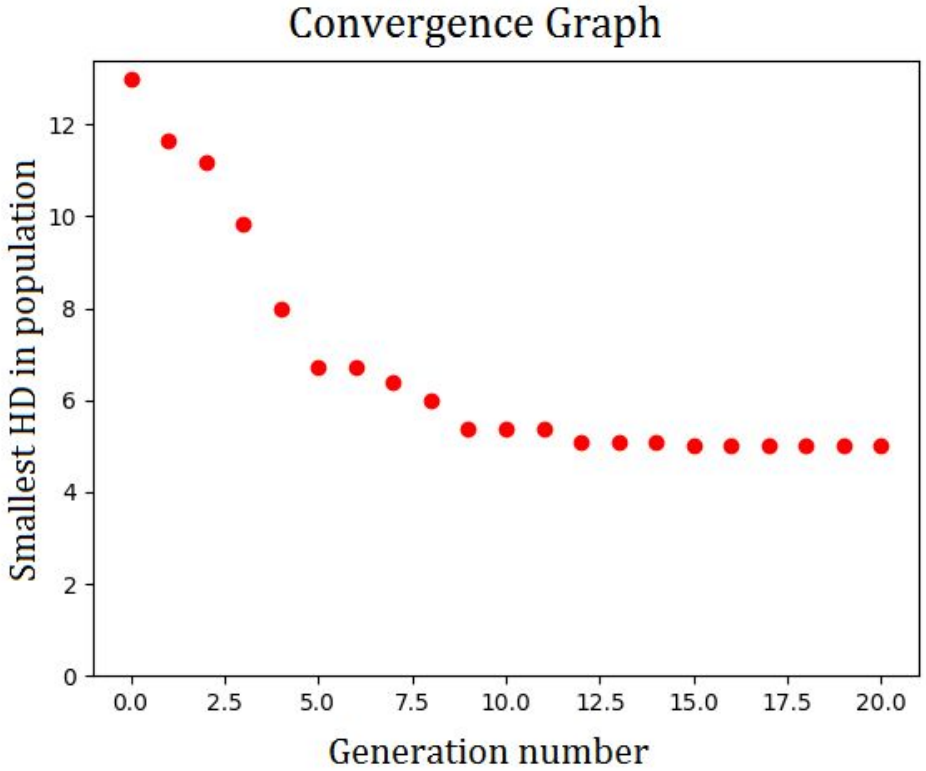




Local minimum trap







Avoiding the trap

- Hausdorff distance too discrete? Consider other distance metrics
- Tweak survival rate α
- Tweak mutation Gaussian’s σ decay and reset rates
- N = 40, G = 20 may be too small

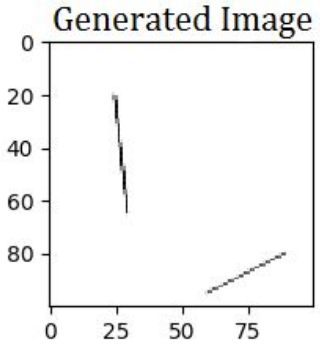
Other distance metric

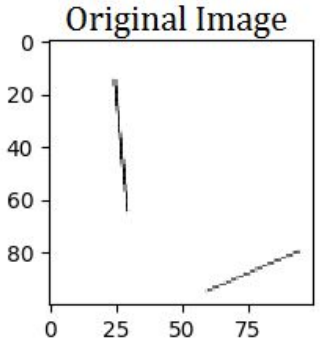
One possible alternative is the *average Hausdorff distance*:

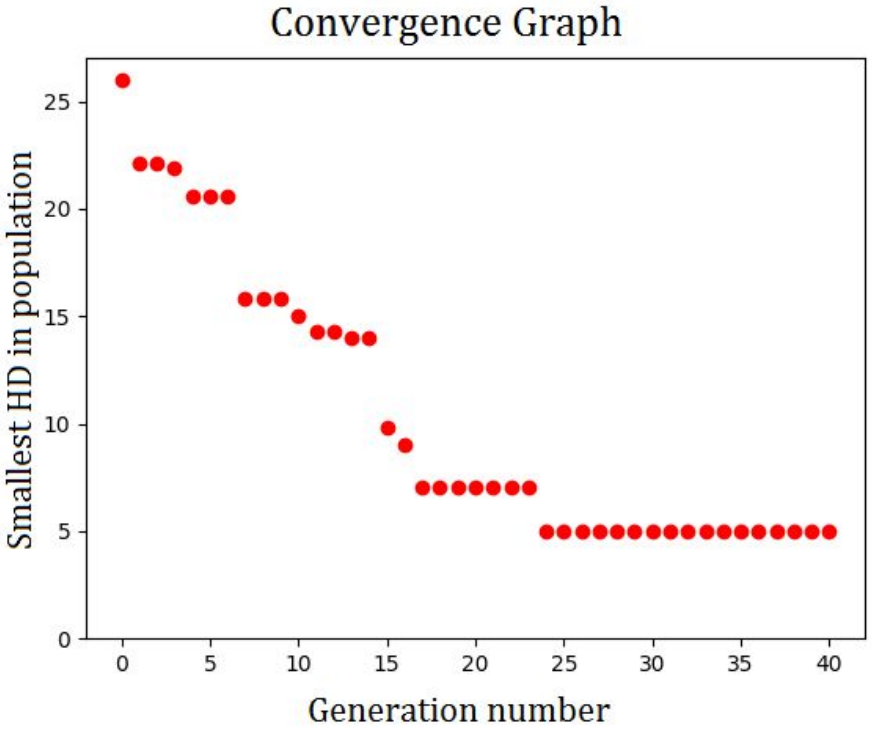
$$\texttt{avgHausdorffDist}(P,Q) = \max\left\{ \frac{1}{|P|}\sum_{p\in P}\inf_{q\in Q} d(p,q), \frac{1}{|Q|}\sum_{q\in Q}\inf_{p\in P} d(p,q) \right\}$$

Average distance between each point in P and its closest point in Q, and vice versa.

More run examples

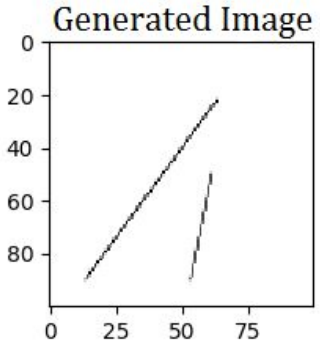


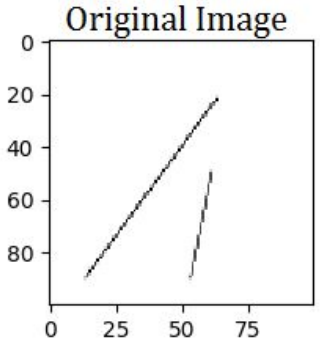


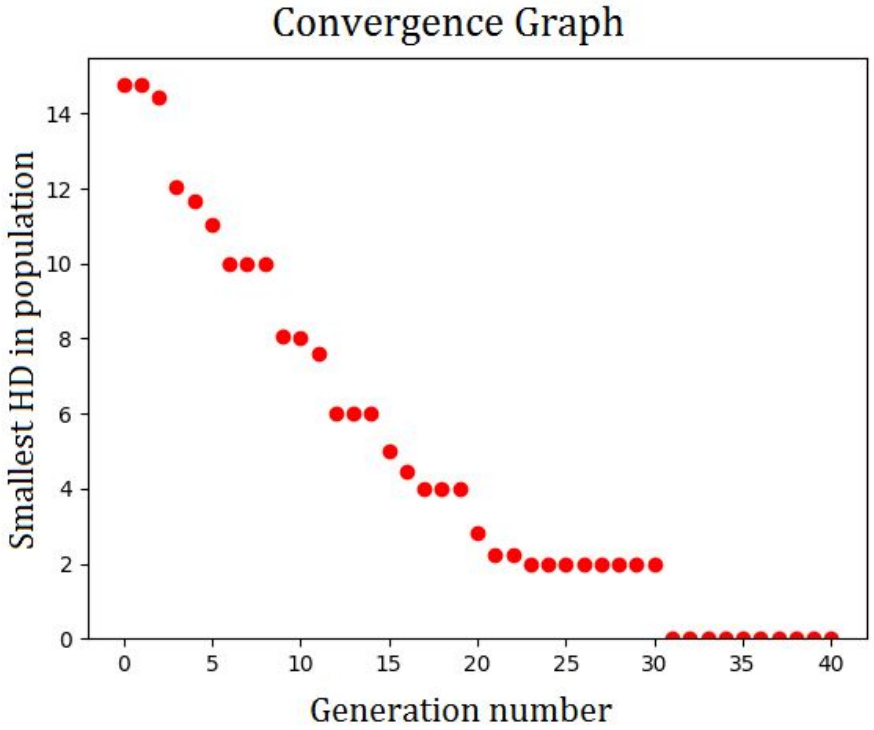


N = 100, α = 0.25, G = 40; s = 5

More run examples

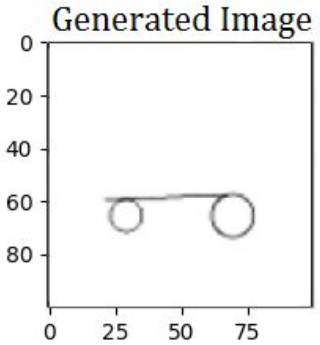


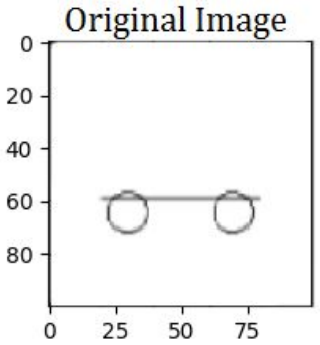


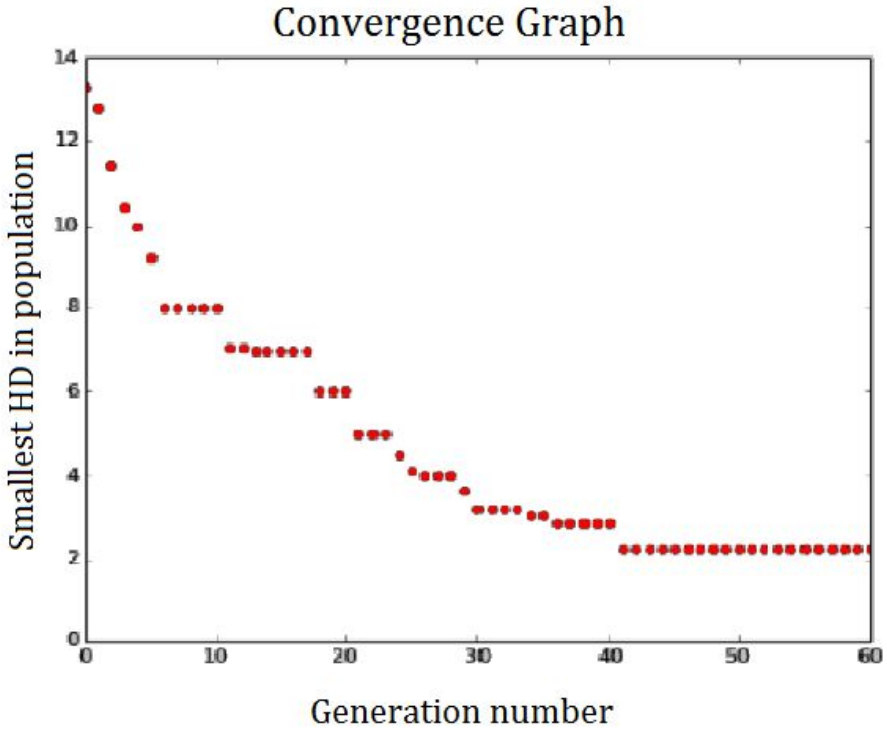


N = 100, α = 0.25, G = 40; s = 2

More run examples







N = 150, α = 0.25, G = 60; s = 2

Future work

Programs that grow new objects

- Start with one line or circle
- Fit it somewhere on the image
- Introduce the next line or circle once the partial image matches
- Repeat

Mutating one object at a time is much faster than trying to mutate everything at once

Future work

Neural network to assist in the growth of a program

Looks at partial image and goal scene image and suggests what type of object needs to be added

Long-term goals

This is preliminary work as a proof-of-concept for the program induction approach to computer vision

Farther down the road, the goal is to expand to more complex 2D color images and eventually to 3D scenes, with representations in 3D graphics systems like CAD software.

Acknowledgements

- Kevin Ellis, my mentor
- Sean Campbell, my computer science teacher
- MIT PRIMES, for this opportunity
- My family, for their support
- All of you, for being a great audience

Questions?

Benefits of syntax tree: #1

A syntax tree takes full advantage of object-oriented programming.

Python classes for:

- Program
- Draw command
- LinePath
- Circle
- Pair (a 2D point)
- Numeric (an integer value, usually a coordinate)

Benefits of syntax tree: #2

It is easy to convert a syntax tree into MP code.

Programs are stored internally as syntax trees, and to obtain the image output one simply runs a recursive toCode() function on the tree's root and compiles the image to a PNG via a command-line call to the MP compiler.

Mutation: Gaussian distribution

The Gaussian distribution is centered around the current value of the Numeric, and it has standard deviation σ which determines how volatile the mutation will be

Trade-off between volatility, speed, and accuracy

- Large σ (take larger steps)
  - Approach correct value more quickly
  - Can’t settle down on correct value
- Small σ (take smaller steps)
  - Approach correct value more slowly
  - Settles down effectively on correct value

Mutation: Picking σ

Potential solution to the trade-off problem: why not do both?

- At start, σ is relatively large
- Over time, σ decays exponentially
  - Narrows Gaussian distribution
  - Jump in each mutation decreases over time; hones in on value
- Occasionally, σ will reset to its original value
  - Accelerates steps if they are taking too long to approach correct value
  - Helps jump out of local minimum traps

Genetic algorithm

This algorithm has a few parameters:

- N = 40: population size
- α = 0.25: survival rate
- G = 20: number of generations

Grid snapping

Rather than allow lines and circles to go wherever, they can be snapped to a grid

All Numerics are rounded to multiples of s, the snap factor

Genetic algorithm

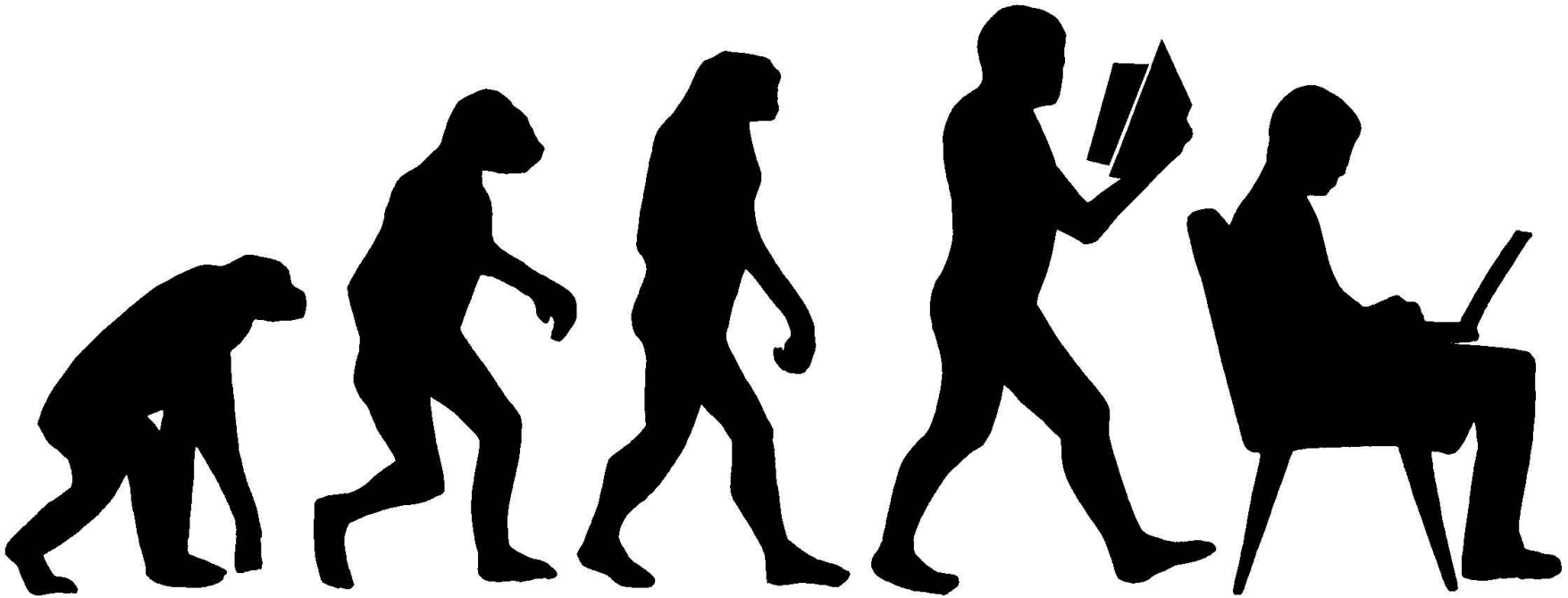


Image obtained from Wikimedia Commons under a “free for reuse and modification” license

Genetic algorithm

For much of the remainder of the talk I will be discussing populations that have gone through significant genetic mutations and frequent purging to ensure only the very fittest survive.

The “organisms” in my population are MetaPost programs!