Evolutionary computation for turbulence-degraded space images

Jorge Munoz

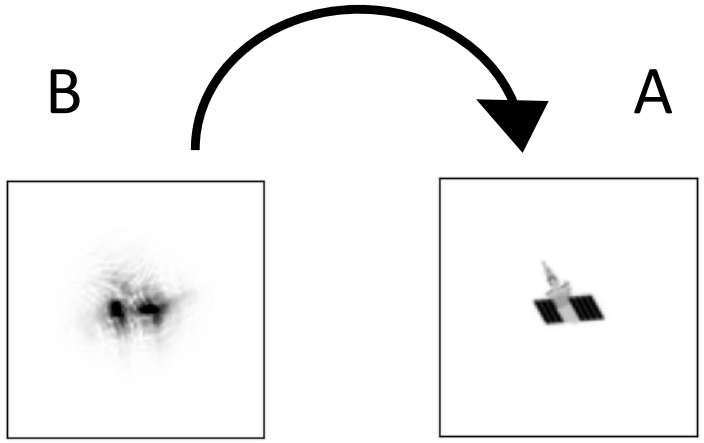
The University of Texas at El Paso

Air Force Research Lab Summer Faculty Fellow

Advisor: Rao Gudimetla (AFRL Maui Space Surveillance Branch)

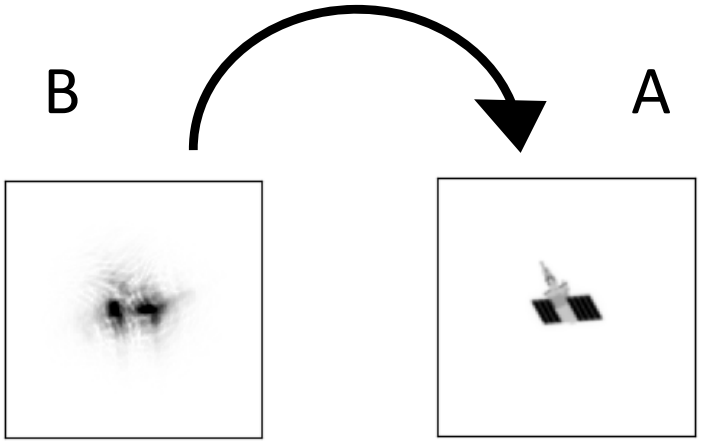
August 19, 2019

Problem statement:



How do we go from B to A?

Answer:



We can’t… at least not uniquely

Content of this talk

- Complexity considerations – let's quantify how difficult the problem really is in order to make an informed decision on which artificial intelligence techniques could work. (Known unknown before the summer)

  Spoiler alert: it is very difficult

Content of this talk

- Complexity considerations – let's quantify how difficult the problem really is in order to make an informed decision on which artificial intelligence techniques could work. (Known unknown before the summer)

  Spoiler alert: it is very difficult

- Evolutionary computation – let's approach it by iteratively reducing the state-space complexity, finding plausible solutions in the current state-space, and porting those solutions to use as ansatzes at the next state.

  Spoiler alert: it is a promising approach but it is still WIP

As of last week, your old boss is my new boss

Access, excellence, <u>impact</u>

46% of student who requested financial aid had family income of less than $20k/year

About 40% of our students are part-time

5% live in Juarez

25,000 students

3rd in the country in number of BS,

5th in MS degrees to Hispanics

7th Hispanics who go on to PhD programs

We serve the Borderplex – 2.7 million people largest bilingual and binational workforce in the Western Hemisphere

We won the 1966 NCAA Basketball championship and got a movie made about that (Glory Road)

UTEP ATTAINS NATIONAL RESEARCH TOP TIER RANKING

The University of Texas at El Paso has attained the R1 designation (top tier doctoral university with very high research activity) in the Carnegie Classification of Institutions of Higher Education.

Only R1 University that is Mexican-American majority

Only taste of Bhutanese architecture in the Western hemisphere

First female students enrolled in 1916

In 1955 we became the first university in Texas to desegregate

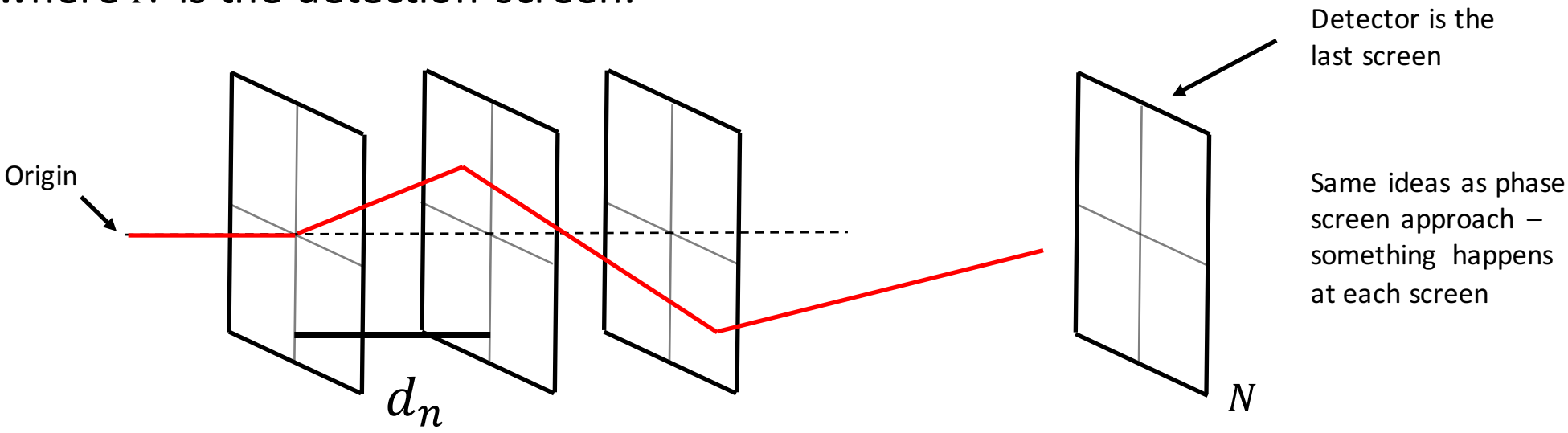
https://www.flickr.com/photos/ob-1/1771801424

A ray of light travels in the direction defined by angles $\vartheta_0 = \phi_0 = 0$.

ϑ_n and ϕ_n are angles with respect to the normal of plane n and the projections on the plane of the ray are orthogonal.

Planes n and $n+1$ are separated by distance d_n and the ray is propagated forward at each time step t_n.

The path is described by the set of N coordinates $\{(\vartheta_0, \phi_0, d_0), \dots, (\vartheta_N, \phi_N, d_N)\}$, where N is the detection screen.



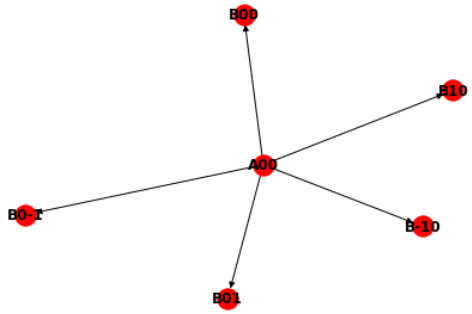
The ray makes a 'decision' at each screen that affects its trajectory. Let's define the set of decisions $S = \{\text{up}, \text{down}, \text{left}, \text{right}, \text{straight}\}$.

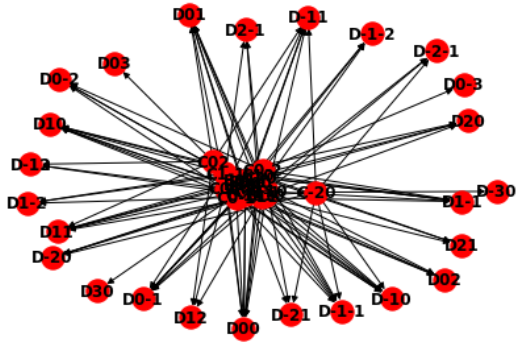
The number of possible states for the system after N screens is given by $\mathcal{F}_N = \mathcal{F}_{N-1} + 4N$, except for $\mathcal{F}_0 = 1$, so the logarithm of the state-space complexity is:

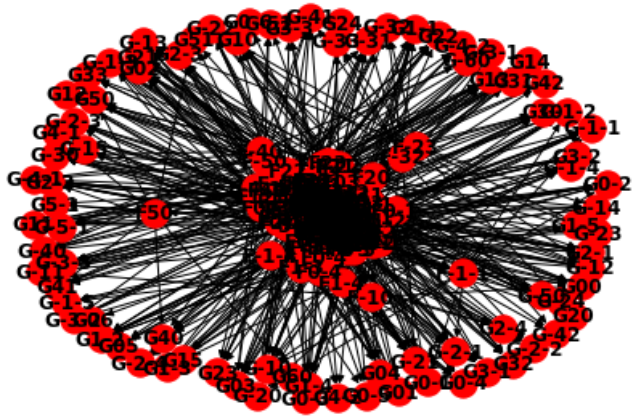
$$\log(\mathcal{F}_N) \approx 2\log(N).$$

The state-space complexity of chess, the number of possible chess games, is about 10^{47}. For Go it is about 10^{170} and before AlphaGo it was intractable for computers. There are between 10^{78} and 10^{82} atoms in the universe.

If d_n is of the order of the Kolmogorov microscale, the state-space of the ray propagating in turbulence reaches the size of the state-space of Go in as little as 2 cm. And the model is pretty naïve.







The decision tree can be represented as a mathematical graph and the path of a ray is given by the simply ordered set consisting of the nodes it traversed, e.g. {‘A00’, ‘B-10’, ‘C0-2’, …}

Just for fun, let's call this the Plinko model

Separation between screens

d_n

Ray makes a decision at each screen

If a <u>single</u> ray of light is detected in the slot labeled 'CAR' at the bottom, can we determine from which slot it started?

Just for fun, let's call this the Plinko model

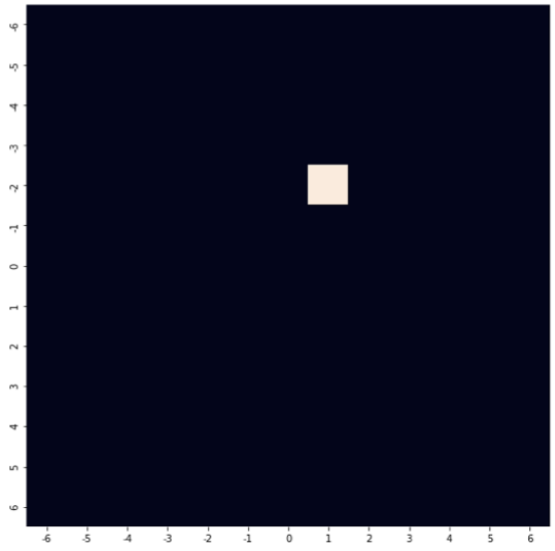
If we have 10,000 rays collected at the bottom all originating from the same slot at the top, can we determine from which slot they started?

Most real-life cases are in between: we have more than one detection but not enough to trace their origin with a small margin of error using traditional statistics.

Of course, objects in real life consist of a continuum of such points.

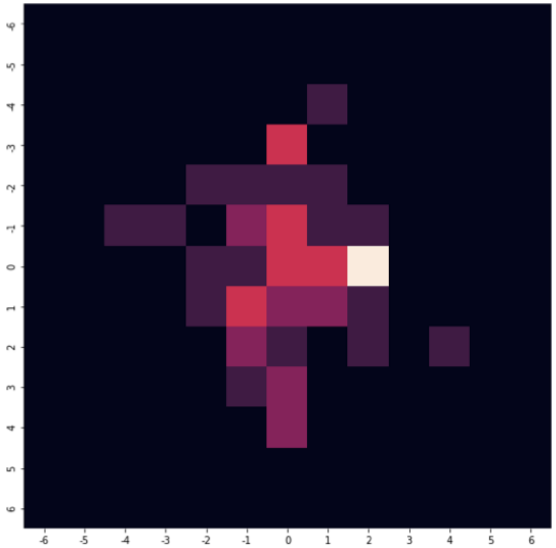
Distribution of end nodes ‘Gxy’ for simply ordered sets with initial node ‘A00.’

1 traverse



100 traverses

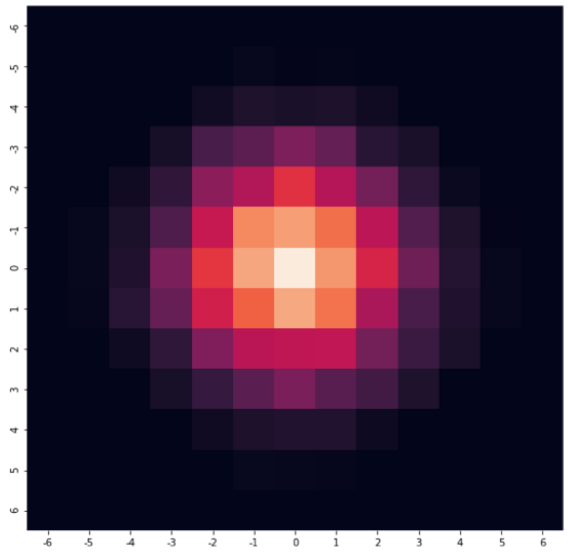
Some paths explored



Speckle

10,000 traverses

All paths explored



Blur

The sun produces 10^{47} photons per second.

We recover typical patterns with this model when insufficient photons

Let's formalize it a bit.

Are there any simply ordered sets of traverses that have different initial nodes but the same end node?

Not for 1 layer (which corresponds to no turbulence), but for 2 layers, e.g.:

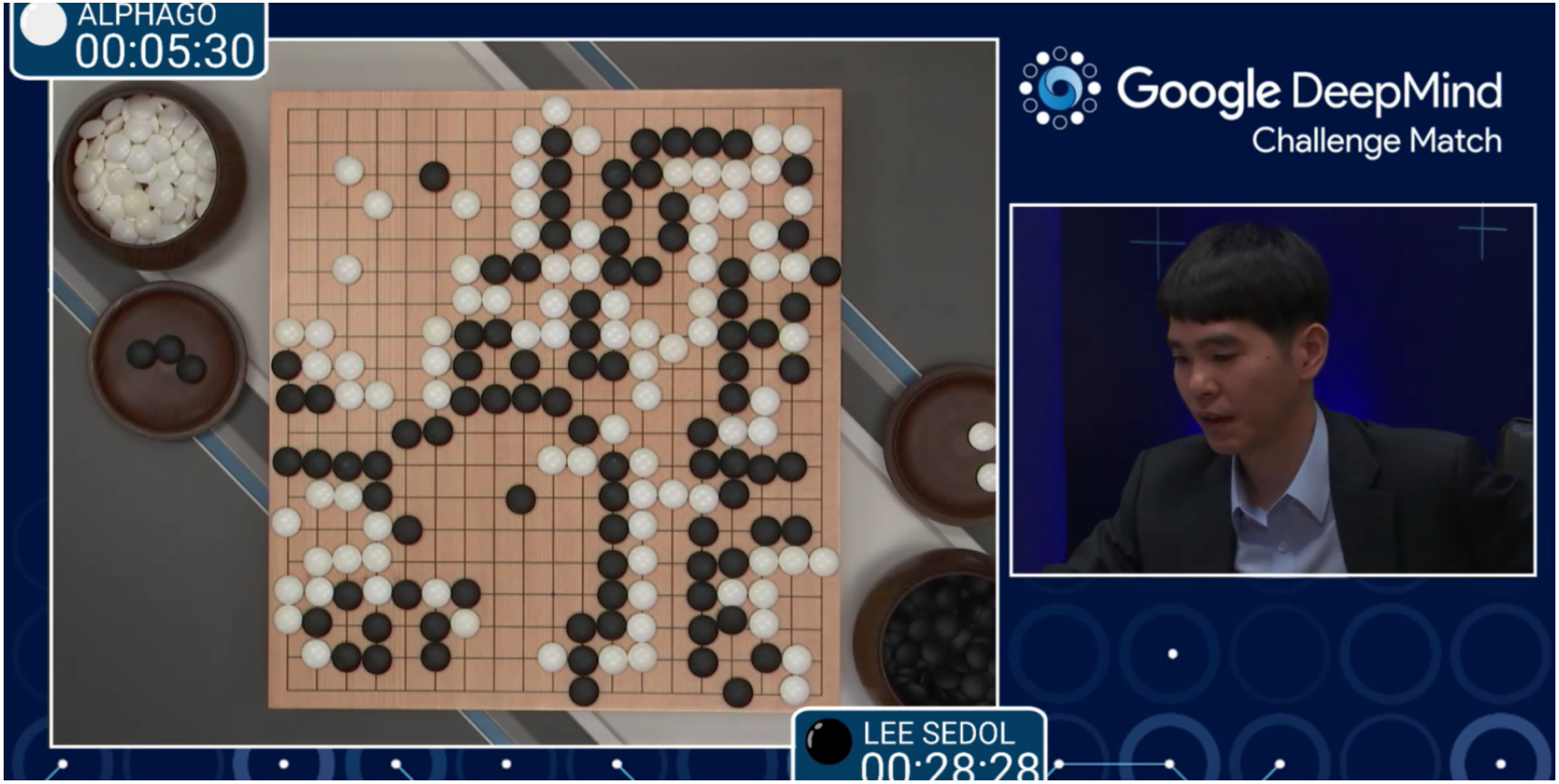
{'A00', 'B-11'}	{'A10', 'B00'}
{'A00', 'B00'}	{'A10', 'B10'}
{'A00', 'B10'}	{'A10', 'B20'}
{'A00', 'B01'}	{'A10', 'B11'}
{'A00', 'B0-1'}	{'A10', 'B1-1'}

From an information theory point of view, we transmit a message through a noisy channel, e.g. the message is ‘00’ but instead we get ‘-42.’

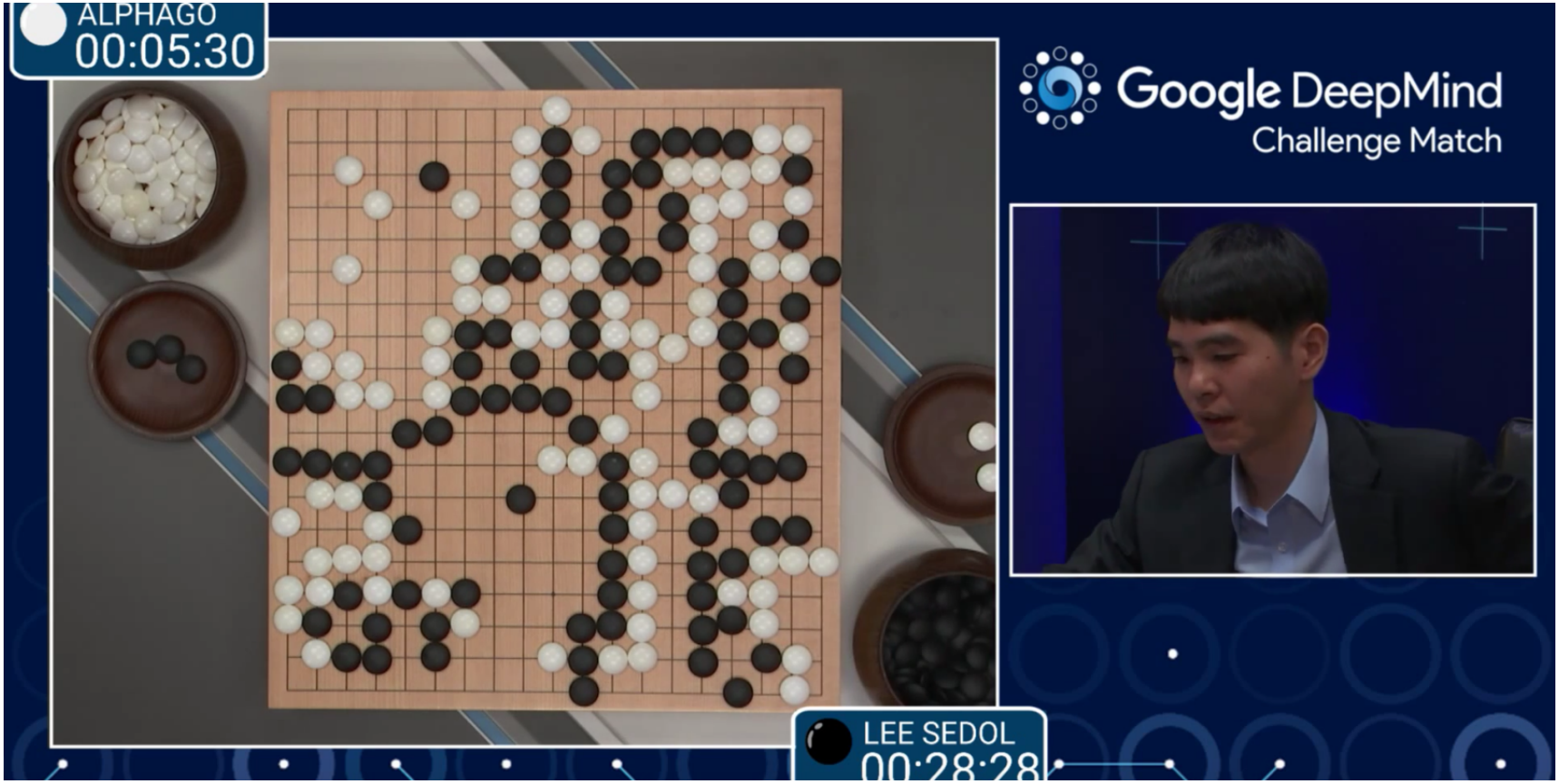
{‘A00’, …, ‘G-42’}

“If the channel is noisy it is not in general possible to reconstruct the original message or the transmitted signal with *certainty* by any operation on the received signal.” - Claude Shannon in A Mathematical Theory of Communication, 1948.

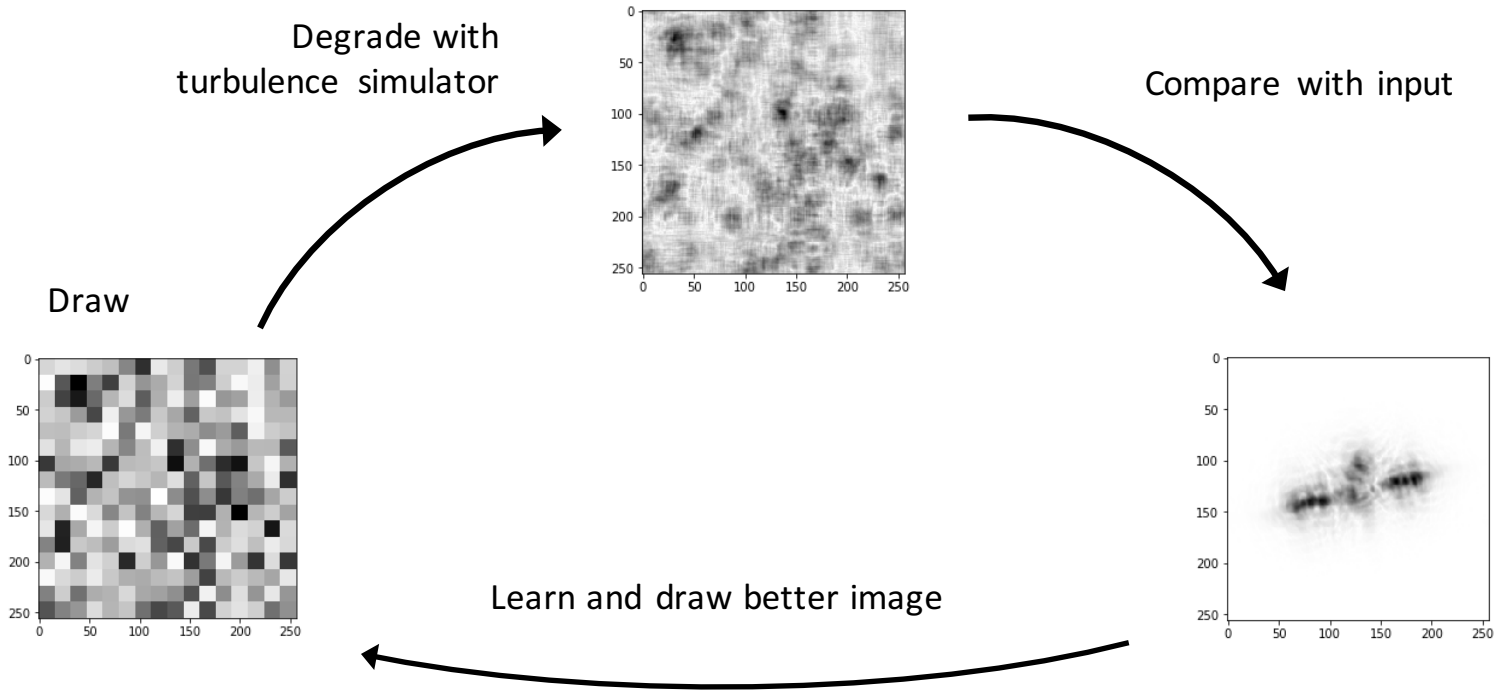
Emphasis his own.



The following pattern is measured on a 19x19 pixels detector. Can we determine the sequence of screens/layers/moves that created it? This is different than predicting the sequence that wins the game (AlphaGo).



One option is to enumerate all possible games with 100 stones at the end. A more feasible option is to simulate m games and see which end up looking more like the target, then combine the best to try to converge on a solution.



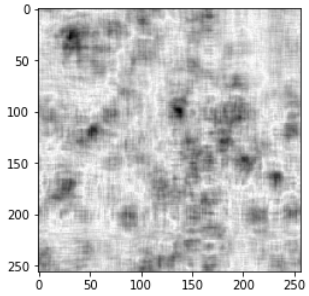
[There's] nothing you can see that isn't shown.

- John Lennon in All you need is Love, 1967.

How do we simulate the turbulence?

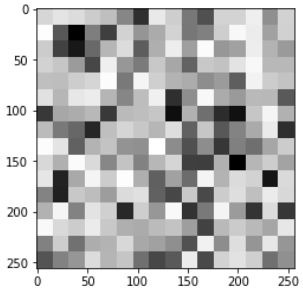
Which penalty function(s) do we use?

Degrade with turbulence simulator

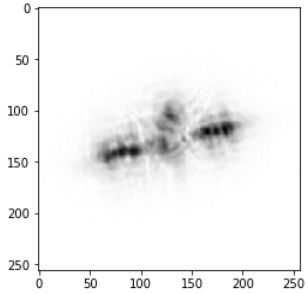


Compare with input

Draw



How do we keep what we have learned?



Learn and draw better image

How do we represent a drawing/solution in code?

How do we make improvements?

The number of things that can be shown is huge. For 256x256 pixels and maximum intensity per pixel to 512, what is the number of images we can draw?

$512^{256\times256}$, although we have to consider symmetries. How many of those are good enough?

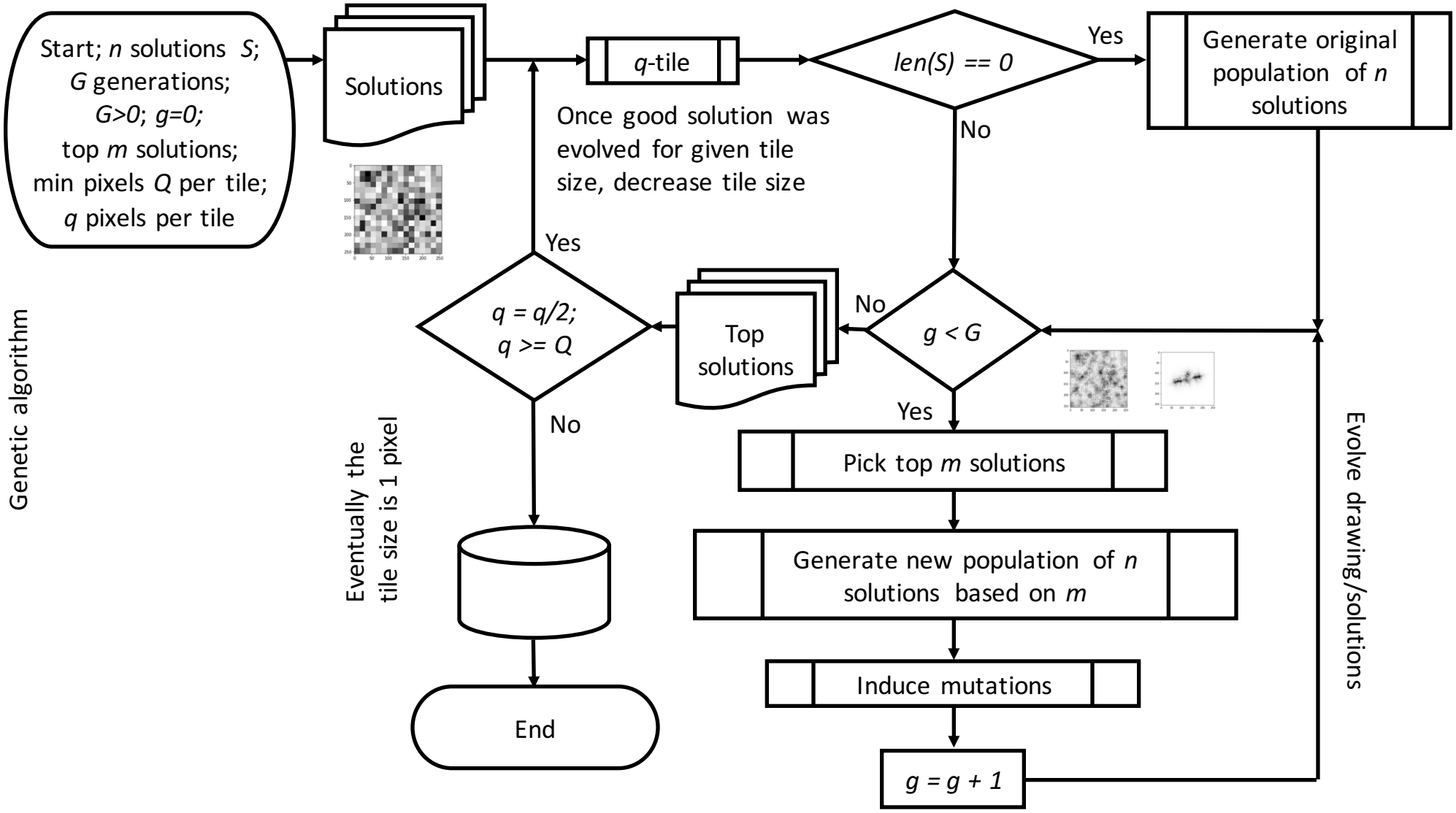
512 ^ 65536

Infinity

How do we represent a drawing/solution in code?	List of intensities
How do we simulate the turbulence?	Phase screen method with FFT as shown in (Schmidt, 2010), but algorithm is agnostic to this.
Which penalty function(s) do we use?	I have been using MSE, SSIM (Z. Wang et al. 2004), and some original functions.
How do we keep what we have learned?	I am using a genetic algorithm, so the fittest solutions are kept and their code passes preferentially to new generations.
How do we make improvements?	Original mutation operations, but it is though.

For sufficient intensity:

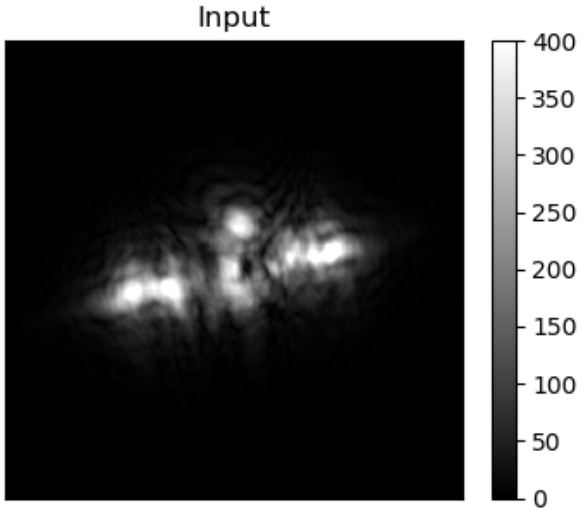
The secret is to reduce the complexity of the problem by 'tiling' the image, find the best solutions at a given tiling level, and increase the complexity (smaller tiles) a little bit while using the previous solutions as starting points.

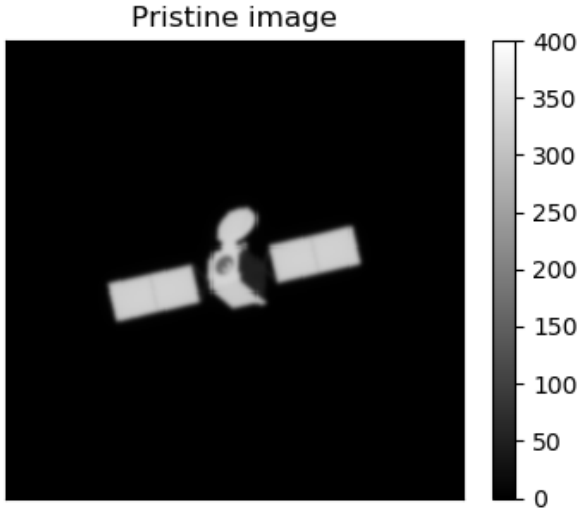

Start; n solutions S;
G generations;
G>0; g=0;
top m solutions;
min pixels Q per tile;
q pixels per tile
Solutions
q-tile
len(S) == 0
Yes
No
Generate original population of n solutions
Once good solution was evolved for given tile size, decrease tile size
Yes
q = q/2;
q >= Q
No
Top solutions
No
g < G
Yes
Pick top m solutions
Generate new population of n solutions based on m
Induce mutations
g = g + 1
End
Genetic algorithm
Eventually the tile size is 1 pixel
Evolve drawing/solutions

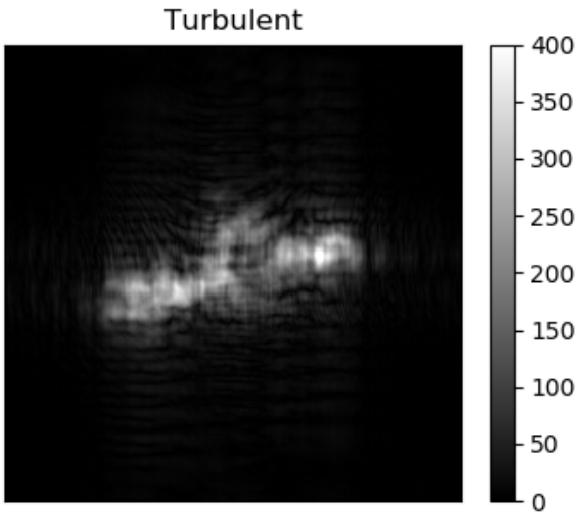
$\sigma_R = 0.02$

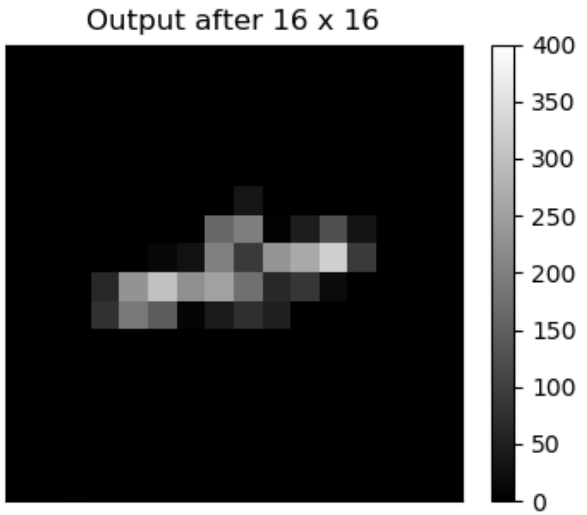
& sufficient intensity

‘measurement’









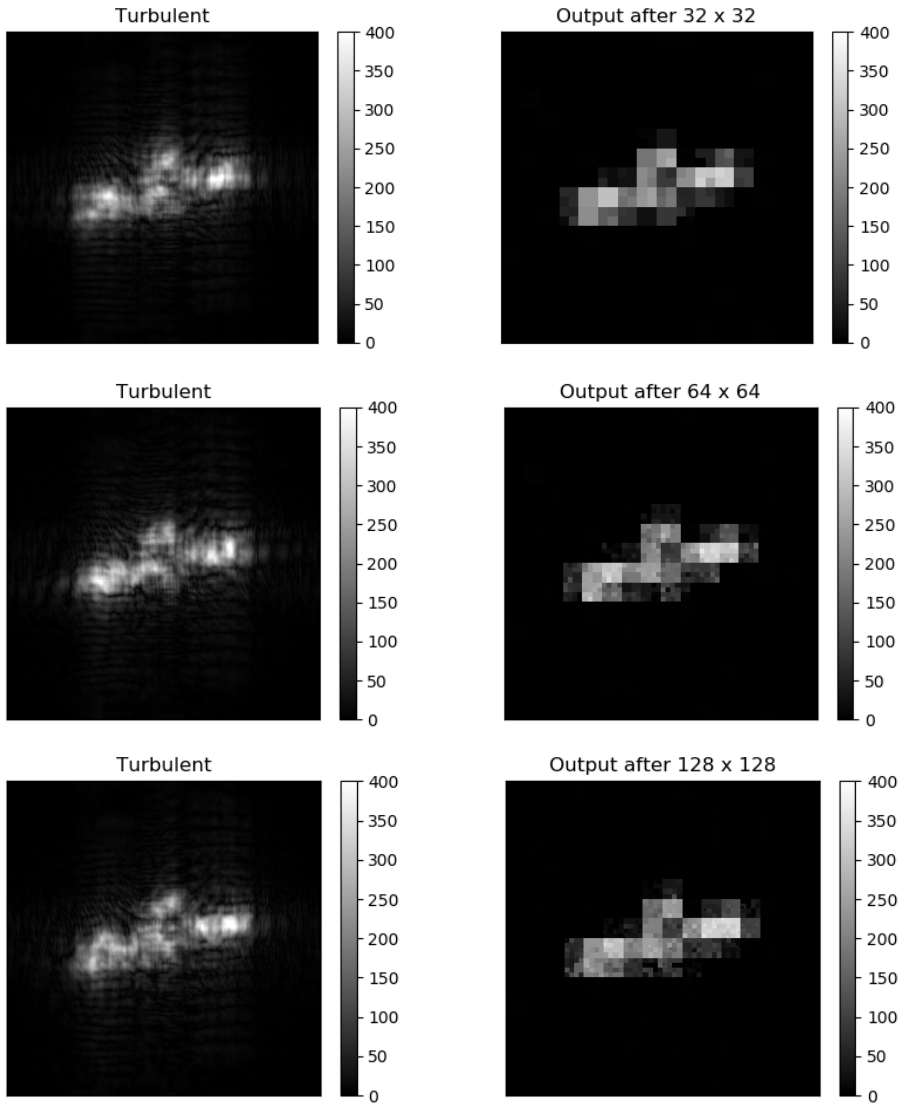
Pristine images by Michael Werth

The algorithm never sees the pristine image, only the measurement.

$\sigma_R = 0.02$

& sufficient intensity

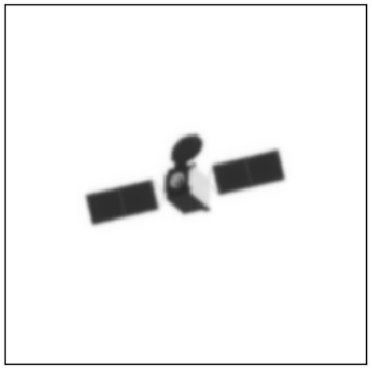
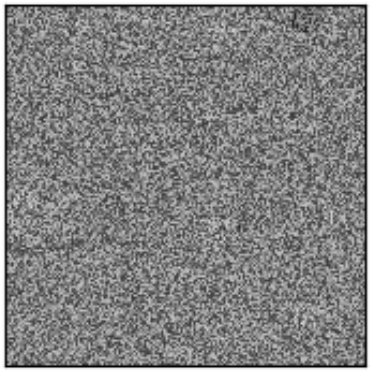
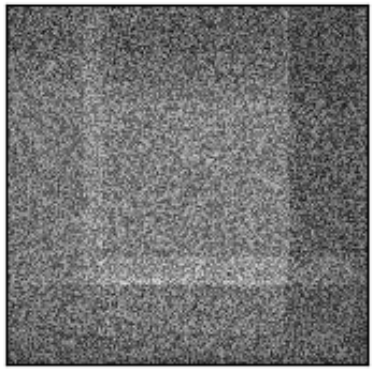
Genetic algorithm



Halving the tile size is too big of a jump

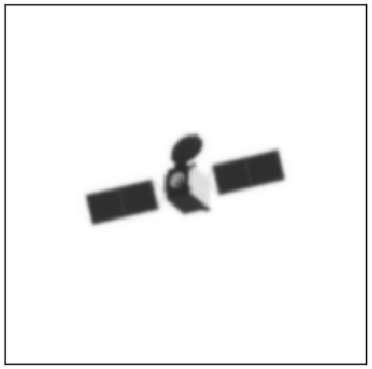
Reducing search space - tile approach

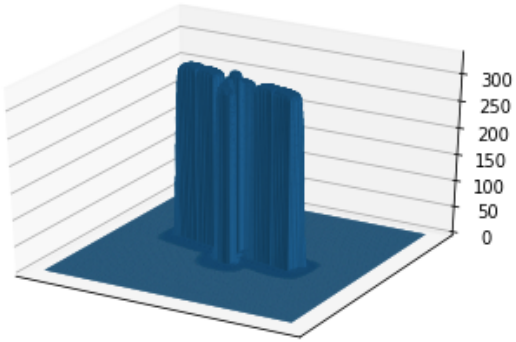
- The search space is so staggeringly huge for the 256x256 image that the algorithm can't even find a 'best' direction.
- Below are results with different combinations of generation, crossover and mutation operations that I devised and obviously failed.

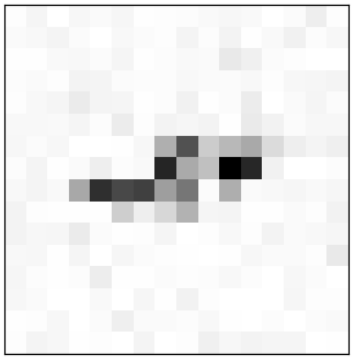

Reducing search space - tile approach

- Starting with tiles actually produces results.
- For C, the intensity of each tile in each canvas (solution) was taken randomly from a Gaussian distribution, the fitness function included separating the distribution of intensities among high and low intensity regions.
- For D, all the intensities start at zero and they preferentially increase.

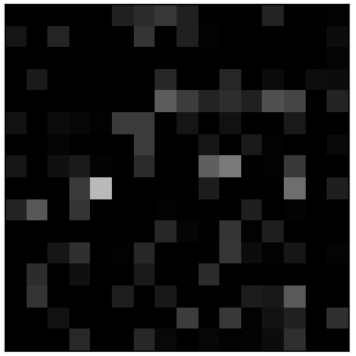




C

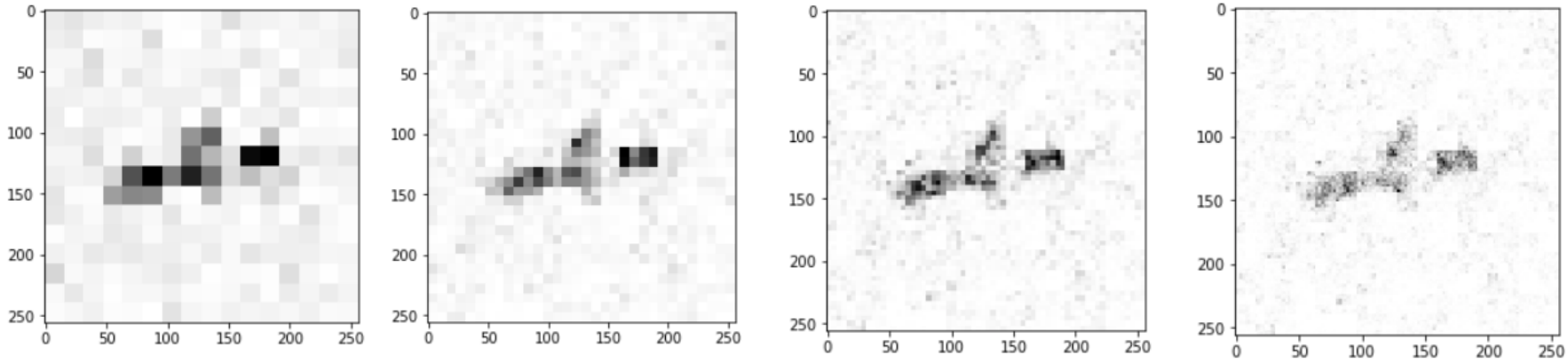


D



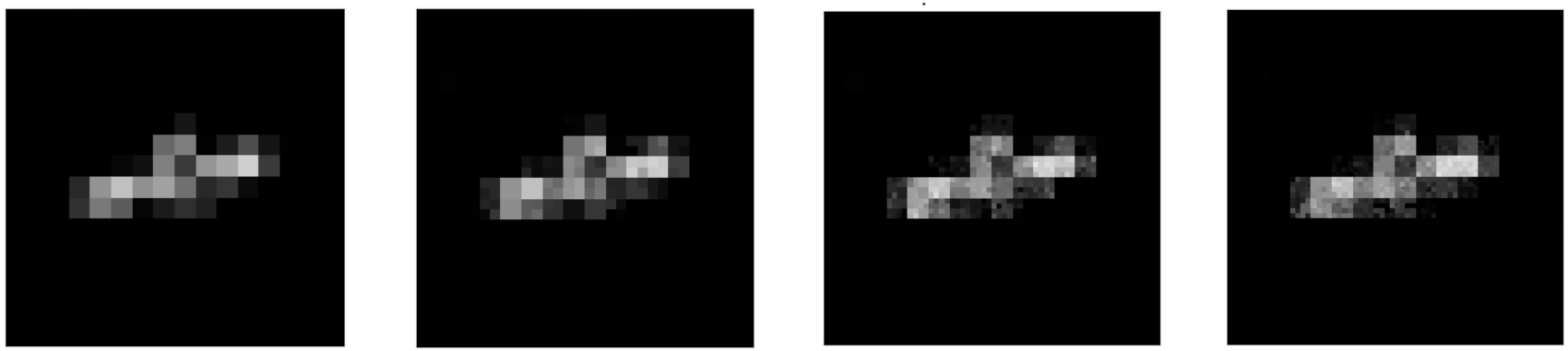
Reducing search space - tile approach

- Results using method C, halving the tile side after a given number of generations.



Reducing search space - tile approach

- Results using method C with halving of the tile side, but removing tiles with no intensity in the input from the search space (this is fair).
- Results don’t look that great after the 16x16 because the jump in the dimension of the search space is too large.
- Conceptually easy fix, but I will have to rewrite a lot of the code.



Genetic algorithm

Generate original population of *n* solutions

```
def generate_orig_population(N, Nindividuals, tiles_per_side, target):
    population = []
    for i in range(Nindividuals):
        population.append(generate_individual(N, tiles_per_side, target))

    return population
```

```
def generate_individual(N, tiles_per_side, target=None):
    genetic_code = []
    if target is None:
        for i in range(tiles_per_side**2):
            genetic_code.append(abs(np.random.normal(loc=0.0, scale=1.0)))

    else:
        tile_indices = generate_tile_indices(N, tiles_per_side)
        for i in range(len(tile_indices)):
            (x1, x2), (y1, y2) = tile_indices[i]
            m = np.mean(target[x1:x2, y1:y2])
            gene = m + np.random.normal(loc=0.0, scale=1.0)
            gene = gene if gene >= 0 else 0
            genetic_code.append(m + np.random.normal(loc=0.0, scale=1.0))

    return genetic_code
```

Essentially take the mean intensity in the analogous tile of the target image and add/subtract intensity based on a Gaussian distribution; repeat for as many solutions;

I apologize if my coding skills hurt your eyes and offends your sensibility! ☹

Genetic algorithm

Pick top *m* solutions

Traditional mean squared error works well for large tiles.

Can use any turbulence simulator

SSIM works better for finer resolution.

The SSIM index is calculated on various windows of an image. The measure between two windows x and y of common size $N \times N$ is:[4]

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

with:

- μ_x the average of x;
- μ_y the average of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables to stabilize the division with weak denominator;
- L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel} - 1$);
- $k_1 = 0.01$ and $k_2 = 0.03$ by default.

```
def pick_top_i(N, tiles_per_side, population, Ntop, Uout, method='mse'):
    mses = []
    entropies = []
    free_energies = []

    if method == 'mse':
        for genetic_code in population:
            canvas = embody(N, tiles_per_side, genetic_code=genetic_code)
            Cout = abs(ft_sh_phase_screen(canvas, phz_params_dict=phz_params_dict,
                                          genetic_code=None))

            mse = sum(sum((Cout-Uout)**2)) / N**2
            free_energies.append(mse)

    if method == 'ssim':
        for genetic_code in population:
            canvas = embody(N, tiles_per_side, genetic_code=genetic_code)
            Cout = abs(ft_sh_phase_screen(canvas, phz_params_dict=phz_params_dict,
                                          genetic_code=None))

            ssim = 1 / compare_ssim(Cout, Uout)
            free_energies.append(ssim)

    if len(population) != len(free_energies):
        return None

    topIndices = np.argpartition(np.array(free_energies), Ntop)[:Ntop]
    topInds = []
    topMses = []
    for ti in topIndices:
        topInds.append(population[ti])
        topMses.append(free_energies[ti])

    return topInds, topMses
```

Male elephant seal about 5% Elitism

☹ ☹

Genetic algorithm

	Generate new population of *n* solutions based on *m*	

But how many genes they pass on to the child depends on how fit they are.

In biological reproduction, the child gets strictly 50% of DNA bases from each parent.

The population of solutions converges more rapidly but at the cost of being brittle to changes in the environment.

The environment does not change in this particular situation, so this is a good approach.

Two top solutions are selected randomly

```
def generate_population_i(Nindividuals, topInds, topLses):
    population = []
    for i in range(Nindividuals):
        p1, p2 = np.random.randint(len(topInds), size=(2))
        q1 = int(topLses[p1] / (topLses[p1] + topLses[p2]) * len(topInds[p1]))
        q2 = len(topInds[p1]) - q1

        genesIdx = range(len(topInds[p1]))
        s1 = set(np.random.choice(genesIdx, q1, replace=False))
        s2 = set(genesIdx) - s1

        newInd = [0 for x in range(len(topInds[p1]))]
        for gidx in s1:
            newInd[gidx] = topInds[p1][gidx]
        for gidx in s2:
            newInd[gidx] = topInds[p2][gidx]

        population.append(newInd)

    return population
```

Which genes they pass is also random

Generate as many individuals as required

Genetic algorithm

Induce mutations

Mutations are induced in every solution; how many tiles are affected and how severely are parameters.

Because the elitism is high, most of the change in the solutions and hence the selected improvements come from the mutations.

The correlation mutation changes the intensity in the direction of the mean intensity of its neighbors.

```
def mutate_i(population, tiles_per_side, mutation_rate=0.1, area_affected=1,
                                                    min_intensity=0, method='naive'):
    if method == 'naive':
        for ind in population:
            cummulative_area = 0
            while cummulative_area < area_affected:
                gidx = np.random.randint(len(ind))
                if min_intensity > ind[gidx]:
                    continue
                else:
                    rnd = np.random.randint(2)
                    mutation = np.random.normal(loc=0.0, scale=1.0) * 100 * mutation_rate
                    ind[gidx] = ind[gidx] + mutation if 0 == rnd else ind[gidx] - mutation
                    cummulative_area += 1
        return population

    elif method == 'corr':
        for ind in population:
            cummulative_area = 0
            while cummulative_area < area_affected:
                gidx = np.random.randint(len(ind))
                if min_intensity > ind[gidx]:
                    continue
                else:
                    pm = plus_or_minus(ind, gidx, tiles_per_side)
                    mutation = np.random.normal(loc=0.0, scale=1.0) * 100 * mutation_rate
                    ind[gidx] = ind[gidx] + mutation * pm
                    cummulative_area += 1

        return population
```

The naïve mutation changes the intensity of random tiles

Halving size of the tile is too big of a step

Need a better way…

Conclusion

- We developed machinery to quantify the size-space complexity of the propagation of light in turbulent media and used it to guide the development of artificial intelligence methodology.

- We developed the components of a genetic algorithm with the purpose of generating images that, when passed through a turbulence routine, reproduce the raw image arbitrarily well.

Future work

- We will train the mathematical graphs shown before to remove turbulence from images using supervised learning. We call them ‘focusing nets.’

- We will continue developing our genetic algorithm so that they reproduce raw images arbitrarily well in a finite amount of time.

Thanks much

- Images provided by Michael Werth (Boeing)
- Funding came from the Faculty Summer Fellowship Program
- All my knowledge about turbulence came from or through Dr. Rao
- Logistics managed by the Directed Energy Directorate Maui Surveillance Branch
- Stimulant conversation with people in 550 and IfA.

Question space

https://cdn.theculturetrip.com/wp-content/uploads/2016/02/3186631988_fedd4bce36_b.jpg