INFO 6205
Program Structure and Algorithms

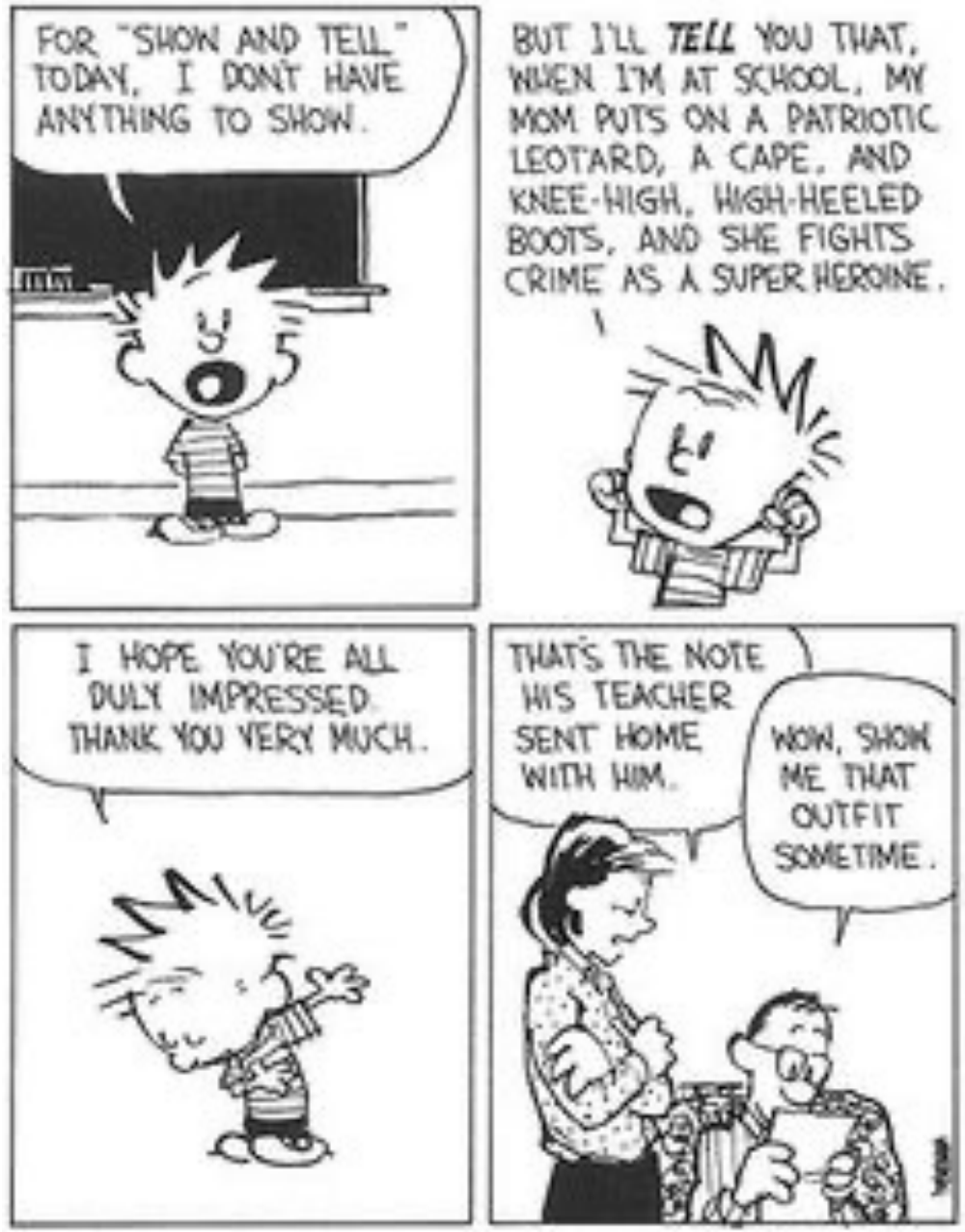
Nik Bear Brown

Research Projects

Topics

- Show and tell
- Research progress examples


CcDdEeFfGgHhIiJjKkLlMmNnOoPpQq
Show + Tell
Thank you, Nelson. I look forward to seeing it again next week.
quickmeme.com


FOR "SHOW AND TELL" TODAY, I DON'T HAVE ANYTHING TO SHOW.
BUT I'LL *TELL* YOU THAT, WHEN I'M AT SCHOOL, MY MOM PUTS ON A PATRIOTIC LEOTARD, A CAPE, AND KNEE-HIGH, HIGH-HEELED BOOTS, AND SHE FIGHTS CRIME AS A SUPER HEROINE.
I HOPE YOU'RE ALL DULY IMPRESSED. THANK YOU VERY MUCH.
THAT'S THE NOTE HIS TEACHER SENT HOME WITH HIM.
WOW, SHOW ME THAT OUTFIT SOMETIME.

“In god we trust.
All others must
bring data.”

– W. Edwards Deming

progress report

a) An account of how much work has been done on something and what still needs to be done
b) An assessment that conveys details such as what sub-goals have been accomplished, what problems have been encountered, and whether the project is expected to be completed on time.

- http://www.businessdictionary.com/definition/progress-report.html

Finding the Most Beautiful Path

Got the idea from:

<http://www.technologyreview.com/view/528836/forget-the-shortest-route-across-a-city-new-algorithm-finds-the-most-beautiful>

I thought that I might write python code that found the shortest path and factor beauty into account.

Approach ::

** Get the graph from (openstreetmap.org) as an html file and parse it into a graph format (a dictionary of dictionaries) .

** Have a dictionary of beauties for some node provided by the user (the smaller the number, the higher the beauty) .

Approach continued...

** The code then adds 100 to the edges that link the nodes the have no beauty as input.

** It then add the beauty given by the user to the edges the come from the node that has a user inputted beauty.

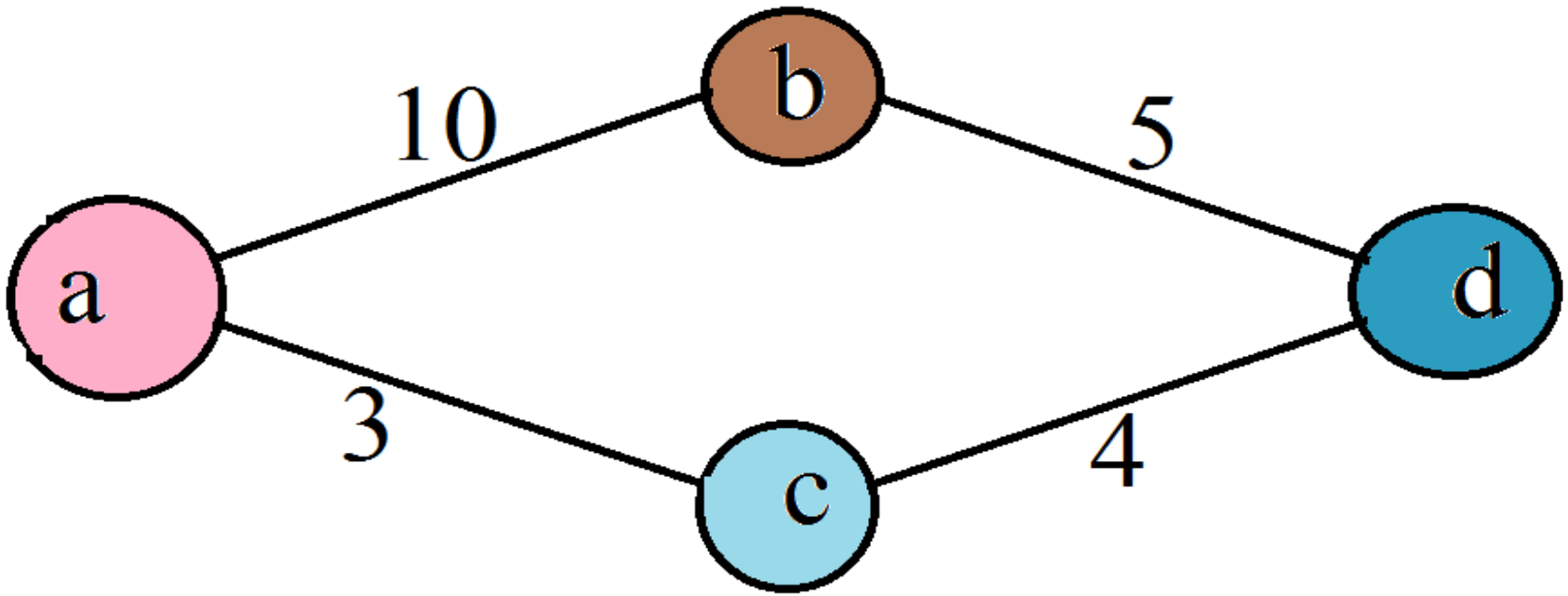
Getting a path:

After the graph is altered as shown above,

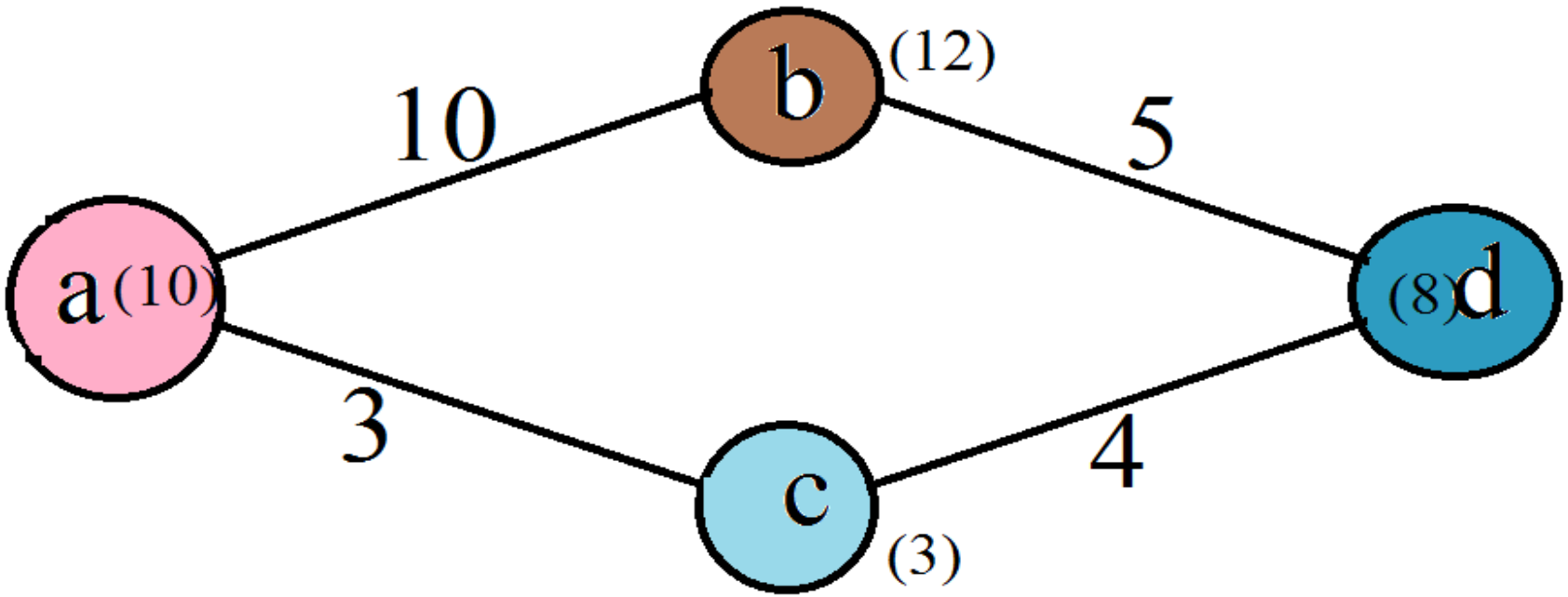
Dijkstra is run on the code and a path is found

to check that it runs and give the best path I ran the code on some user inputed graphs.

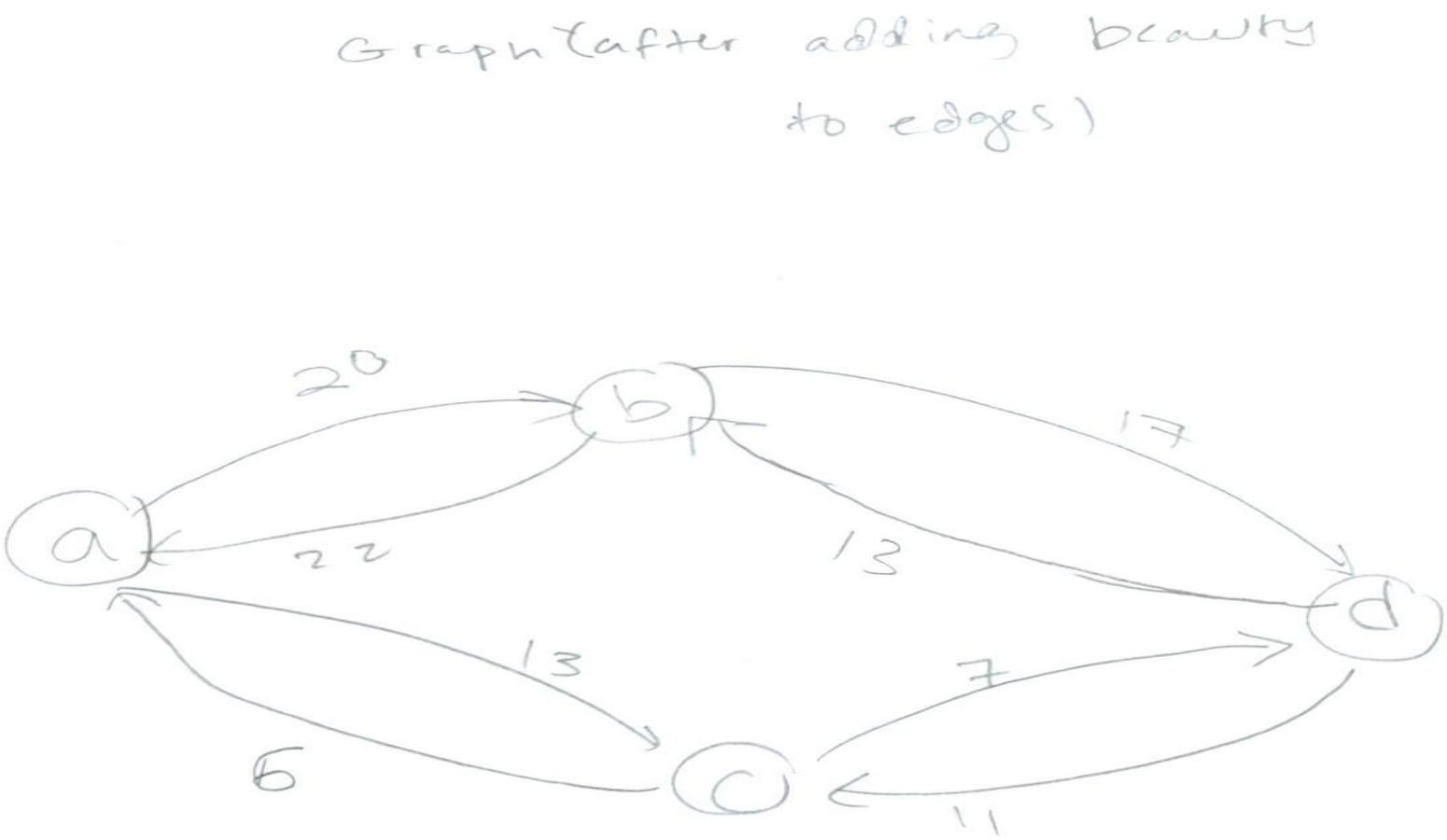
Example:: (original graph)



Graph with user inputted beauty ::



Graph with added beauty ::



Example in code

Enter Graph - with distances "example: {"a":{"b": 12.00}, "b": {"a": 11.99}} : {'a':{'b': 10, 'c': 3} , 'b': {'a':10,'d':5}, 'c': {'a':3, 'd':4}, 'd': {'b': 5, 'c': 4}}
('Graph', "{'a': {'c': 3, 'b': 10}, 'c': {'a': 3, 'd': 4}, 'b': {'a': 10, 'd': 5}, 'd': {'c': 4, 'b': 5}}")
Enter Graph - with beauty "example: {"a":1.00(very beautiful), "b":x(not beautiful) x < 100.00}" : {'a': 10.00, 'b': 12.00, 'c': 3.00, 'd': 8.00}
('Graph Beauty ', "{'a': 10.0, 'c': 3.0, 'b': 12.0, 'd': 8.0}")
Enter "StartNode": 'a'
('StartNode', 'a')
Enter "EndNode": 'd'
('EndNode', 'd')
{'a': {'c': 13.0, 'b': 20.0}, 'c': {'a': 6.0, 'd': 7.0}, 'b': {'a': 22.0, 'd': 17.0}, 'd': {'c': 12.0, 'b': 13.0}}
dist from: a:{'a': 0, 'c': 13.0, 'b': 20.0, 'd': 20.0}
path : {'c': 'a', 'b': 'a', 'd': 'c'}
shortest_path: ['a', 'c', 'd']
shortest_path_distance: 20.0 meters

Enter Graph - with distances "example: {"a":{"b": 12.00}, "b": {"a": 11.99}} : {'a':{'b': 10, 'c': 3} , 'b': {'a':10,'d':5}, 'c': {'a':3, 'd':4}, 'd': {'b': 5, 'c': 4}}
('Graph', "{'a': {'c': 3, 'b': 10}, 'c': {'a': 3, 'd': 4}, 'b': {'a': 10, 'd': 5}, 'd': {'c': 4, 'b': 5}}")
Enter Graph - with beauty "example: {"a":1.00(very beautiful), "b":x(not beautiful) x < 100.00}" : {'a': 10.00, 'b': 12.00, 'c': 3.00, 'd': 8.00}
('Graph Beauty ', "{'a': 10.0, 'c': 3.0, 'b': 12.0, 'd': 8.0}")
Enter "StartNode": 'd'
('StartNode', 'd')
Enter "EndNode": 'a'
('EndNode', 'a')
{'a': {'c': 13.0, 'b': 20.0}, 'c': {'a': 6.0, 'd': 7.0}, 'b': {'a': 22.0, 'd': 17.0}, 'd': {'c': 12.0, 'b': 13.0}}
dist from: d:{'a': 18.0, 'c': 12.0, 'b': 13.0, 'd': 0}
path : {'a': 'c', 'c': 'd', 'b': 'd'}
shortest_path: ['d', 'c', 'a']
shortest_path_distance: 18.0 meters

Recommender Systems

Overview

Recommender Systems:

- Predict the ‘rating’ or ‘preference’ that a user would give to an item and provide recommendations to them based on those predictions
- Majority of large-scale commercial and social websites utilize such systems to recommend products to users or people to connect with
- Well-known examples: Amazon.com, Pandora Radio, Twitter
- Sort through large amounts of data in order to identify potential user preferences
- Allows companies and websites provide a more engaging user experience, which helps with customer retention

Basic Approaches

I. Collaborative Filtering

-Constructs model from user's previous activity

-Incorporates behavior from similar users

-Diverse scope of recommendations

-Requires a large amount of existing data on users to be accurate

II. Content-based Filtering

-Keywords are used to describe items, user profile includes items that he/she liked in the past

-Every Item has a profile with specific characteristics – Used to find other, similar items

-Does not require as much user data to start

-Suggestions are limited to content of same type

My Project

*Implementation of **collaborative filtering** on real-world data:*

1. Collecting Preferences

- Users, Items, Ratings
- *Nested Dictionary*

```
# A dictionary of movie critics and their ratings of a small
# set of movies
critics={'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5,
 'Just My Luck': 3.0, 'Superman Returns': 3.5, 'You, Me and Dupree': 2.5,
 'The Night Listener': 3.0},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5,
 'Just My Luck': 1.5, 'Superman Returns': 5.0, 'The Night Listener': 3.0,
 'You, Me and Dupree': 3.5},
'Michael Phillips': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.0,
 'Superman Returns': 3.5, 'The Night Listener': 4.0},
'Claudia Puig': {'Snakes on a Plane': 3.5, 'Just My Luck': 3.0,
 'The Night Listener': 4.5, 'Superman Returns': 4.0,
 'You, Me and Dupree': 2.5},
'Mick LaSalle': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
 'Just My Luck': 2.0, 'Superman Returns': 3.0, 'The Night Listener': 3.0,
 'You, Me and Dupree': 2.0},
'Jack Matthews': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
 'The Night Listener': 3.0, 'Superman Returns': 5.0, 'You, Me and Dupree': 3.5},
'Toby': {'Snakes on a Plane':4.5,'You, Me and Dupree':1.0,'Superman Returns':4.0}}
```

My Project

2. Finding Similar Users

- How similar are users in their tastes? – *similarity score*
- *Euclidean Distance Score*
- Compare everyone against a given user - find the closest matches and group them

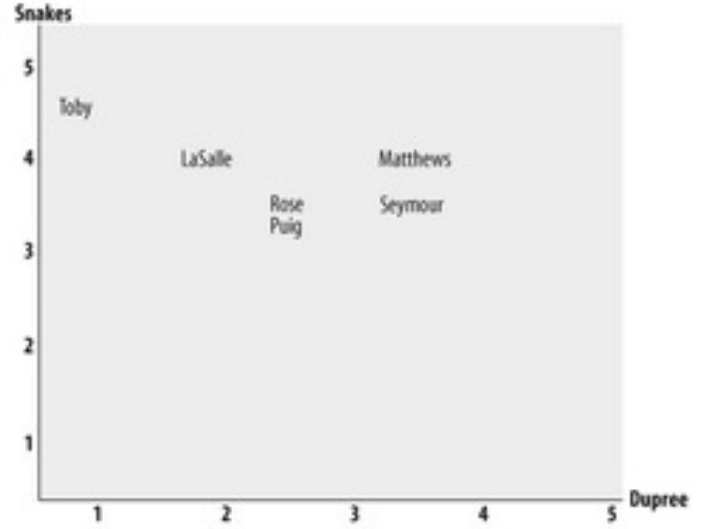


Figure 2-1. People in preference space

My Project

3. Recommending Items

I. *Naïve Approach*: Find person most similar to me and look for a movie he likes that I have not seen.

II. *Instead*: Rank movies via weighted score that ranks other users within my similarity group

- Multiply similarity score of each user by the rating they gave each movie
- Those who are more similar to me will have a greater affect on the overall score of a given movie
- Divide overall score of each movie by the sum of every relevant user's similarity scores
- Results are sorted and returned to me

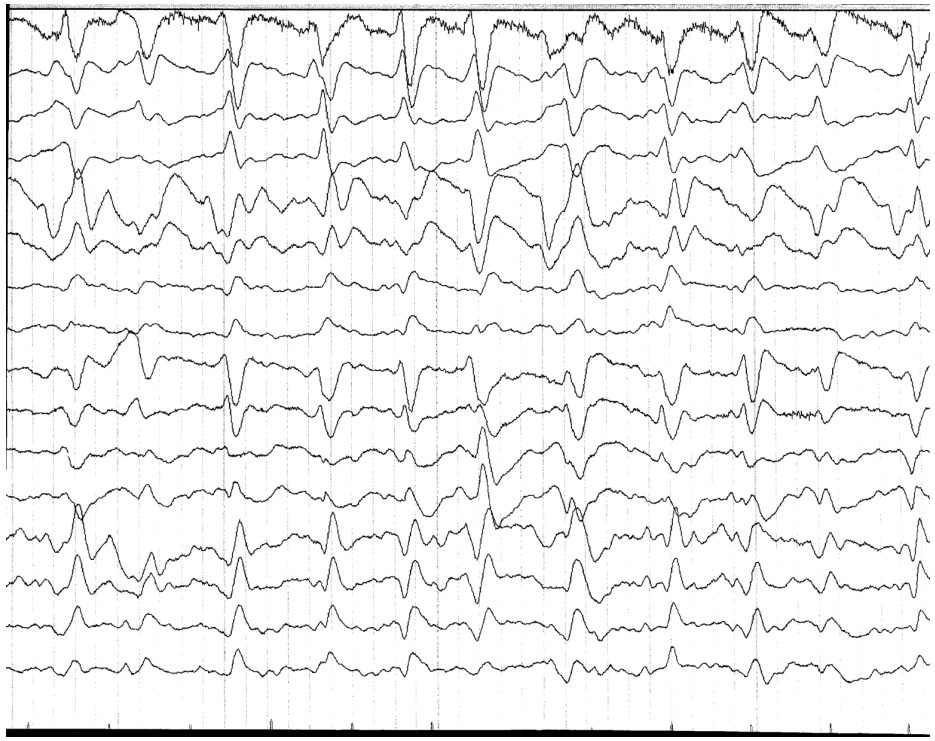
EEG Data Analysis
Aided by Quicksort

Electroencephalography (EEG)

- The recording of electrical activity in the scalp
- Electric potentials emitted by neuron activity

EEG Data

- In the form of sin waves
- Different waves associated with various activities



Challenges with EEG

- High signal to noise ratio
- Data from different sources
- Non-stationary – same activity may generate different signals across sessions

Classifier Algorithms

- Algorithms that automatically estimate the type or class of the data
  - Static-dynamic
  - Generative-Discriminative
  - Regularized

Thesis

- EEG data can efficiently be made easier to process by running quicksort on the data to identify and remove outliers to reduce the amount of noise

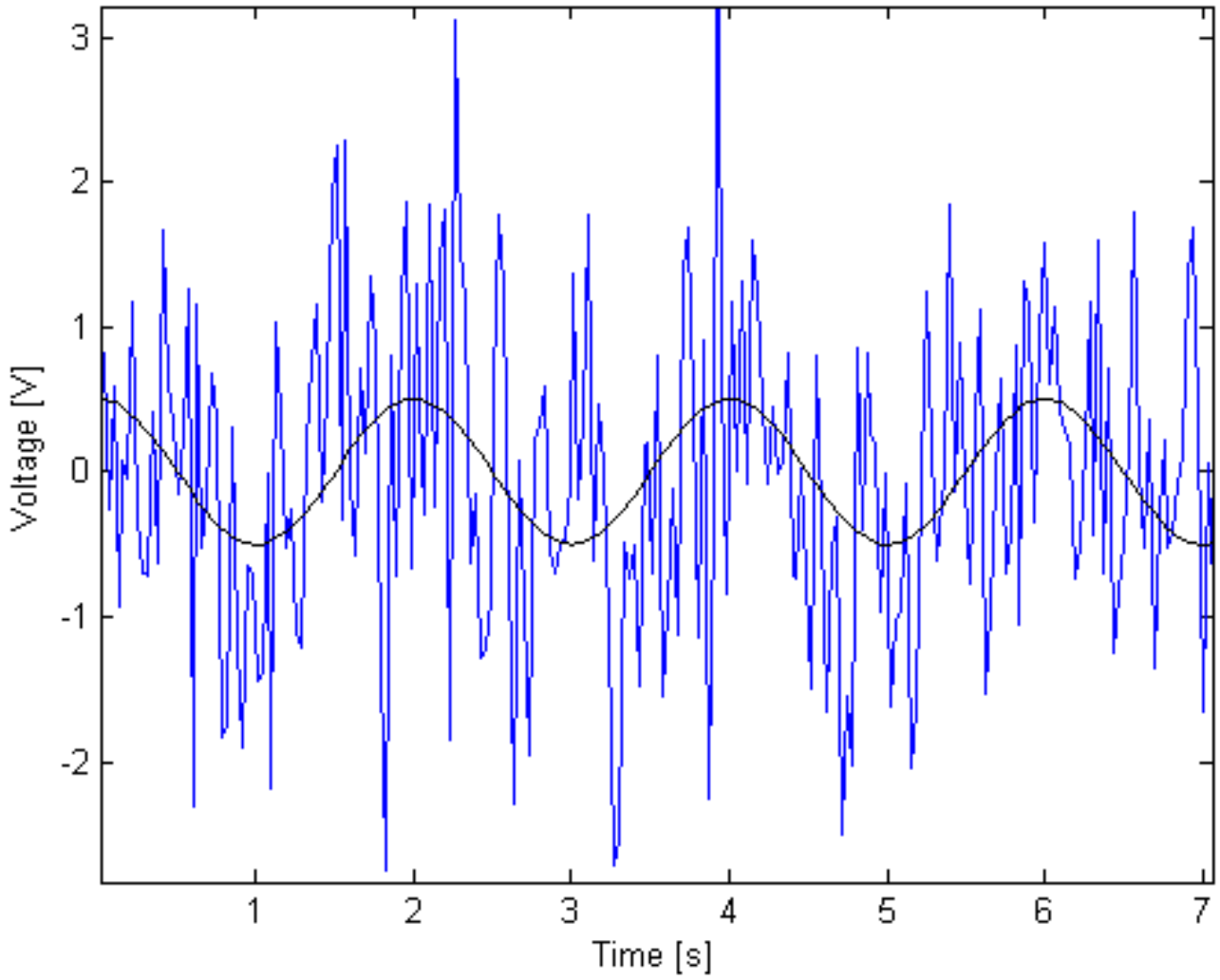
Implementation

- MATLAB – Used for EEG signal processing

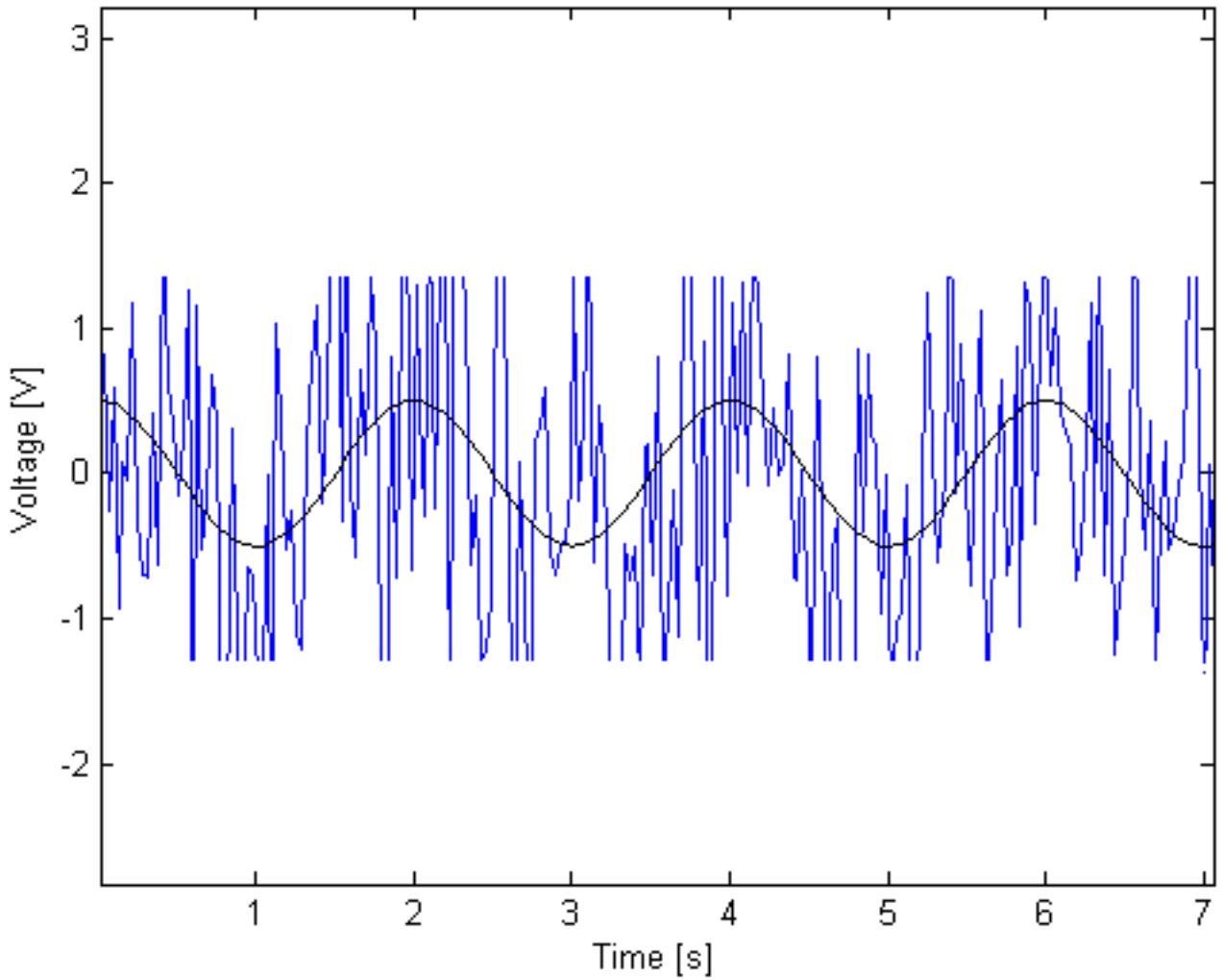
- Run quicksort on data
- Keep all original indices within sin wave
- Remove points above certain bound
- Re order and plot

- Runs in O(n*log(n))

Results



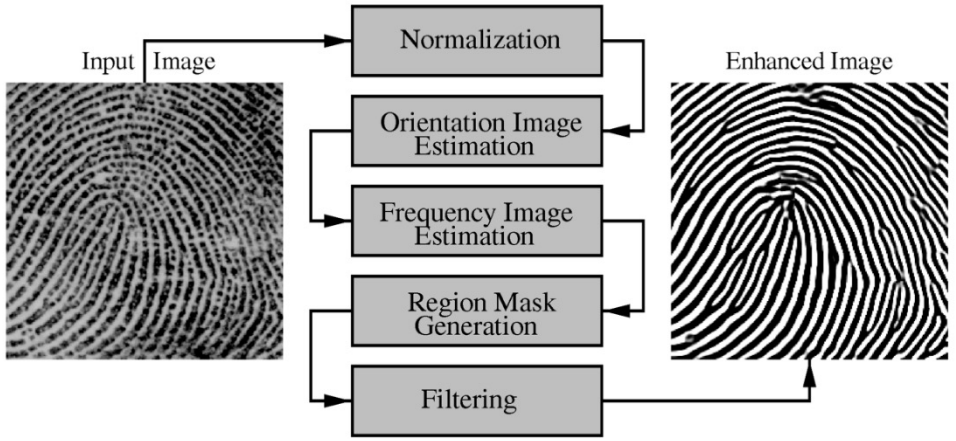
Result



Fingerprint Enhancement

Fingerprint Image Enhancement: Algorithm and Performance Eval.

- This algorithm is based on several steps:
- *Normalization:* Which cleans up the image of a lot of ‘noise’.
- *Orientation Image Estimation:* returns an Orientation image from the Normalized one they are kind of like line indicators of the flow of the lines on a fingerprint
- *Region Mask Estimation:* divides up the normalized image into recoverable and unrecoverable regions.
- *Filtering:* using the Gabor filters it enhances the image even more



Fingerprint Enhancement: Fourier Filtering

- *Fourier Filtering Stage:* The filtering stage produces a directionally smoothed version of the image from which most of the unwanted information ('noise') has been removed, but which still contains the desired information (i.e. the ridge structure and minutiae). The thresholding stage produces the binary, enhanced image.
- Threshold Stage: binarises the directionally filtered image using a local average as the threshold surface.

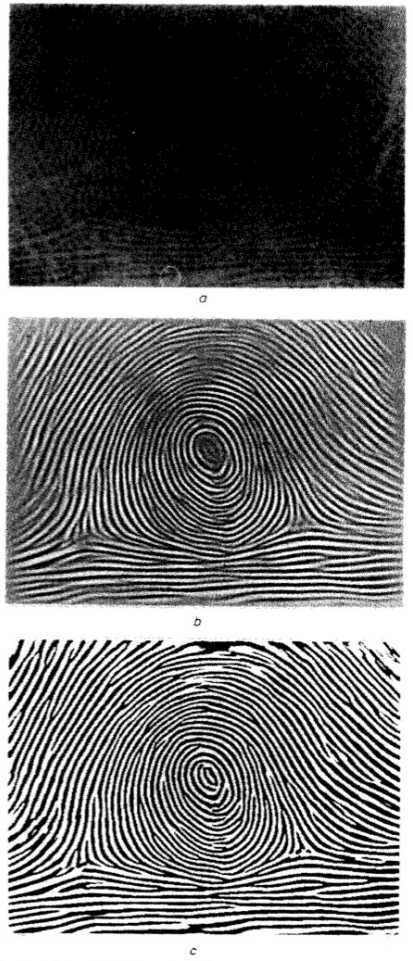


Fig. 4 *Stages of the fingerprint enhancement process*
a Original fingerprint
b After filtering
c After thresholding

Fingerprint Enhancement: Wavelet Transform

- combines the texture unit of global information with the ridge orientation of local information.
- characteristic of hierarchical framework of multi-resolution representation, the wavelet transform is used to decompose the fingerprint image into different spatial=frequency sub-images by checking the approximation region.

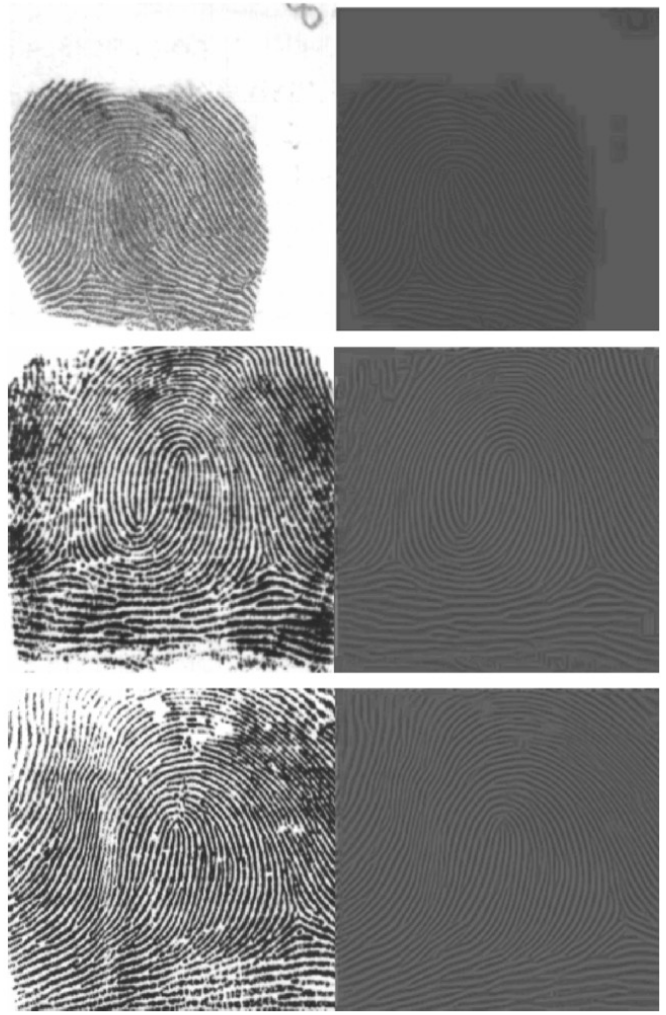
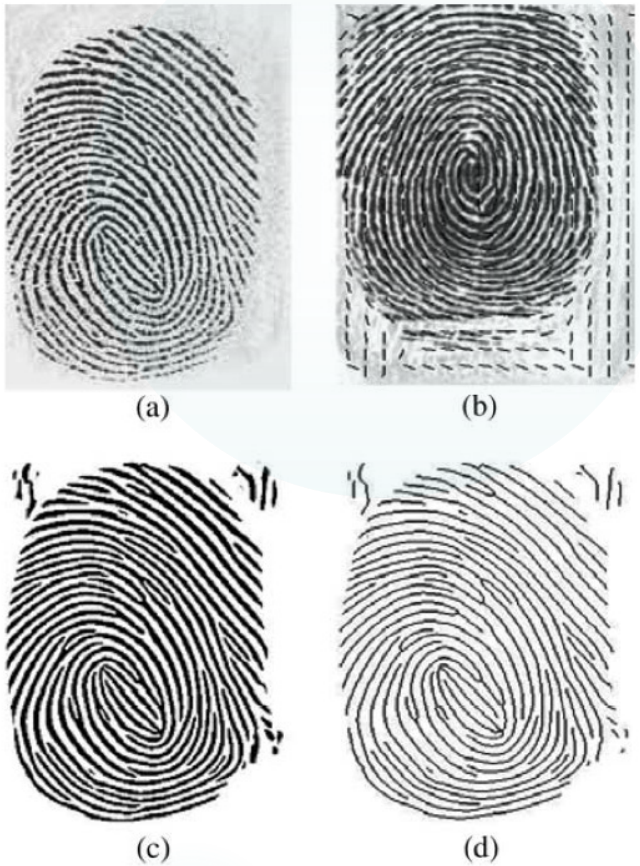


Fig. 10. Some reconditioned images by using the proposed wavelet-based enhancement algorithm.

Image Enhancement & Minutiae matching in Fingerprint verification

- Fingerprint Image Enhancement: It divides up the image into a matrix and finds the color gradients.
- After finding out the darkest to the lightest it follows the darkest line (marking it as the fingerprint) and gives it a direction
- after such procedure it binarises the image into black and white
- then it thins out the image and gives a more accurate and detailed image



Algorithmic Composition

What?

- **Algorithmic composition** – a set of techniques used to create music that depend on algorithms or other formal processes
- Although programmable computers were not invented until the early 20th century, the concept has existed for much longer
  - Computers have made the field of algorithmic composition more robust

A Brief History

- Algorithmic composition dates back to as early as the 15th century
- **Canonic composition –** composition of music based on certain *canons* (rules or laws)
  - Harmonizing
  - Reverb, Delay, and Doubling
  - Inversion

Four Categories

- 1. Mathematical models and stochastic processes
- 2. Knowledge-based systems
- 3. Grammars
- 4. Systems that learn

1. Mathematical Models & Stochastic Processes

- Random & Non-Deterministic
  - Probabilities of events are weighted by composer

- **Stochastic processes**
  - Sequence of integers % 87 = notes on a keyboard
- **Markov chains**
  - Improves upon purely random processes; states and edges
- **Number theory**
  - The Golden Mean

Quick Example

479, 807, 279, 687, 890, 846, 722, 138

44, 24, 18, 78, 20, 63, 26, 51 →

F4, A2, D#2, D#7, F2, C6, B2, C5

2. Knowledge-Based Systems

- Composer specifies set of rules and constraints
  - Rule may specify vocal range limitations, for example
- Constraint-logic programming & constraint-satisfaction techniques to produce and validate results
- Analysis:
  - Subjectivity based on chosen rules
  - Human interaction is not minimized
  - Systems become extremely complex when exceptions to rules are added (i.e. *accidentals* in a key signature)

3. Rule-Based Systems & Grammars

- Similar to Knowledge-Based Systems in that grammars use rules for a composition to follow
- Higher focus on macro-level composition
  - Rhythm
  - Tempo
  - Key Signature
  - Harmonization

4. Artificial Intelligence

- Systems are not given rules or information on style and genre
  - Unsupervised learning
- Composer provides example material (training set) to the machine to analyze
  - Machine then generates a piece with similar qualities to training set

- Analysis:
  - Limits creativity; composer has little input beyond training set
  - Difficult to pick up on certain qualities in training set (i.e. phrasing, tonality)
  - Composer must filter training set to reduce the number of conflicting properties

Task Sorting and Scheduling

Basic Overview

Forgetful person – I want to always know what I should be working on

- Make a “card” for each class or club “task” I need to do
- Sort the tasks based on varying attributes
  - Due Date
  - Hours of work required
- Keep track of subtasks based on completion order

Basic Overview II

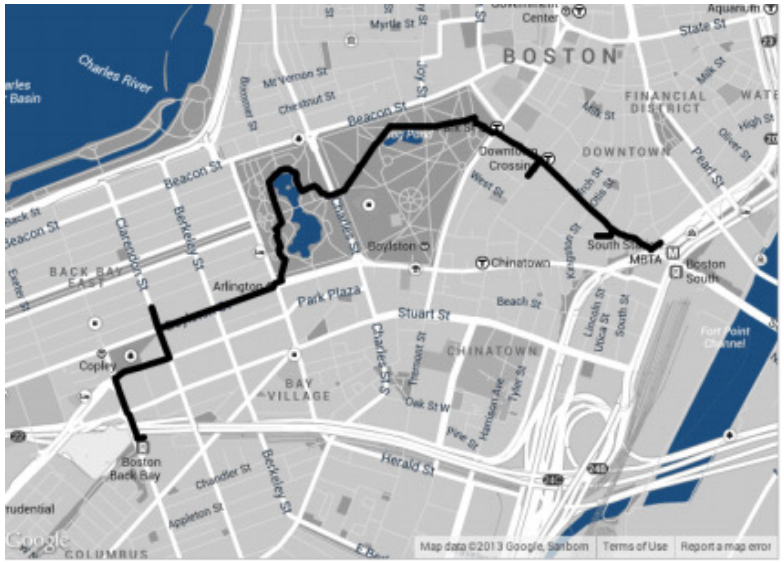
- General output should be a schedule with a list of tasks in the order I should be doing them
- Have both a general stack schedule and a day-by-day schedule
- Update lists when a new task is added

Algorithm Ideas

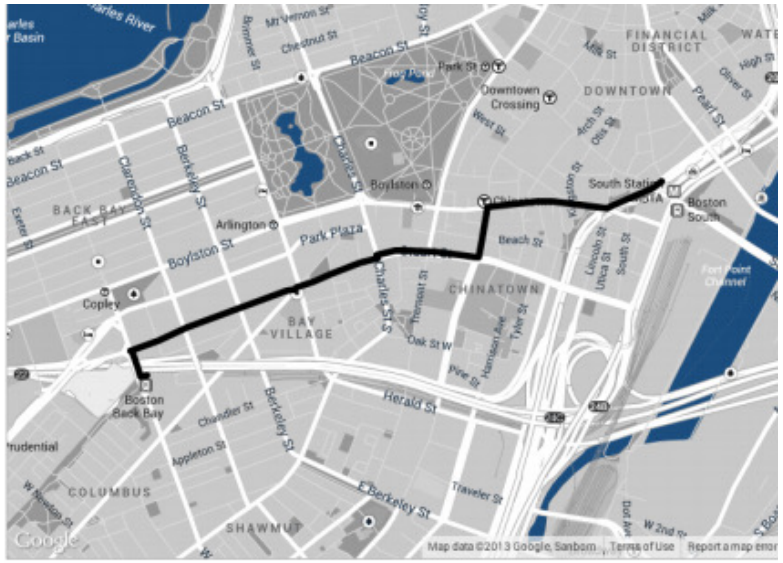
- Dynamic algorithm to cut time
  - Minor updates (name, subtasks) just update task
  - Relevant changes (due date, etc) re-sort entire list
  - Adding/removing tasks can only reschedule tasks on/after that due date

Most Beautiful Path Algorithm

Sample



(b) Flickr Beauty in Boston



(c) Shortest in Boston

Research

- Chang et al. ([2])used a backtracking algorithm to recommend car routes that deviate from a user's familiar/past trajectories.
- Ludwig et al. ([3]) used an adaptive A*-like algorithm to recommend public transport routes that afford both short walks and little waiting times.

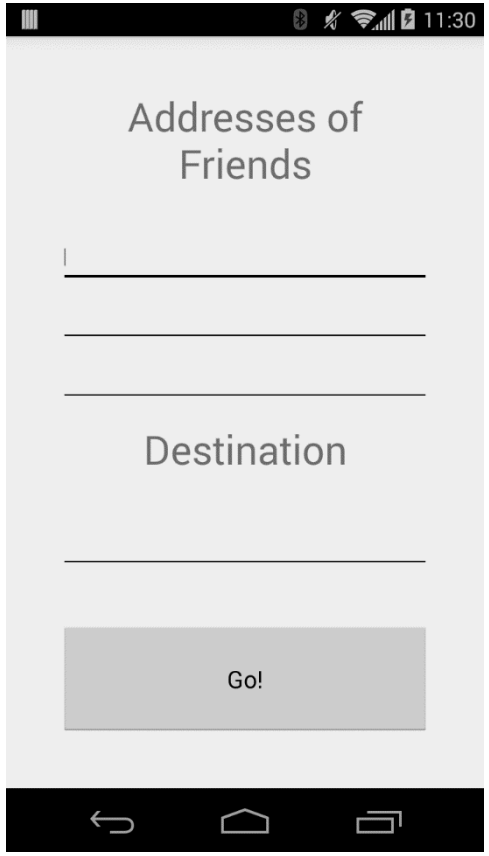
Map of Boston with seven frequently mentioned places

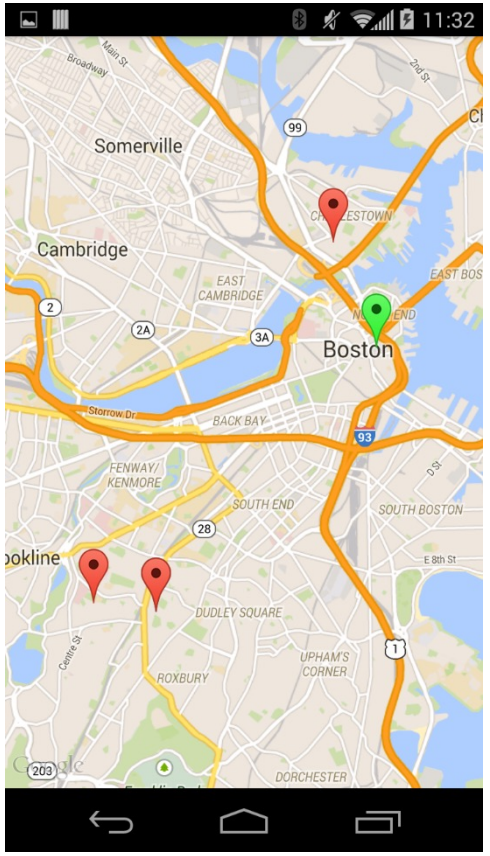
Uber Price Optimization App

What it does

- Takes in multiple addresses
- Pings Uber API to get estimated fares
- Builds graph
- Calculates cheapest route
- Provides navigation

Screenshots of App





Text Auto-Completion

Fuzzy Matching

- Given a word, return suggestions of words that are close
- For example, take “aeek” and return things like “seek” or “peek”

Levenshtein Distance

- Number representing the smallest amount of single-character edits (insertion, deletion, replacement) to change one string to another

Metric Space

- If d(x,y) = 0, then x = y (If the distance between x and y is 0, then x = y)
- d(x,y) = d(y,x) (The distance from x to y is the same as the distance from y to x)
- d(x,y) + d(y,z) >= d(x,z)
  - Triangle Inequality

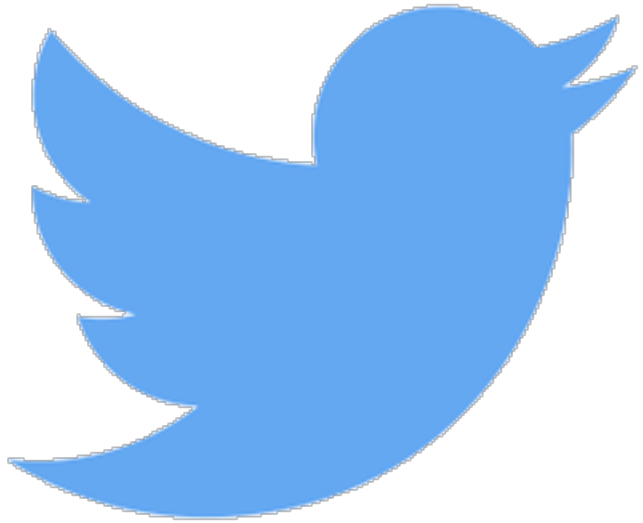
Burkhard-Keller Tree

- Arbitrary word as root node
- Node has zero or more children (subtrees)
- Every subtree is recursively built, such that its children are all the same distance from the root node as the subtree's parent node

Triangle Inequality

- Given 3 strings a b and c
  - (Distance from a->b + Distance from b->c) >= Distance from a->c
- This enables us to create the BK-Tree

Twitter Game Sentiment Analysis

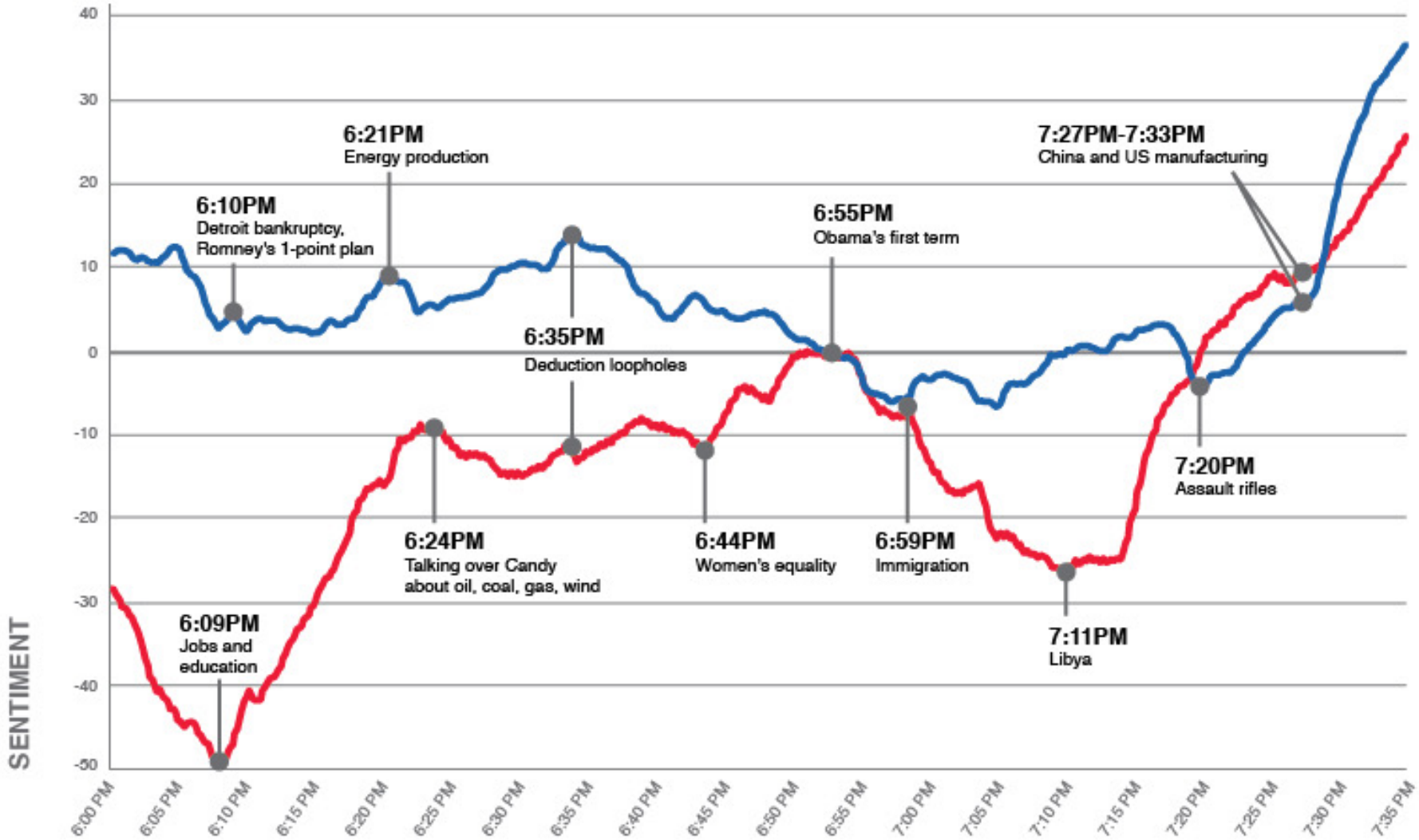


Twitter Game Sentiment Analysis

- Using a large sample of tweets, we hope to ascertain a certain level of positivity/negativity regarding a particular topic in the Twittersphere.
- First, we will strip tweets down to only their “essential” words. Words like “a” “the” “she” will be removed.
- Then, we will compare the “essential” words to a collection of known words that have been ranked on sentiment.
- Ultimately, we’ll be delivering a -1 to 1 scale of overall sentiment on a given topic.

NETBASE 2012 ELECTION MOOD METER

Presidential Debate – October 16, 2012

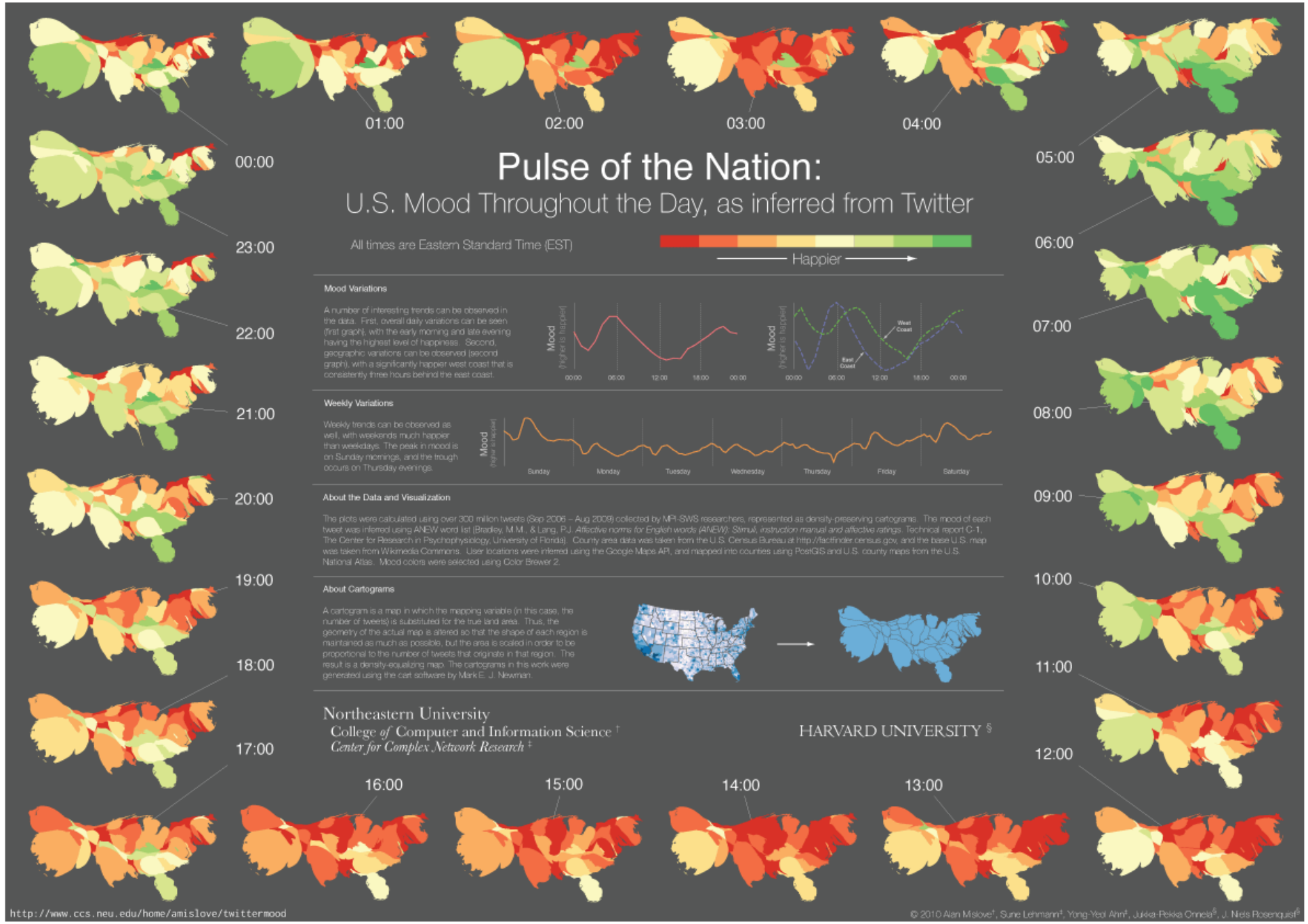


TIME (PST)

Romney Obama

Based on Twitter data. Powered by NetBase.

NETBASE


00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
Pulse of the Nation:
U.S. Mood Throughout the Day, as inferred from Twitter
All times are Eastern Standard Time (EST)
Happier
Mood Variations
A number of interesting trends can be observed in the data. First, overall daily variations can be seen (first graph), with the early morning and late evening having the highest level of happiness. Second, geographic variations can be observed (second graph), with a significantly happier west coast that is consistently three hours behind the east coast.
Mood (higher is happier)
West Coast
East Coast
Weekly Variations
Weekly trends can be observed as well, with weekends much happier than weekdays. The peak in mood is on Sunday mornings, and the trough occurs on Thursday evenings.
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
About the Data and Visualization
The plots were calculated using over 300 million tweets (Sep 2006 – Aug 2009) collected by MPI-SWS researchers, represented as density-preserving cartograms. The mood of each tweet was inferred using ANEW word list (Bradley, M.M., & Lang, P.J. Affective norms for English words (ANEW): Stimuli, instruction manual and affective ratings. Technical report C-1, The Center for Research in Psychophysiology, University of Florida). County area data was taken from the U.S. Census Bureau at http://factfinder.census.gov, and the base U.S. map was taken from Wikimedia Commons. User locations were inferred using the Google Maps API, and mapped into counties using PostGIS and U.S. county maps from the U.S. National Atlas. Mood colors were selected using Color Brewer 2.
About Cartograms
A cartogram is a map in which the mapping variable (in this case, the number of tweets) is substituted for the true land area. Thus, the geometry of the actual map is altered so that the shape of each region is maintained as much as possible, but the area is scaled in order to be proportional to the number of tweets that originate in that region. The result is a density-equalizing map. The cartograms in this work were generated using the cart software by Mark E. J. Newman.
Northeastern University
College of Computer and Information Science †
Center for Complex Network Research ‡
HARVARD UNIVERSITY §
http://www.ccs.neu.edu/home/amislove/twittermood
© 2010 Alan Mislove†, Sune Lehmann‡, Yong-Yeol Ahn‡, Juuso-Pekka Onnela§, J. Niels Rosenquist§

Boston Bar Crawl

Boston Bar Crawl

- Boston Bar Crawl Application
- Allows the user to search for a custom crawl
- Some parameters can be...
  - type of bars
  - importance of stopping at specific bars
  - time for crawl (12 o’clock curfew?)

Algorithm Applications

- Dijkstra's algorithm
- Sorting Algorithms
- Knapsack
- Interval

MOBA character select algorithm

MOBA character select algorithm

- MOBA, Popular and growing
- 5v5, PvP
- Different characters, Character select plays an important role
- Draft 1-2-2-2-2-1 most popular format


Your Team
13
Enemy Team
SKY TEMPLE
Psione
KERRIGAN
Your Team
PICKING...
Psione
KERRIGAN
Rakcet
MURKY
Spyrian
E.T.C.
Zoevia
JAINA
Clooken
PICKING...
Aldrexus
MALFURION
Trikslyr
ILLIDAN
Troysent
AZMODAN
Katealyst
UU
Kibbelz
LOCK IN
Skins
Mounts
Hero Info
Press Enter to chat
0

Goal

- Create the best possible algorithm to suggest picks to players
- Accurately simulate live player drafts using statistics

Reasons

- Game design major
- Helps simulate drafts for balance, an important part of design
- Can also be used to create tables in order to asses overall viability of certain characters
- Professional scene, useful for analysis

Elements to consider

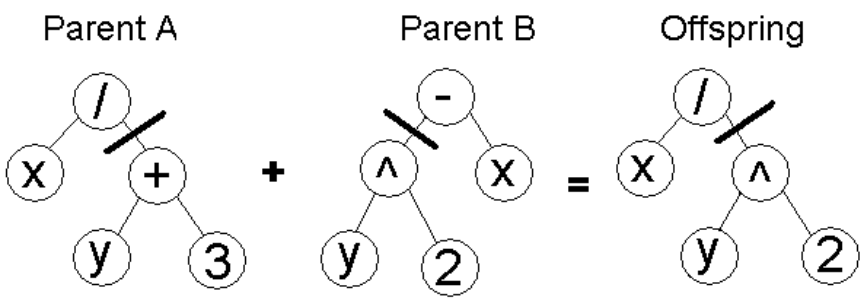
- Global character win rate (tier)
- Personal character win rate (experience)
- Win rate of character vs. characters on the other team (counter-pick)
- Role coverage (team composition)

Genetic Algorithms

Programming pacman Ai with Genetic algorithms

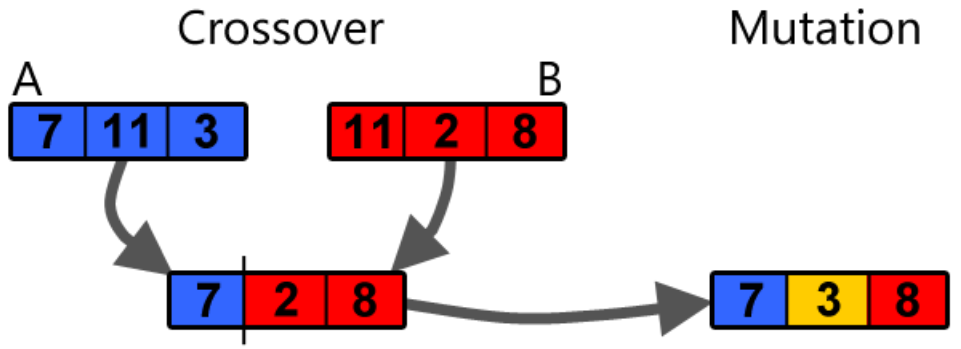
What are Genetic Algorithms?

- Genetic algorithm – is a search heuristic that mimics the process of natural selection.
- Genetic algorithms belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.



Crossover and Mutation in GA

- Crossover -Is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next
- Inheritance - Genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next



Chosen model For Pacman Learning

Basic choices

- Python as a programming language
- Libraries(pygame)

Functions

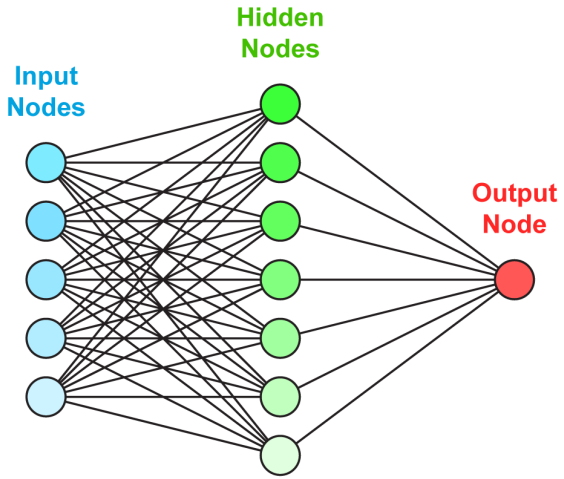
- Basic building blocks for genetic algorithms
- Choosing the minimal set of functions to be able to complete the task

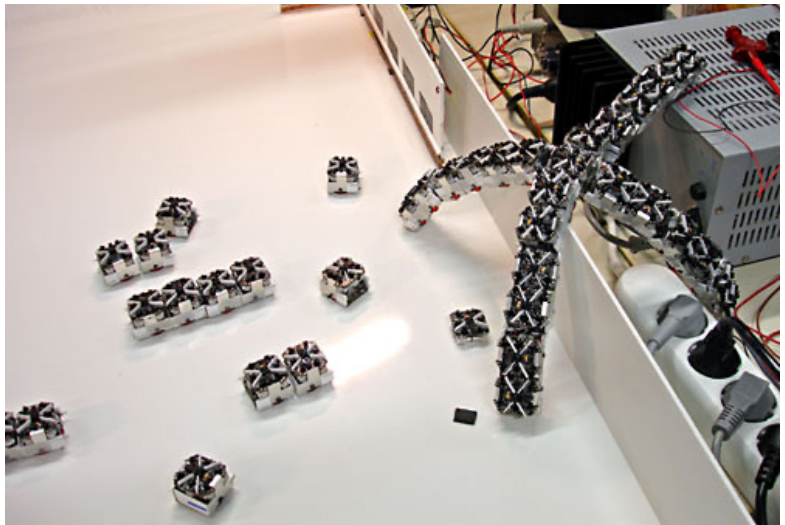
Parse model

- Structure of a tree
- Reasoning for this choice

Things That I Learned

- Programming GA forces thinking in terms of which pieces of functionality should be implemented by hand and which ones should be left out to the GA itself to learn.
- Thinking how the GA will learn certain types of behavior based on your function implementations is generally wrong.
- A range of topics like neural networks and swarm intelligence.





Fast Fourier Transform

My Project:

- Explore the application of the Fast Fourier Transform as a device for the live composition and performance of electronic music.

What is the FFT?

- Signal Processing and Data Analysis
- FAST $\boldsymbol{O[N \log N]}$ algorithm.
- Used to compute the Discrete Fourier Transform (DFT) which is an $\boldsymbol{O[N]^2}$ computation.

FFT to compute DFT

- FFT is important in many fields.
- Python provides many FFTtools and libs.
- NumPy & SciPy have the FFTPACK library.
- Can be found in numpy.fft and scipy.fftpack.

```
>>> import numpy as np
>>> def my_DFT(x):
        '''Computes the DFT of the 1D array x'''
        x = np.asarray(x, dtype=float)
        N = x.shape[0]
        n = np.arange(N)
        k = n.reshape((N, 1))
        M = np.exp(-2j * np.pi * k * n / N)
        return np.dot(M, x)

>>> np.allclose(my_DFT(x), np.fft.fft(x))
True
```

Symmetries

- Divide and Conquer.
- Cooley & Tukey used this approach in deriving the FFT.
- Showed that computing the DFT can happen in two smaller parts.

```
>>> def my_FFT(x):
        ''' A recursive function of the 1D Cooley-Tukey FFT algorithm '''
        x = np.asarray(x, dtype=float)
        N = x.shape[0]
        if N % 2 > 0:
                raise ValueError("size of x must be a power of 2")
        elif N <= 32:
                return my_DFT(x)
        else:
                even_X = FFT(x[::2])
                odd_X = FFT(x[1::2])
                factor = np.exp(-2j * np.pi * np.arange(N) / N)
                return np.concatenate([even_X + factor[:N /2] * odd_X, even_X + factor[N / 2:] * odd_X])


>>> np.allclose(FFT(x), np.fft.fft(x))
True
```

Most Beautiful Path

A study of the “Most Beautiful Path” and its associated algorithms and technologies.

A comparison of this approach and other “X Path” modalities.

An attempt to view abstraction techniques for “X Path” modalities.

Sub-Section 1 : Most Beautiful Path

What is beauty?

- Beauty
  - Green spaces
  - Victorian houses
- Ugly
  - Trash
  - Broken Windows

- In 1967
  - Peterson proposed a quantitative analysis of perceptions of neighborhood appearance
  - Found that **beauty and safety are approximately collinear**.

-A model of preference: *Quantitative analysis of the perception of the visual appearance of residential neighborhoods*. George L. Peterson

Algorithm

- Our goal is to suggest users a short and pleasant path between their current location s and destination d. We meet this goal in four steps:

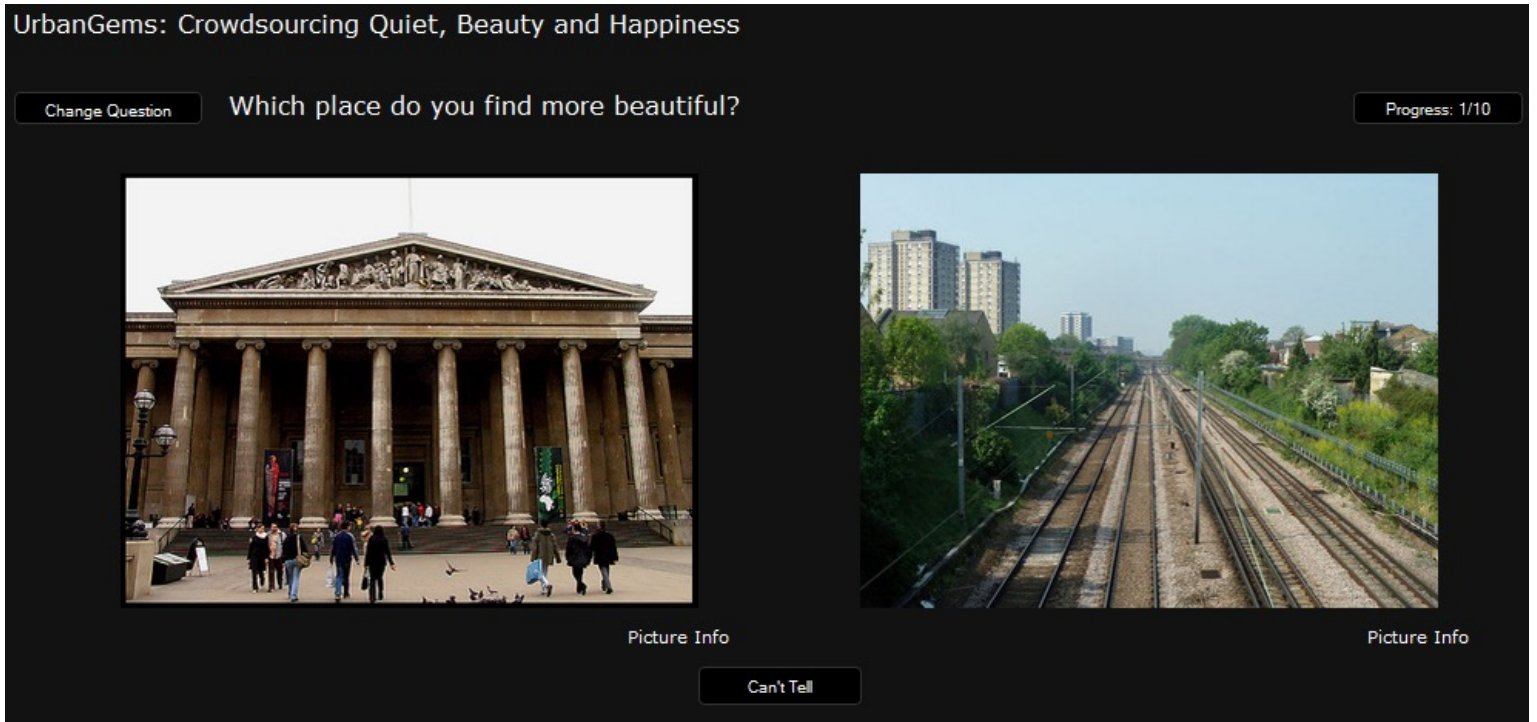
1. Build a graph whose nodes are all locations in the city under study.
2. Crowdsource people's perceptions of those locations along three dimensions: beautiful, quiet, and happy.
3. Assign scores to locations along each of the three dimensions.
4. Run Eppstein’s Algorithm to determine a list of all paths.
5. Select the path between nodes s and d that strikes the right balance between being short and being pleasant.

Process/Procedure

- Previous research has established that 200m tends to be the threshold of walkable distance in urban areas
- Each node is a location and links to its eight geographic neighbors
- Rely on the data gathered from a crowdsourcing web site to assess how locations are perceived to be beautiful, quiet and make people happy
- Data is gathered through the use of a “Game” where users compare two quality similar images and rate one more x than the other

At each game round, users should either click on one of the two scenes or opt for Can't Tell

The user is asked to guess the percentage of other people who shared their views, scoring points for correct guesses.



Eppstein's algorithm

- The algorithm with the lowest asymptotical *worst-case* time complexity that enumerates, in order of increasing length, the *K* shortest paths between two given nodes, *s* and *t*, in a digraph $G = (V,E)$ with *n* nodes and *m* arcs

- Eppstein complexity is O(m+ n log n + k log k)

- Algorithm (Breadth-First Search):

push root.0 onto heap H

for i = 1 to k

pop node r.c1 from top of H

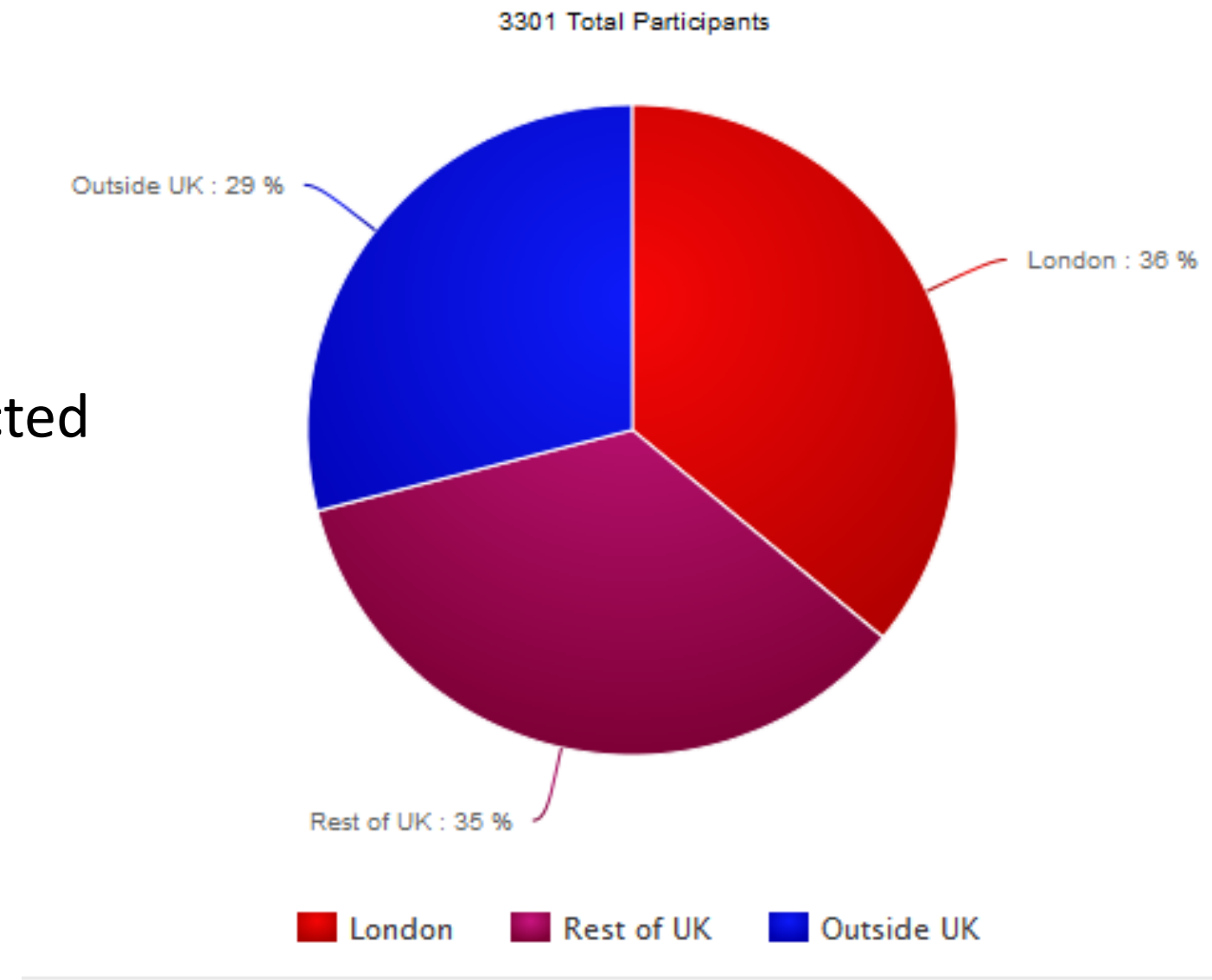
print cost c1 + 55

for j = each edge from r s with cost c2

push node s.(c1+c2) onto H

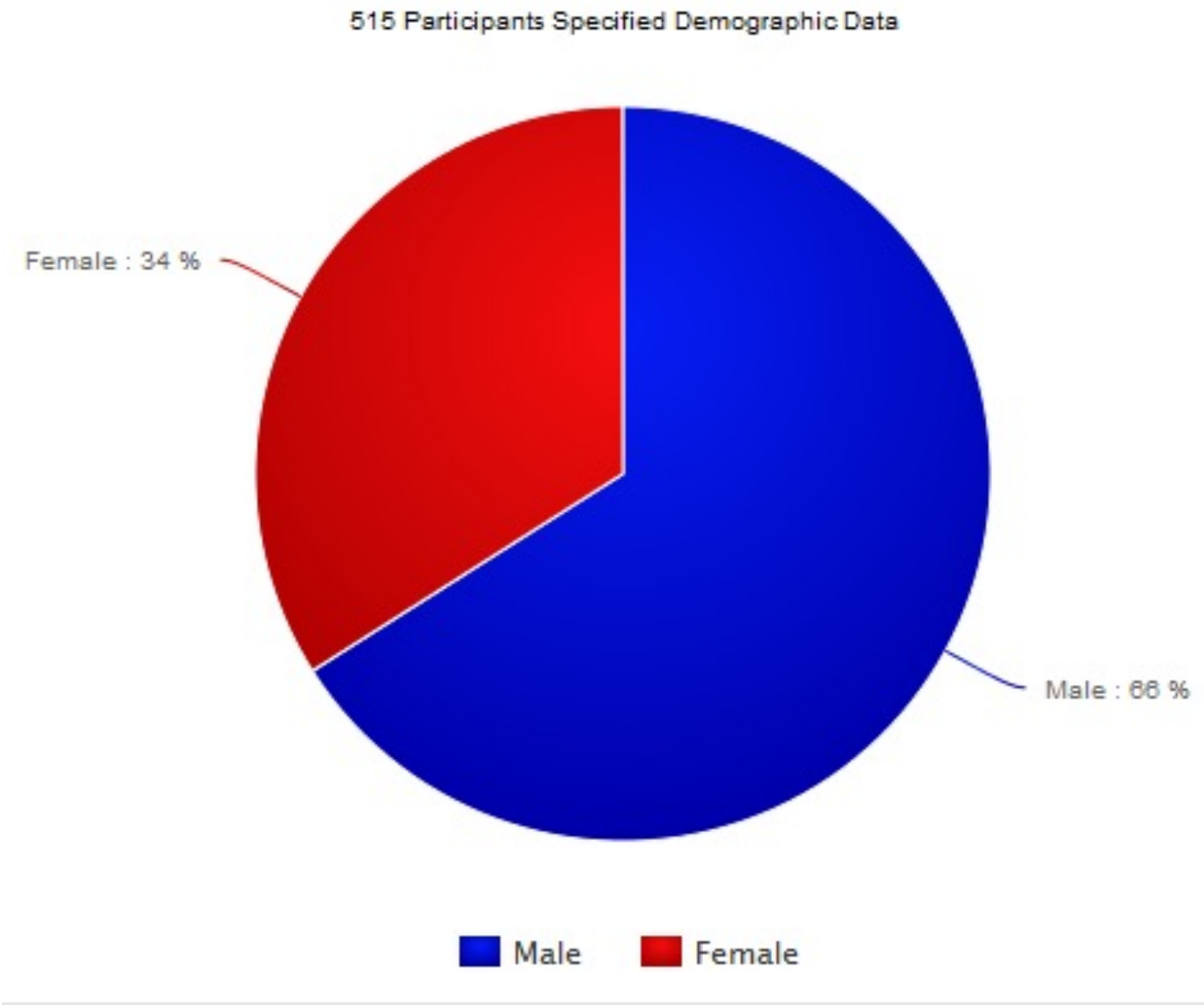
Demographics

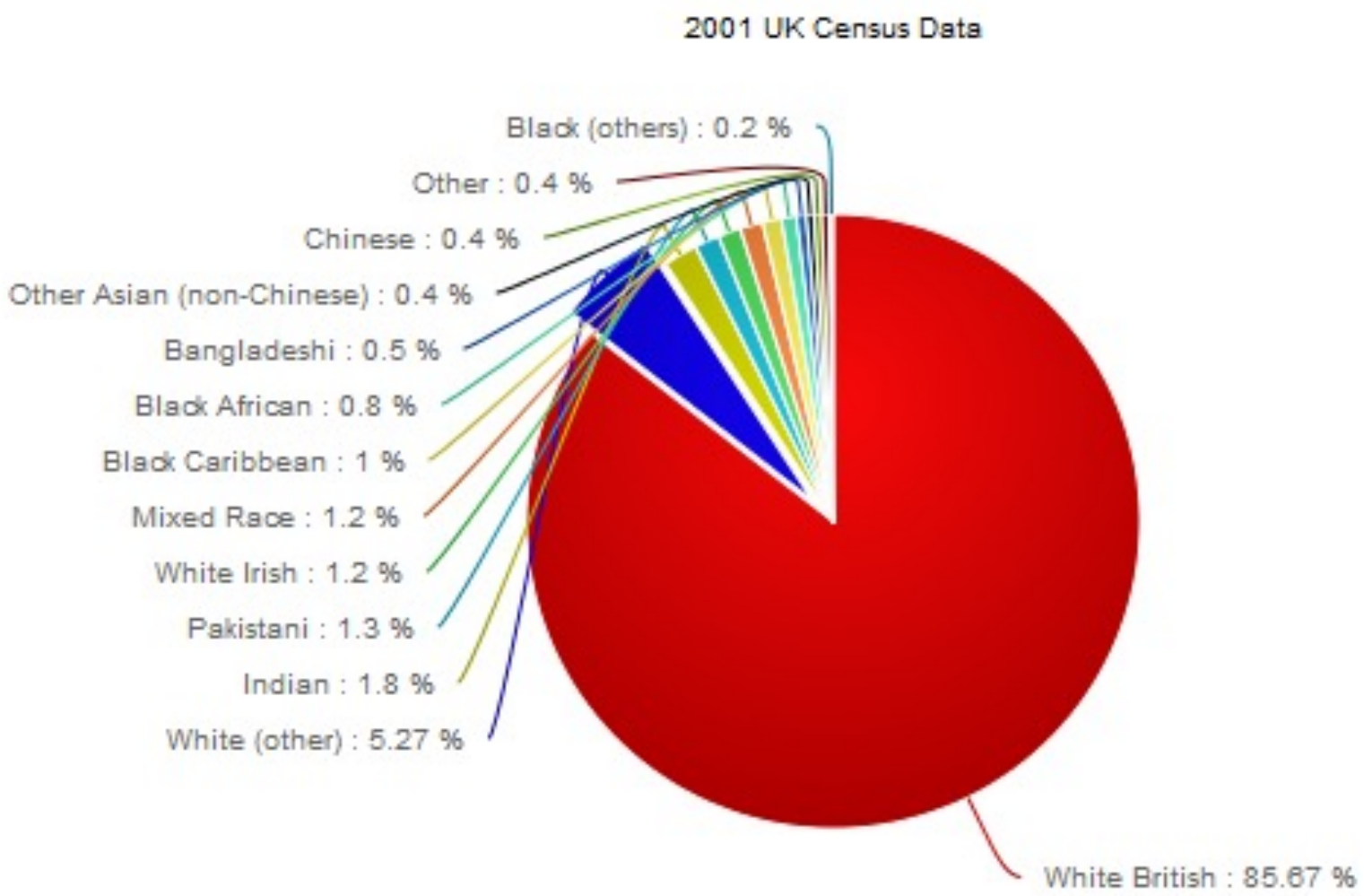
- Platform released in Sept 2012
- After 4 months data was collected from 3,301 participants.
- Participants location was determined by IP Address



Demographics

- 515 Participants specified their personal details:
- Male-Female is 66%-34%
- Avg age is 38.1 years old,



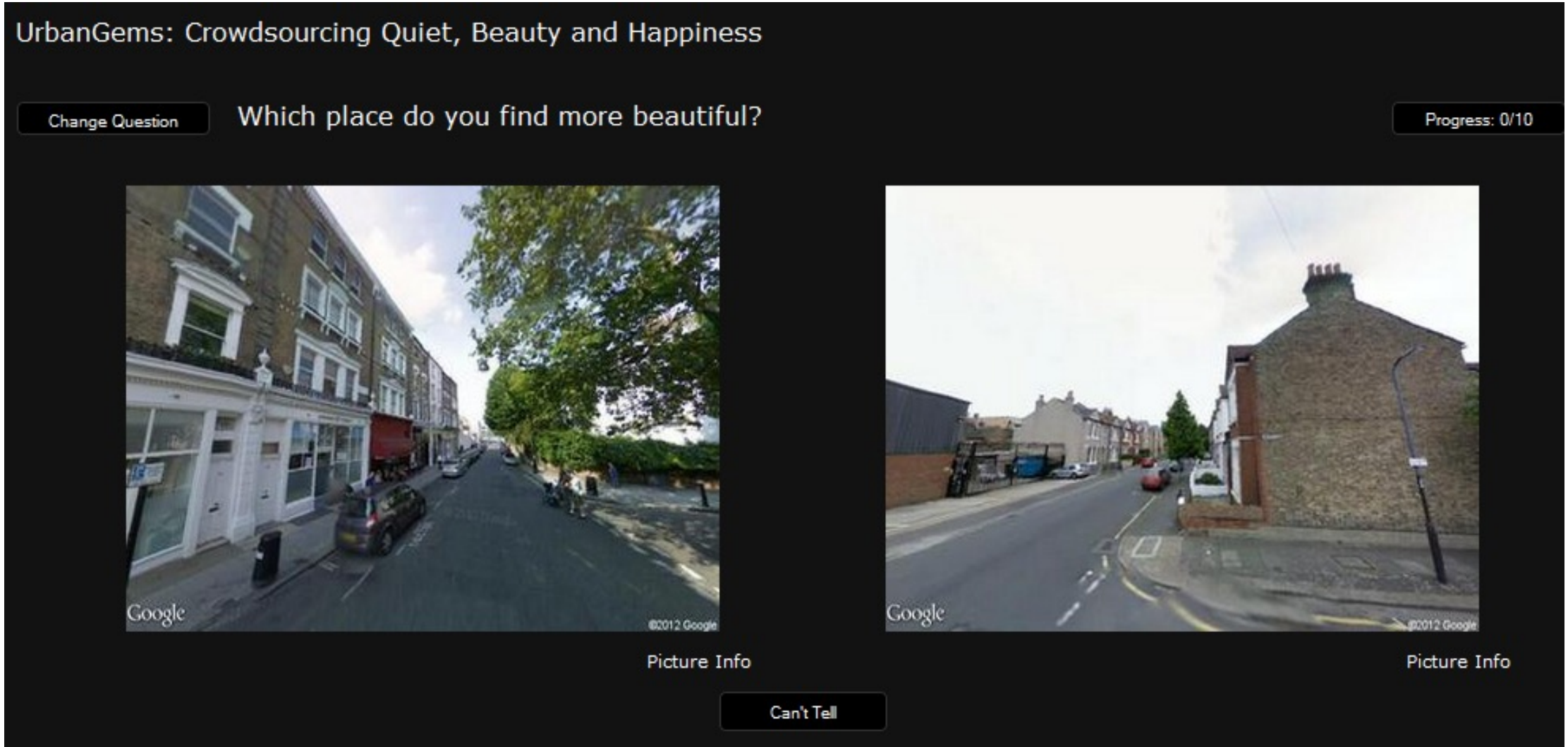


Ethnic group	Population	% of total*
White British	50,366,497	85.67%
White (other)	3,096,169	5.27%
Indian	1,053,411	1.8%
Pakistani	747,285	1.3%
White Irish	691,232	1.2%
Mixed race	677,117	1.2%
Black Caribbean	565,876	1.0%
Black African	485,277	0.8%
Bangladeshi	283,063	0.5%
Other Asian (non-Chinese)	247,664	0.4%
Chinese	247,403	0.4%
Other	230,615	0.4%
Black (others)	97,585	0.2%
Total	58,789,194	100%

* Percentage of total UK population

Ranking/Reliability

- 17,261 rounds of at most ten pairs each
- Rank pictures by their scores for beauty, quiet, and happiness

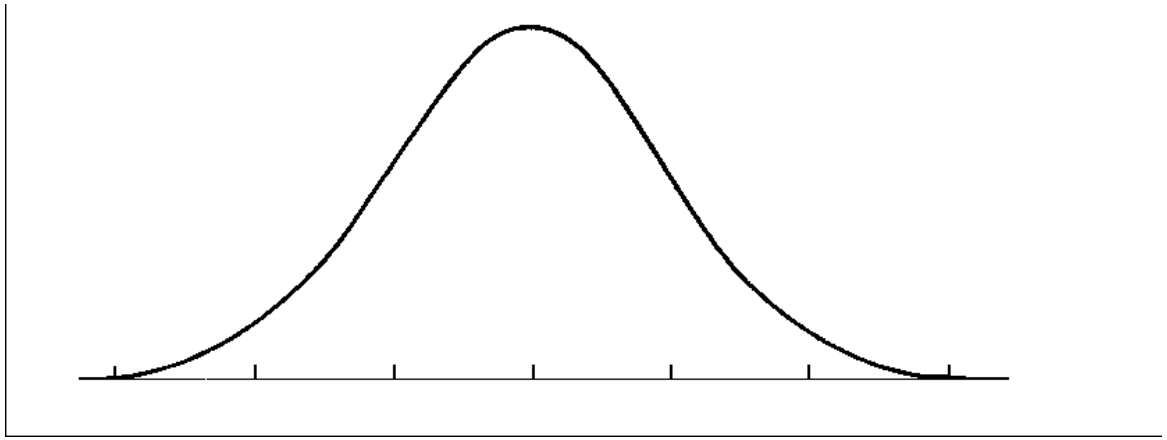


Reliability

The ranking is reliable because

- Number of annotators is >3K
- Distribution of scores is normal with median as high as
  - 171 for beauty
  - 12 for quiet
  - 16 for happy

The number of answers for the three qualities is different as the default question is that on beauty, which thus preferentially attracts more answers.



Controversy

- SketchFactor – Maps “Sketchiness” of an area, Inverse of MBP
  - Creates/Reinforces No-Go Areas
  - Affects rejuvenation efforts and economy of “sketchier” areas
    - Removes capital flow as less people enter and spend
    - Reduces volunteerism and employment
    - Affects property values

Critics

Critics might rightly claim that this work suffers from two main drawbacks

- Relies on crowdsourced ratings which requires intense user involvement.
- Focused on a single city.

To partly address those two concerns, they

- Present a way of predicting the beauty scores from Flickr metadata
- Test not only in London but also in Boston

New Terms

- Psychogeography - dates back to 1955. defined as “the study of the precise laws and specific effects of the geographical environment, consciously organized or not, on the emotions and behavior of individuals“
- The psychogeographer “is able both to identify and to distill the varied ambiances of the urban environment. Emotional zones that cannot be determined simply by architectural or economic conditions must be determined by following the aimless stroll (derive)“
- Mobile applications have been recently proposed to ease making derives (i.e., detours in the city):
- Baeysian learning model

Three main contributions:

- We build a graph whose nodes are locations and whose edges connect geographic neighbors.

With this graph, we rank locations based on whether they are emotionally pleasant.

- We quantitatively validate the extent to which our proposal recommends paths that are not only short but also emotionally-pleasing. We then qualitatively evaluate the recommendations by conducting a user study involving 30 participants in London.

- We finally test the generalizability of our proposal by:
  - a) presenting a way of predicting the beauty scores from Flickr metadata; and
  - b) testing the beauty-derived paths with our 30 participants in London and with a new group of 54 participants in Boston

Extensions

- In addition to using geo-located pictures, one could exploit GPS traces. As opposed to social media, mobile phones enjoy high penetration rates and, as such, GPS traces can help identify interesting places not only in cities but also in suburban regions
- mined such traces by arranging both visited places and mobile users in a bipartite graph, and then ranking places by graph centrality to extract top interesting regions. Their focus was on spotting interesting locations rather than routes.

Selecting Best Path

select the best path from source s to destination d in four steps:

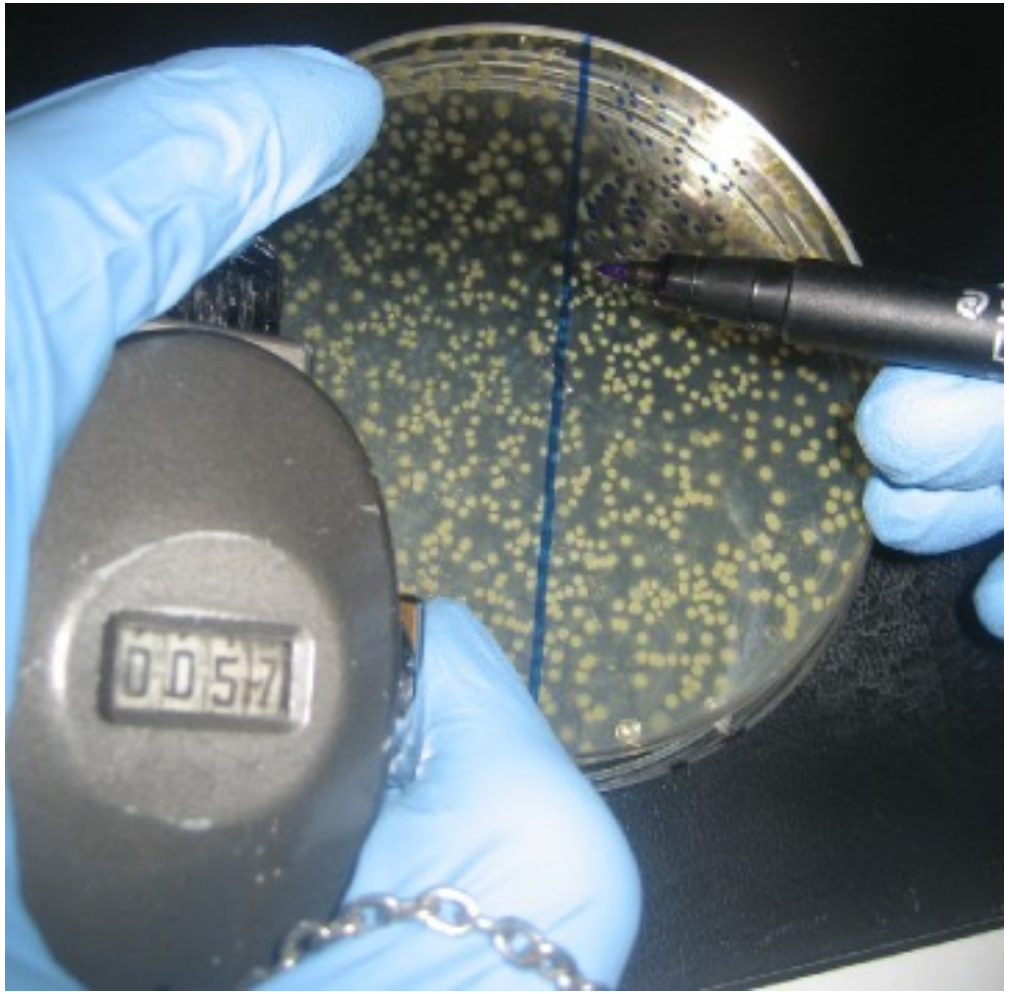
- Step 1. Identify M shortest paths between s and d. To identify them, we
  - run Eppstein's algorithm [11] and the M shortest paths connecting each pair of nodes s and d. To be sufficiently exhaustive", we initially set M to be as high as 106. This choice makes it possible to explore the full set of solutions, including any of the solution that alternative approaches might return (e.g., orienteering algorithms).
- Step 2. Compute the average rank for all locations in each of the first m paths (with m M). For computational tractability, we do not consider all the M paths at once but iteratively explore the first m paths. At each exploration, we record the path with the lowest (best) average rank. Such a path exploration has, of course, diminishing returns (Figure 2, left panel): the more paths we consider (the higher m), the less likely the best value for rank will change. This suggests that it is not necessary to explore all M paths but we can explore a tiny subset of them without loss of performance, and that is what the next step does.
- Step 3. Terminate when the average rank improves less than . To select , we use The Marginal Value Theorem (MVT). This is an analytical tool for optimizing the trade-off between benefit (rank improvement rank) and cost (exploration of the first m paths). One can show analytically that, for the function of rank vs. m, it is best to keep increasing m only until rank m equals rank m ; after that, one should terminate and take the path among those considered that has the best average rank.
- Step 4. Select the path that has so far been found to have the best rank. By repeating these steps for each of the three ranks (beauty, quiet, and happiness), we obtain three paths between s and d in addition to the shortest one (to the baseline).

To rank a location, we need to compute the likelihood that it will be visited because it is pleasant.

- One simple way of expressing that is with p(go|happy) α p(happiness|go).

- By substituting h3i with any of the expressions in Table 1, we obtain the alternative happiness scoring functions. We apply those scoring functions to the two remaining scores of quiet and beauty too. We do so by simply substituting hi with qi (location i's quietness score) and with bi (i's beauty score).

Counting Plated Colonies

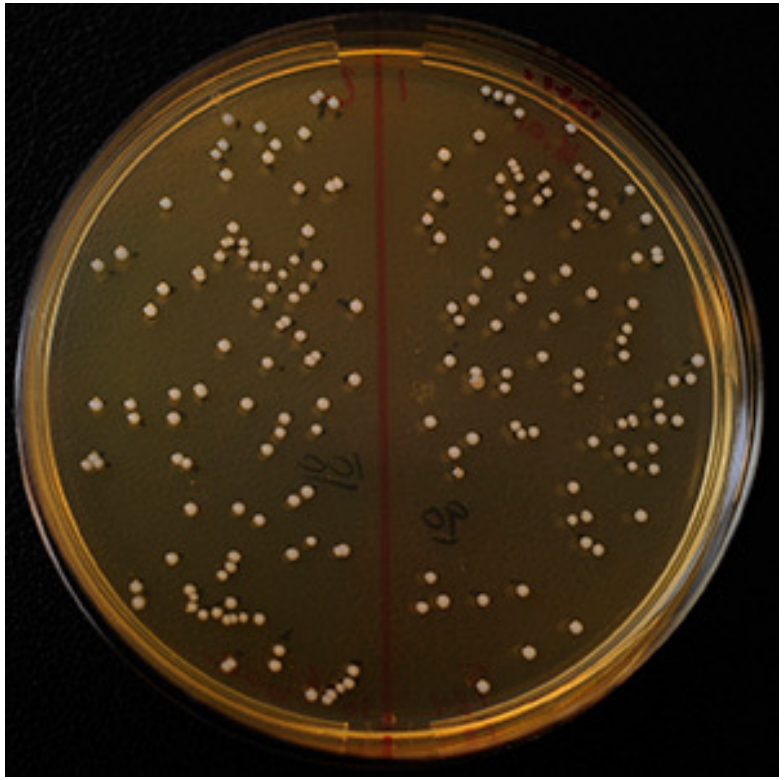


Counting Plated Colonies

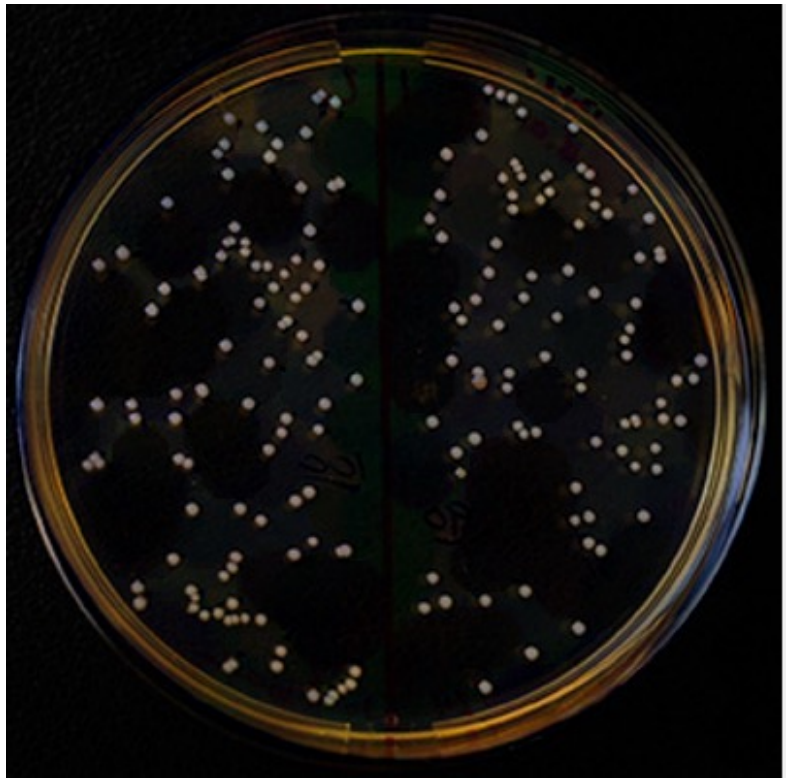
- Binarize the image
  - Top-hat transform
  - Otsu's algorithm for global grayscale thresholding
  - Canny edge detection algorithm
  - Hough's transformation
  - Morphological openings
  - Euclidean distance algorithm
  - Topological watershed with Meyer's algorithm
  - Feature filtering
- Count connected components

Obtaining a uniform background

- White top-hat transform: Image minus its morphological opening by a structuring element (used disk SE w/ radius of 15)

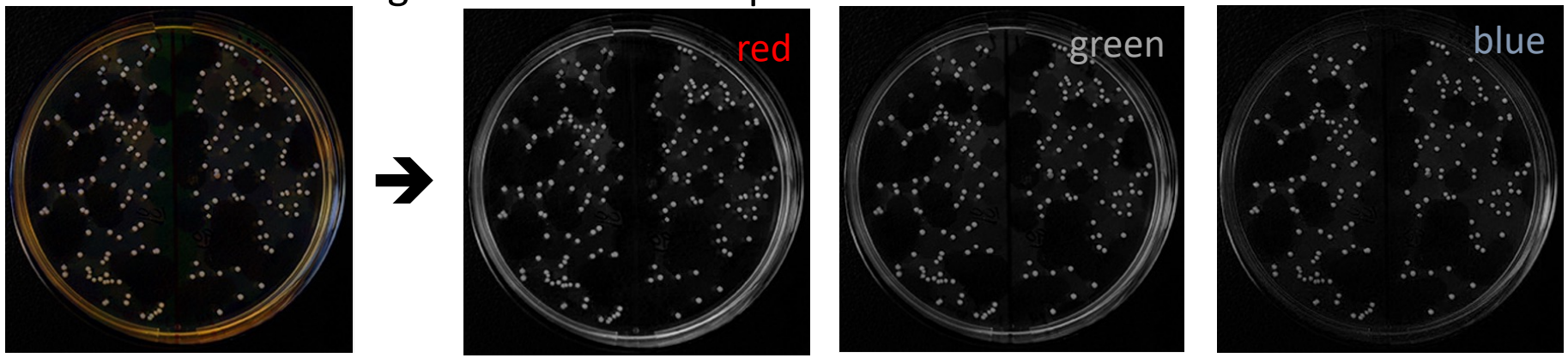


Input image



Computing the initial binarized image

- Extract the image into its RGB components

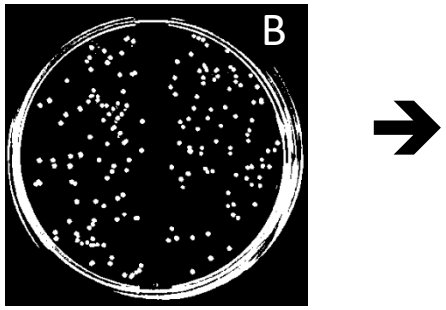


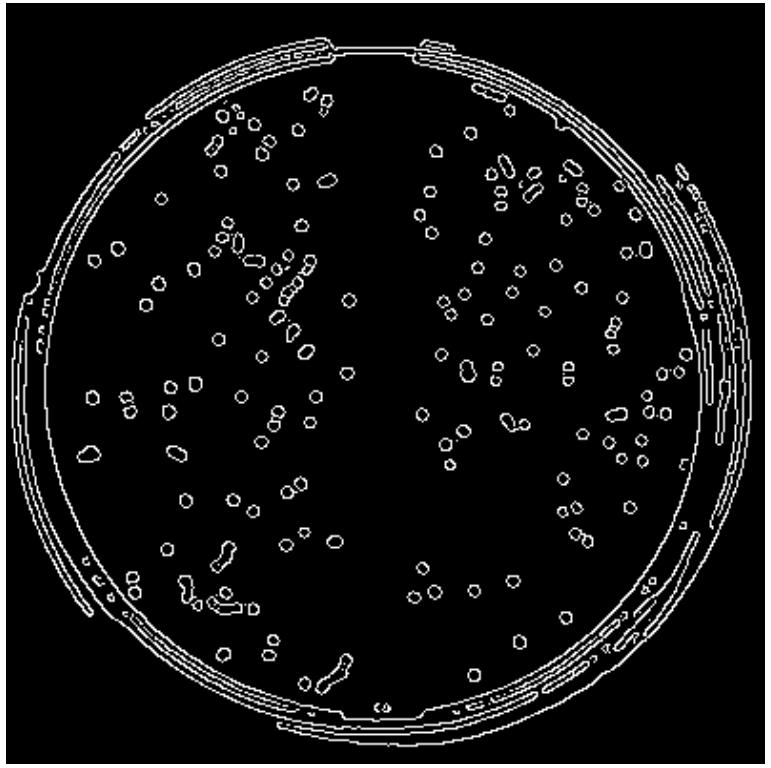
- Compute a global threshold for each of the 4 images above using Otsu's method[1]; create and combine binary images.

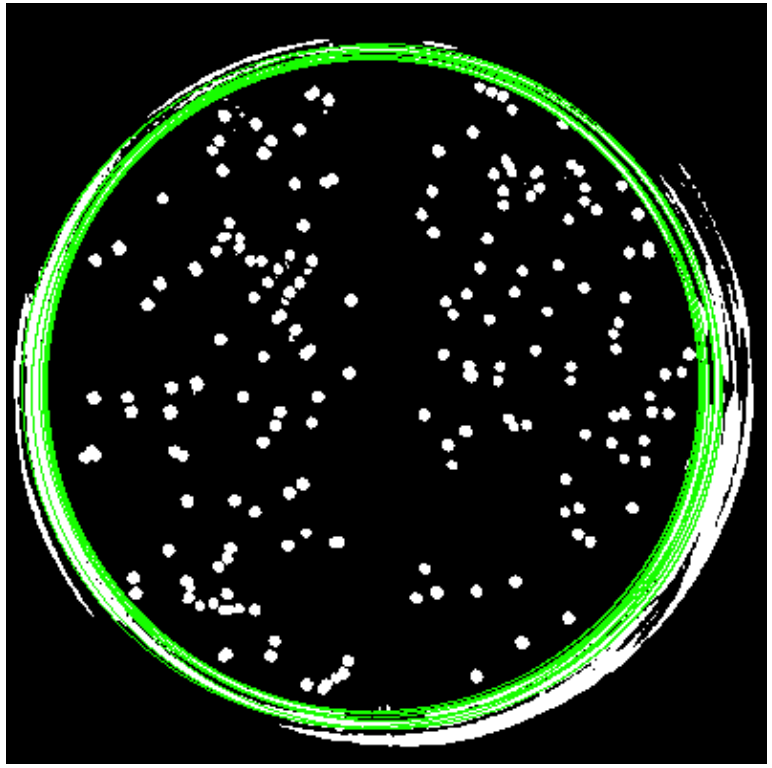
Binary image

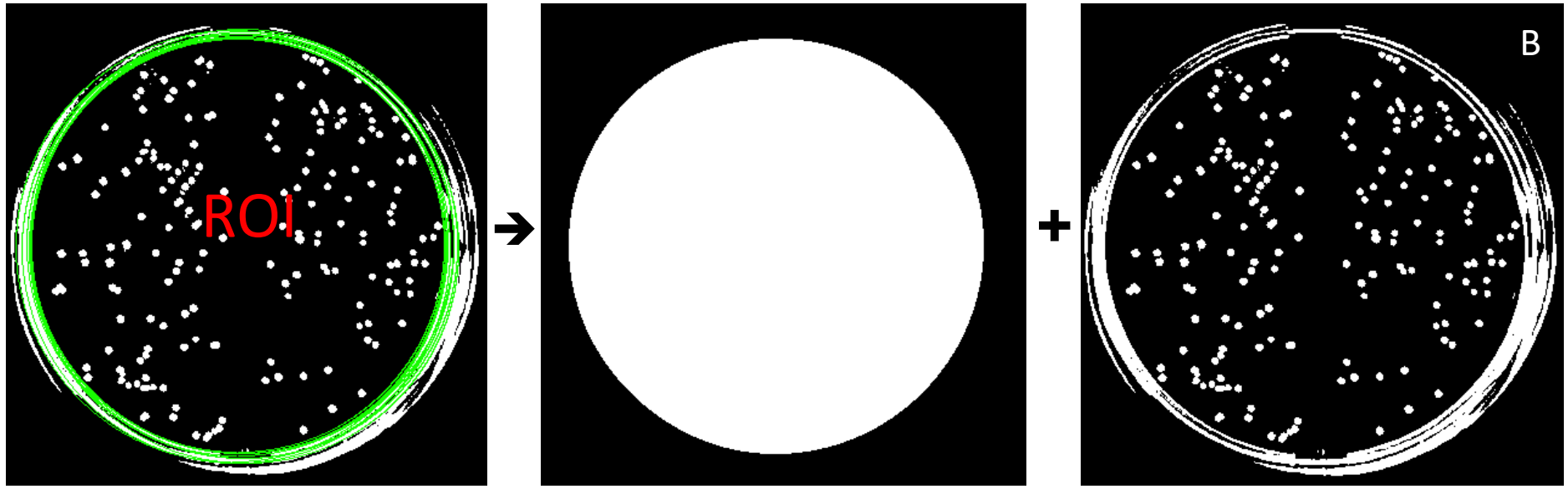
Finding the cell culture plate edge

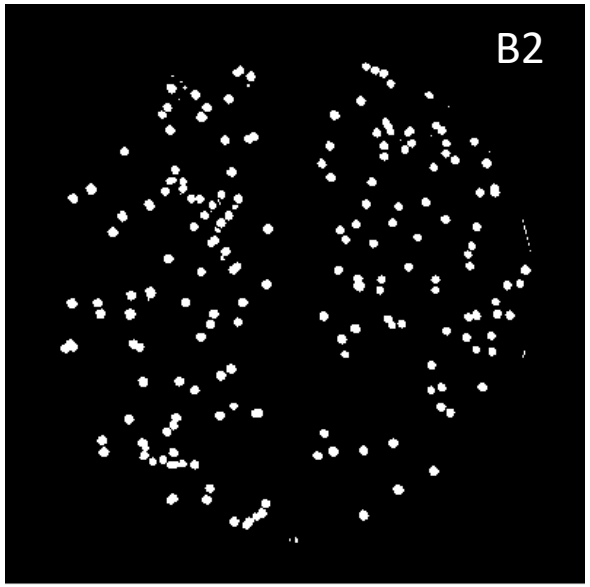
- Canny edge detection algorithm[2]
- Hough's classical transform on circles[3]







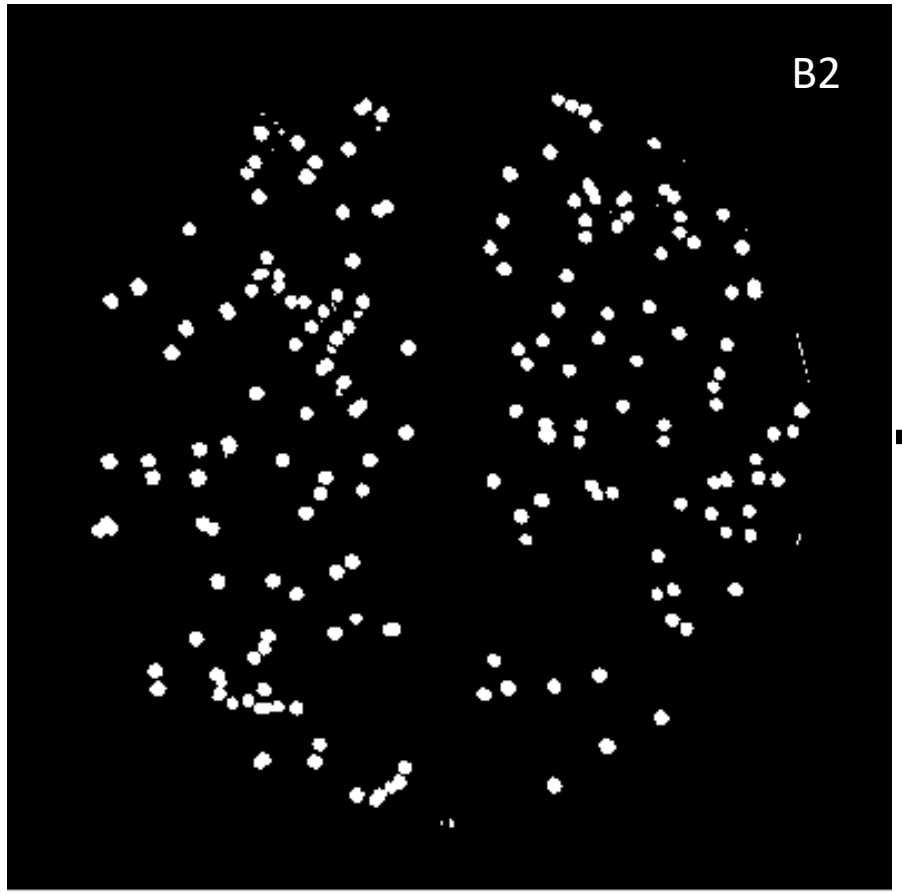


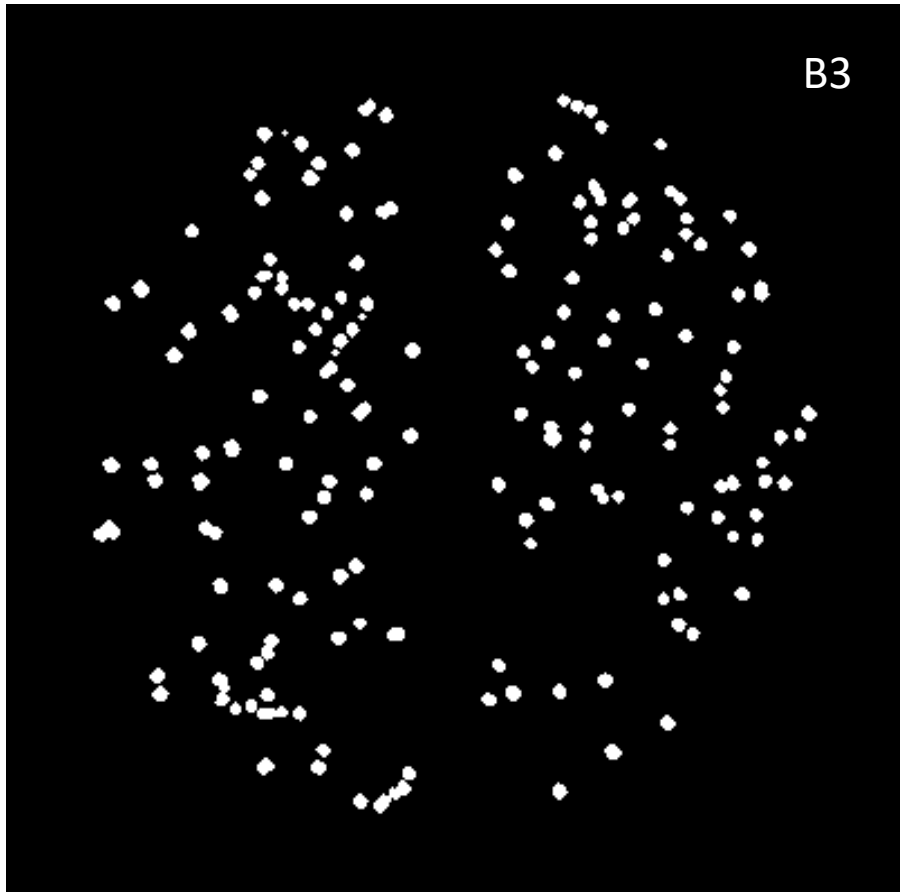


Region of interest masking

Morphological Opening

- Erosion followed by dilation using the same structuring element for both (used disk SE w/ radius of 1)

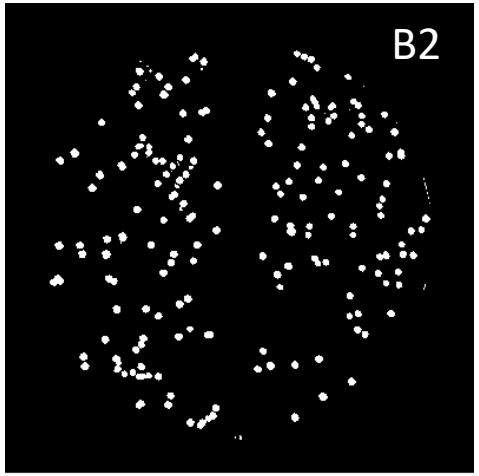




Opened

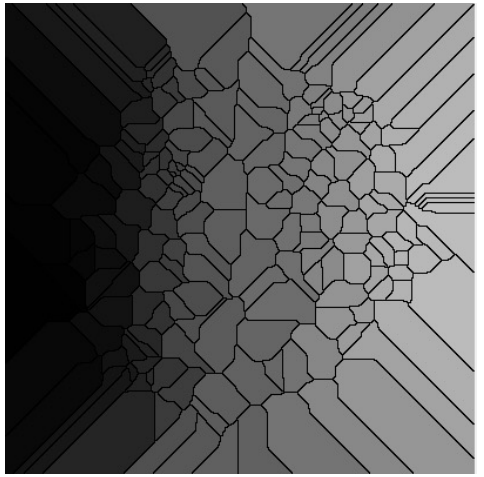
Separate overlapping colonies

Topological watershed via Fernand Meyer's flooding algorithm[4]

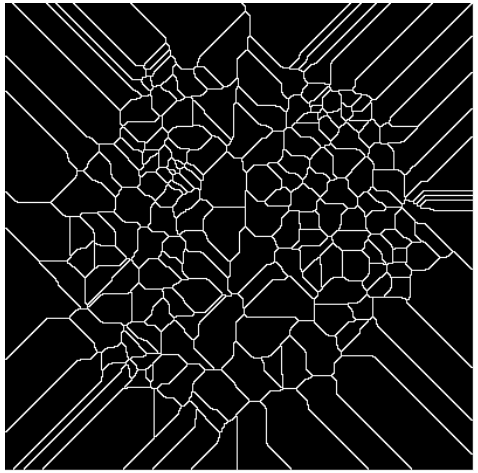


Un-opened binary image

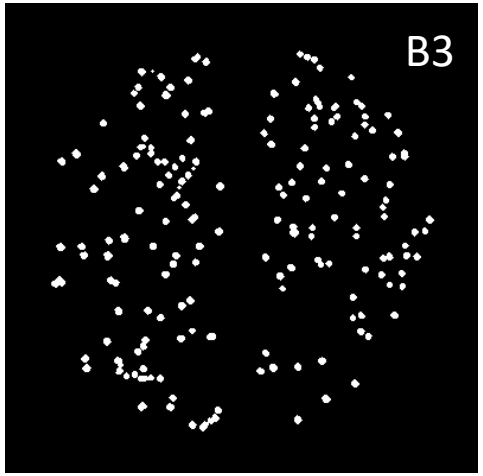
Euclidean distance transform[5]



Watershed transform



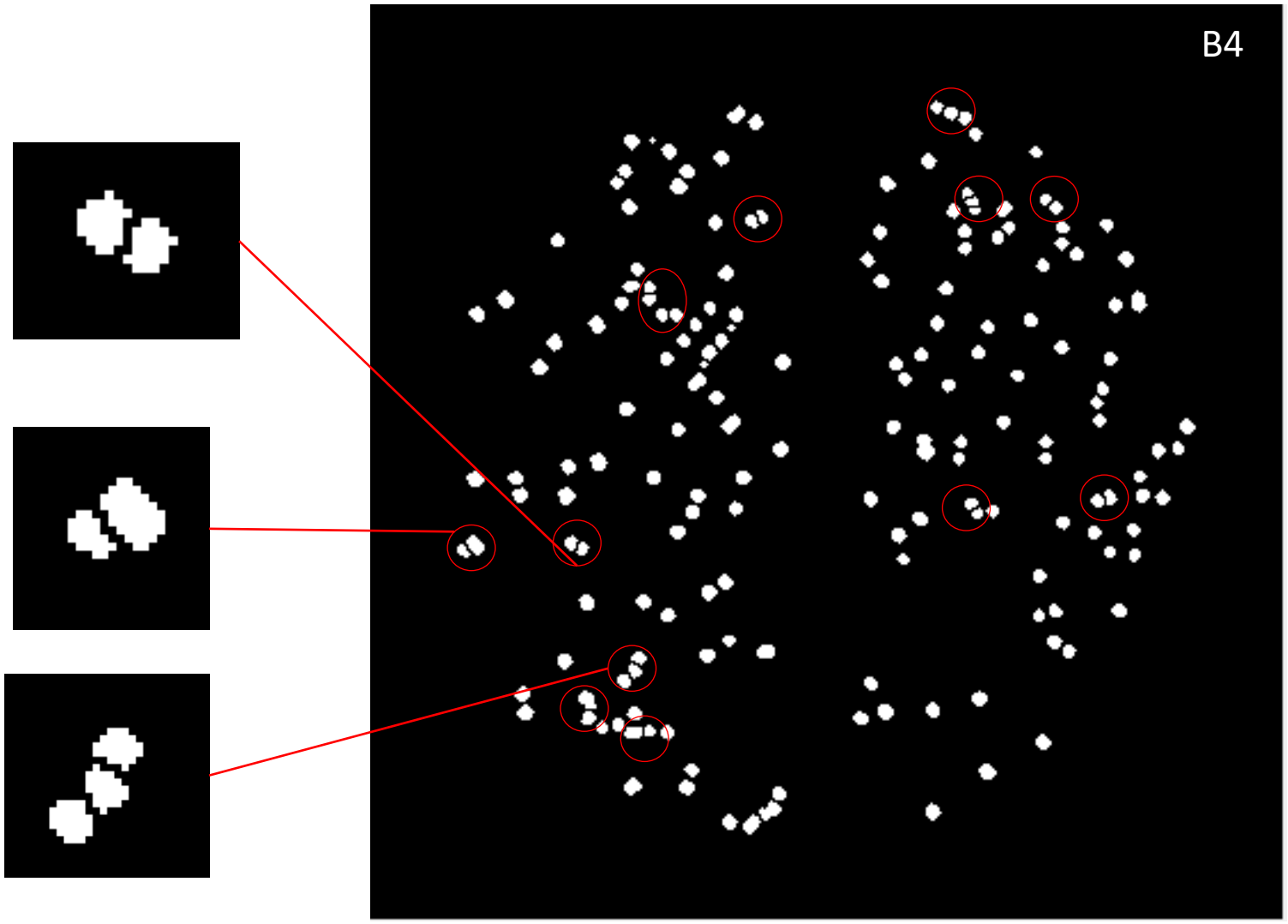
Line separations isolated



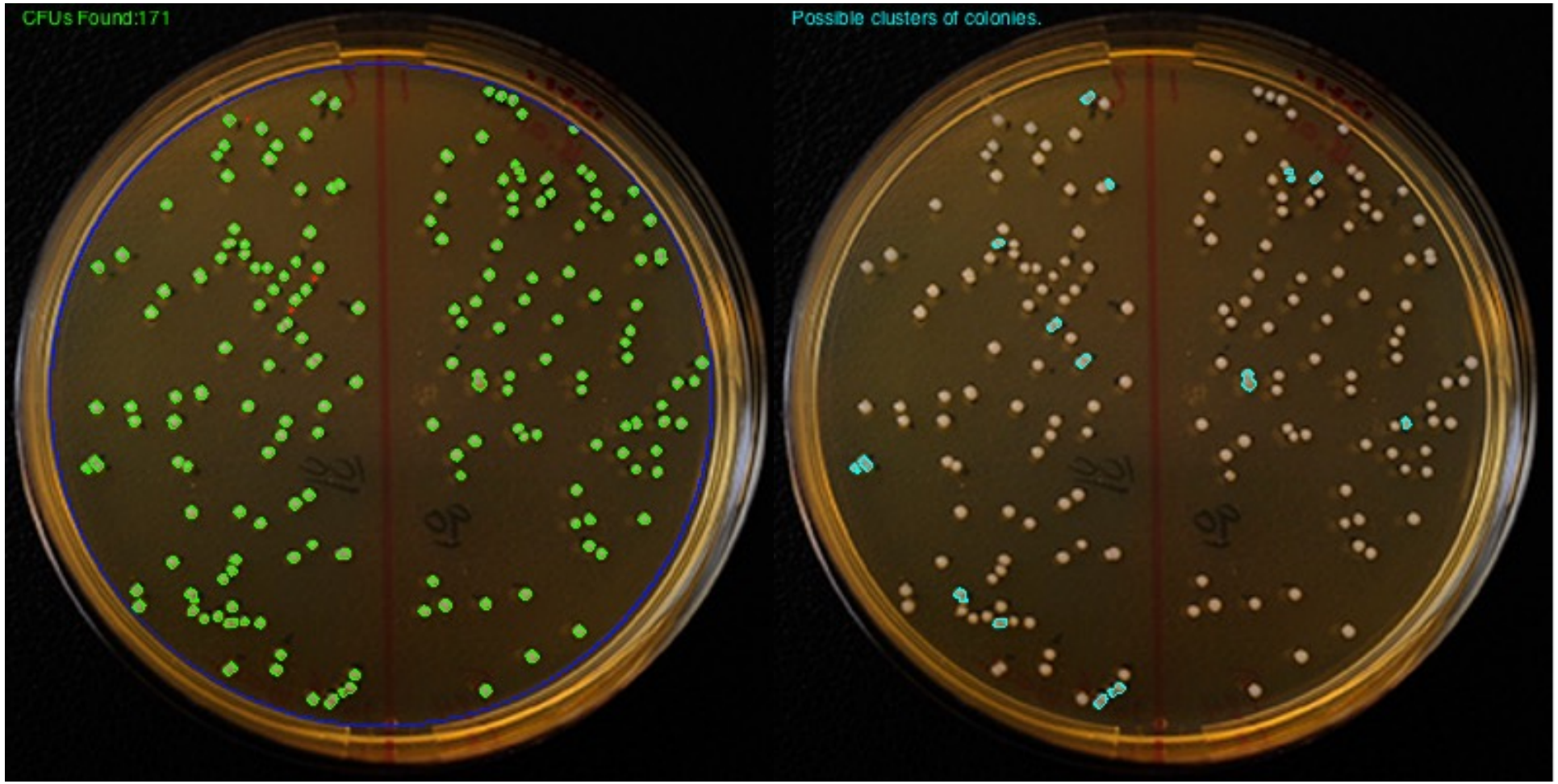
Opened binary image

Set locations of watershed lines in B3 = 0 (black)

Effect of the integrating watershed lines



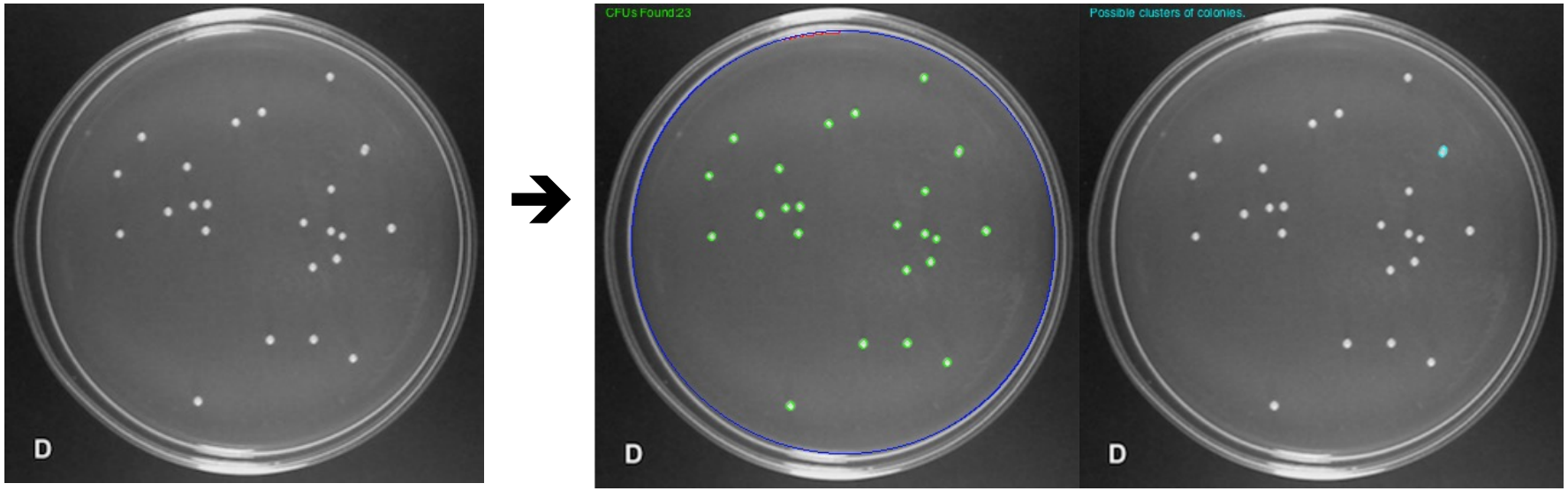
Results so far

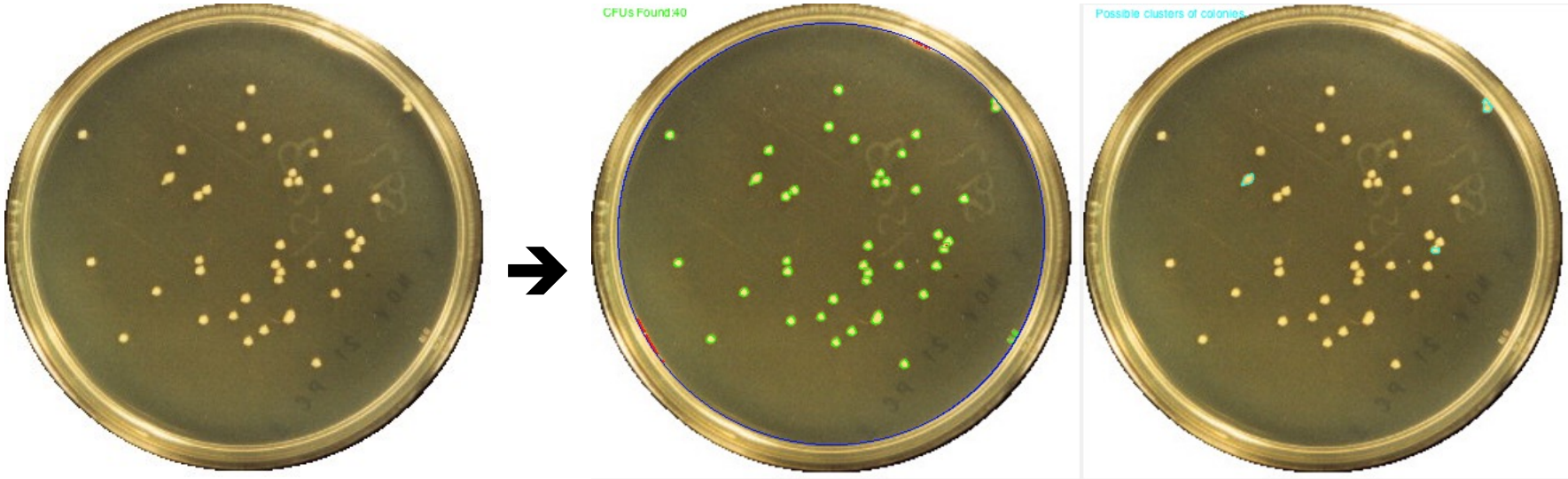


Actual results

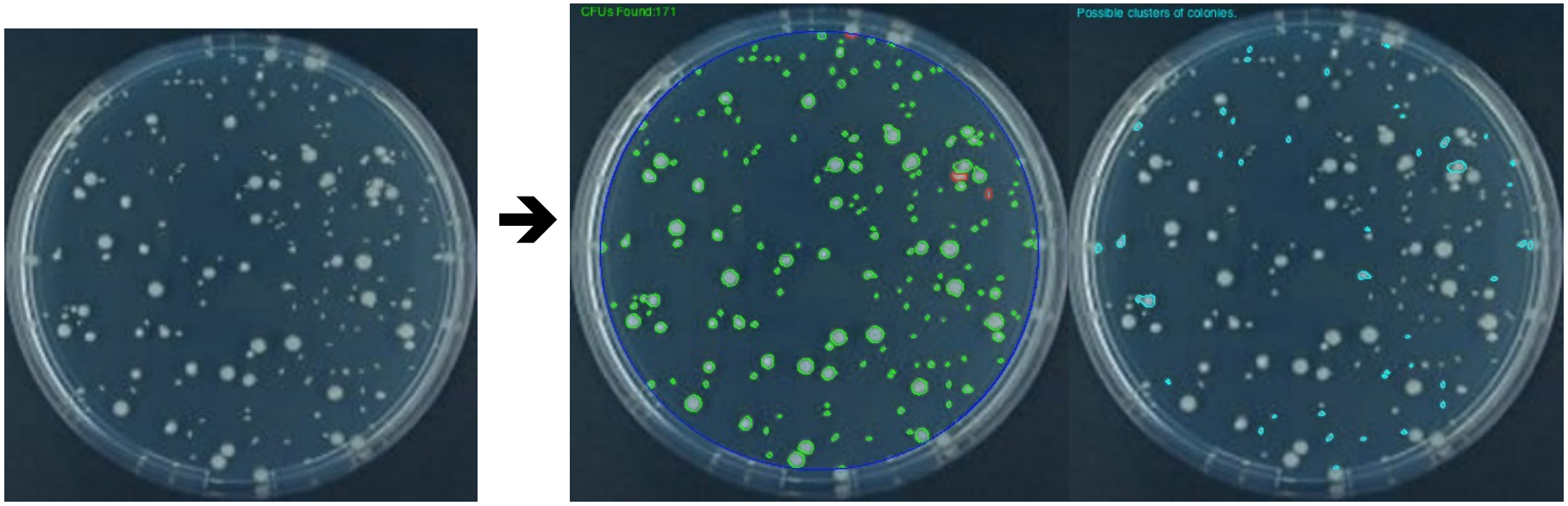
Visualization of colonies that need to be further investigated – not yet implemented

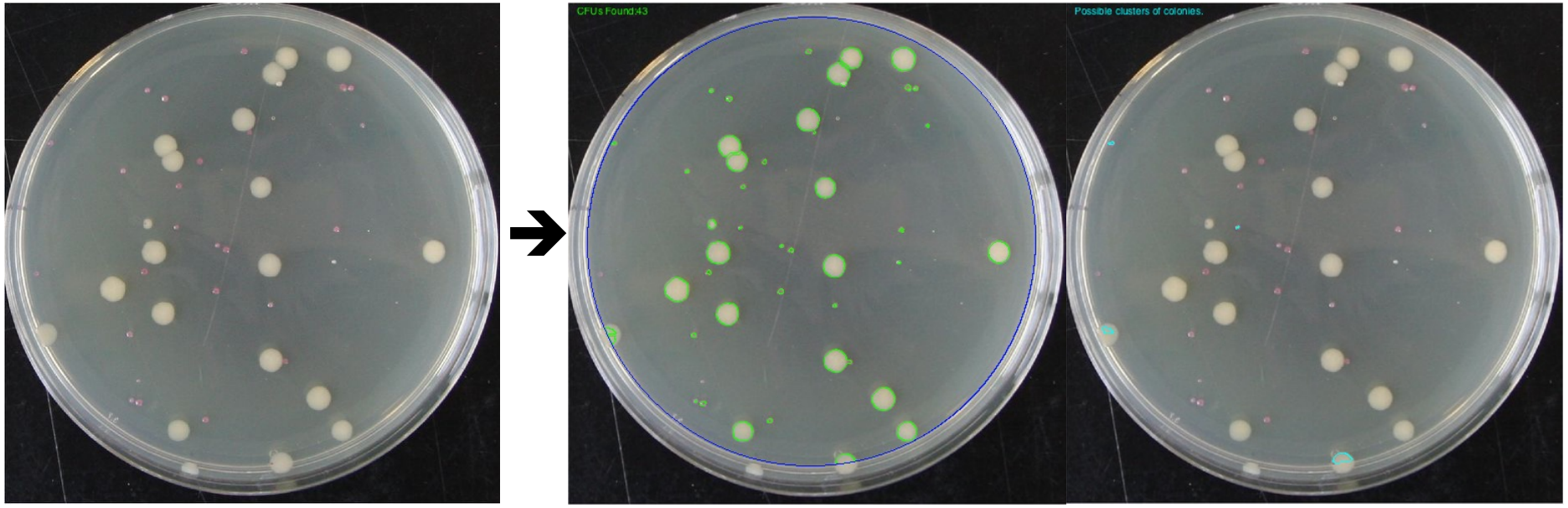
Examples of output from easier images





Examples of output from harder images





Hidden haikus of twitter

Hidden haikus of twitter

- There are a lot of tweets out there, but probably not enough Haikus
- Just below the surface of tweets about trending topics, there are beautiful groupings of 5 and 7 syllables

Finding the Quietest Walking Path Across Boston

Quietest Walking Path

- Most map applications only show the shortest path from start to destinations
- Provide a way for people to find the “quietest” path from A to B
- For people who want to avoid busy roads and tourist attractions, especially within the city

Implementation

- Java based program
- Use Dijkstra's Shortest Path Algorithm
- Method of measuring "quietness"
- Limit the map to a suitable walking distance radius around the starting point A
- If multiple paths are equal in quietness, the user will be given a choice between them

How Quietness is Measured

- Quietness is a percentage score that is based on this scale:
- Highways (0%) < intersections (10%) < main roads (20%) < tourist attractions/shopping centers (40%) < one way streets (60%) < parks (80%) < trails/alleyways (100%)
- Busyness of current road = total walking distance / quietness
- The busyness of all roads in path will be added up and the least busyness value will be determined
- 1000ft path with 100% quietness = 1000ft busyness
- 1000ft path with 50% quietness = 2000ft busyness
- Search algorithm will minimize busyness

Sample Map


Whole Foods Market
Symphony Rd
Public Alley 810
Gainsborough
Hemenway Terrace
Hemenway St
one way street
St Stephen St
0.4 mile
Symphony Sushi
Symphony
9
main road
Huntington
Wine & Spirits
St Botolph St
New England
Conservatory
Speare Pl
Forsyth St
Northeastern Univ

Momentum High-Frequency Trading

High-Frequency Trading

"A program trading platform that uses powerful computers to transact a large number of orders at very fast speeds. High-frequency trading uses complex algorithms to analyze multiple markets and execute orders based on market conditions. Typically, the traders with the fastest execution speeds will be more profitable than traders with slower execution speeds."

-Investopedia

Our Algorithm

Blackberry (BBRY)

Our Algorithm

Magic (MGIC)

Our Algorithm

Xcerra (XCRA)

Procedural Dungeon Generation

Procedural Dungeon Generation – The End Goal

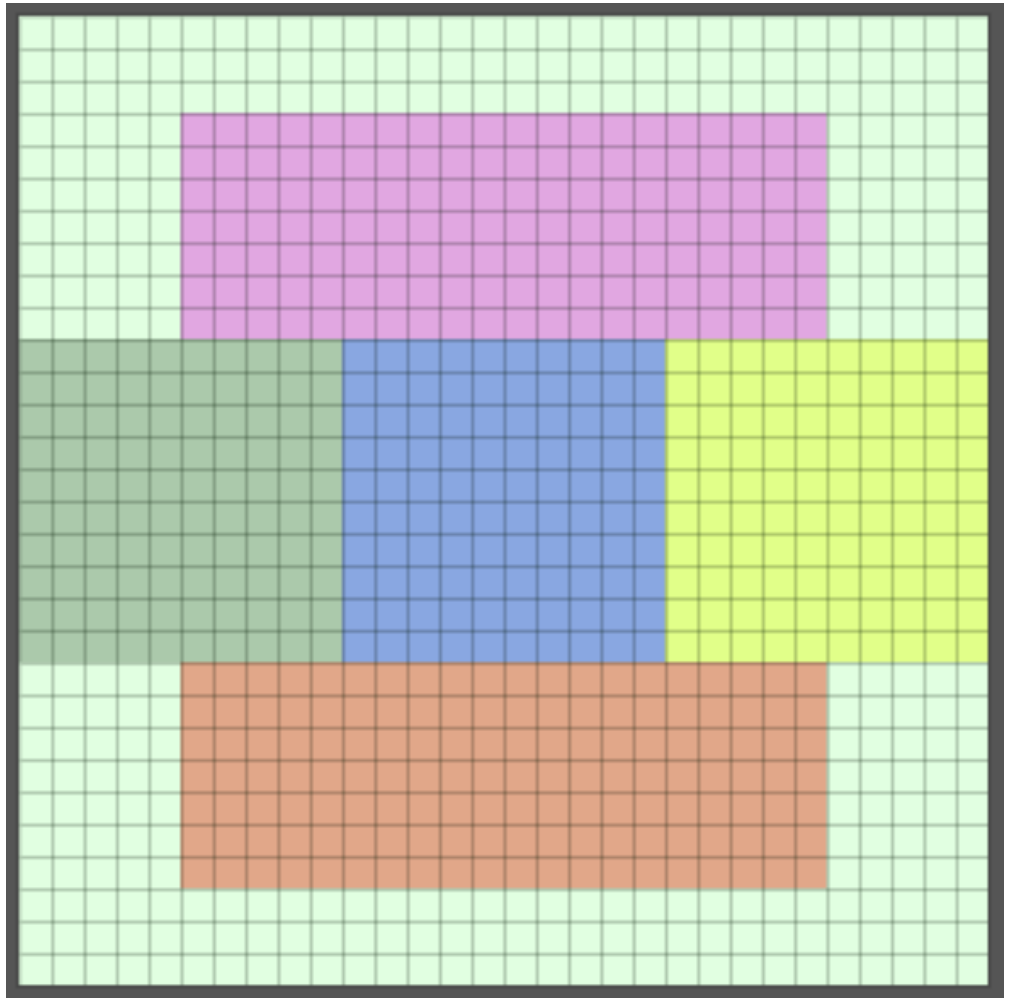
- Create a procedurally generated dungeon for a new map every game
- Create different zones
- Create Hub and Boss rooms
- Create Treasure and Event Rooms
- Don’t create any unreachable rooms

Stages of Creation

- Stage 0: Hardcode in zones and start point
- Stage 1: Create hub and boss rooms
- Stage 2: Create random mid-sized rooms
- Stage 3: Create treasure and event rooms
- Stage 4: Connect nearby rooms

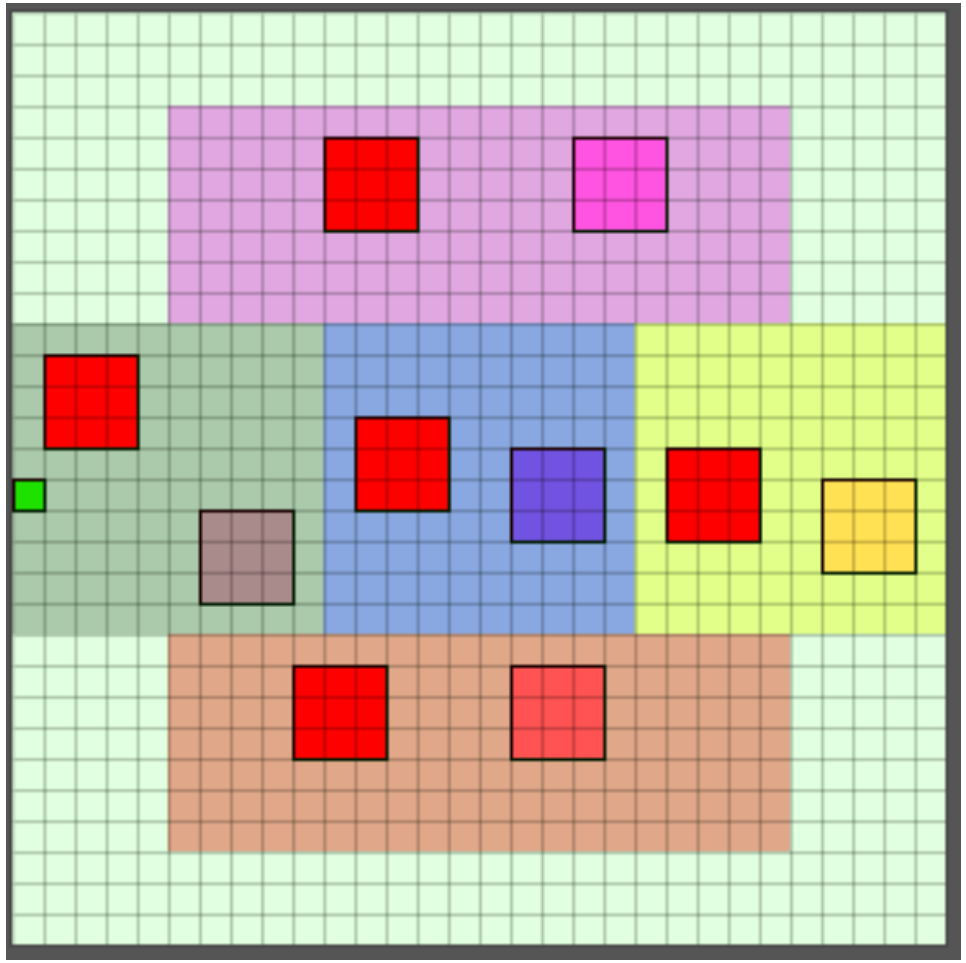
Stage 0

- Creates 5 zones
- Start is made but not yet rendered



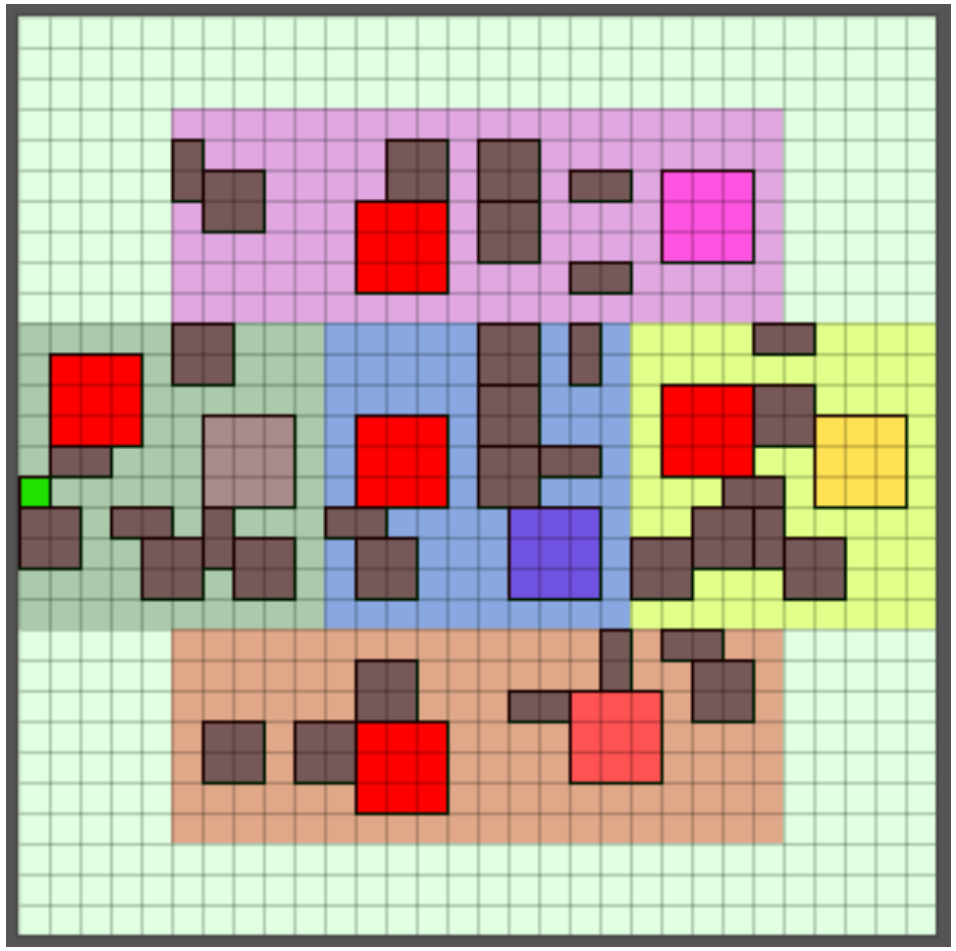
Stage 1

- Creates hub rooms in left half of zone (red)
- Creates boss rooms in right half of zone (zone color)



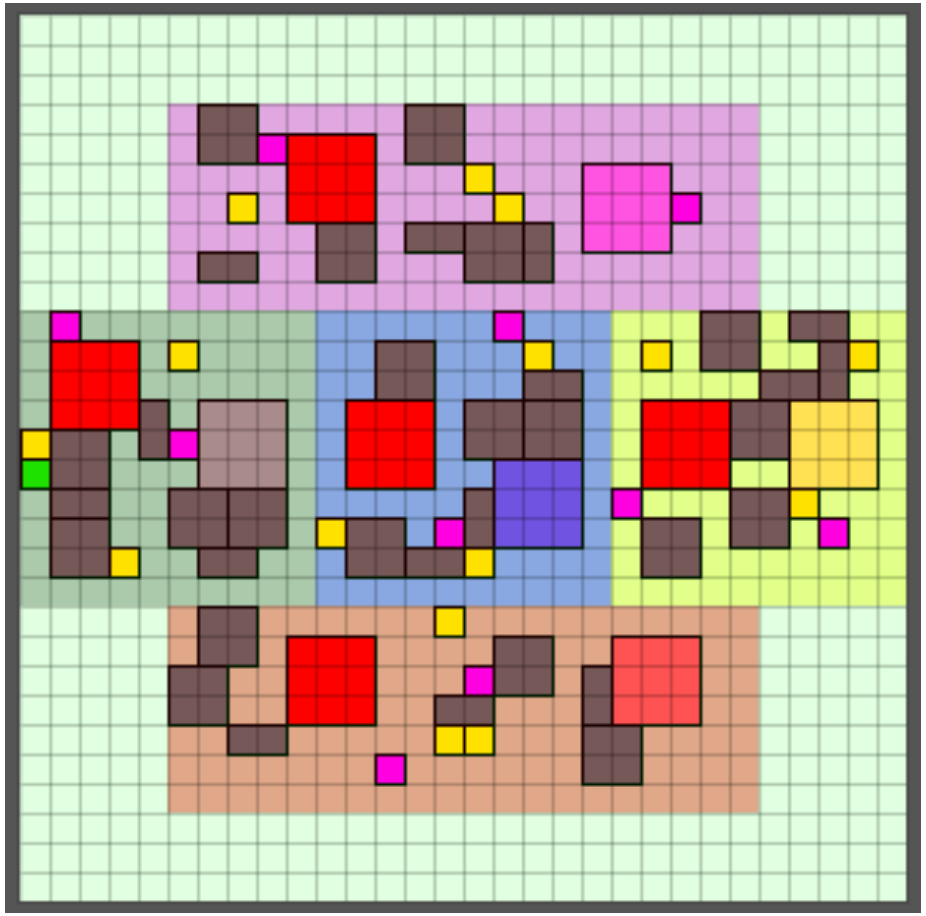
Stage 2

- Creates random rooms of sizes 2x2, 2x1, 1x2



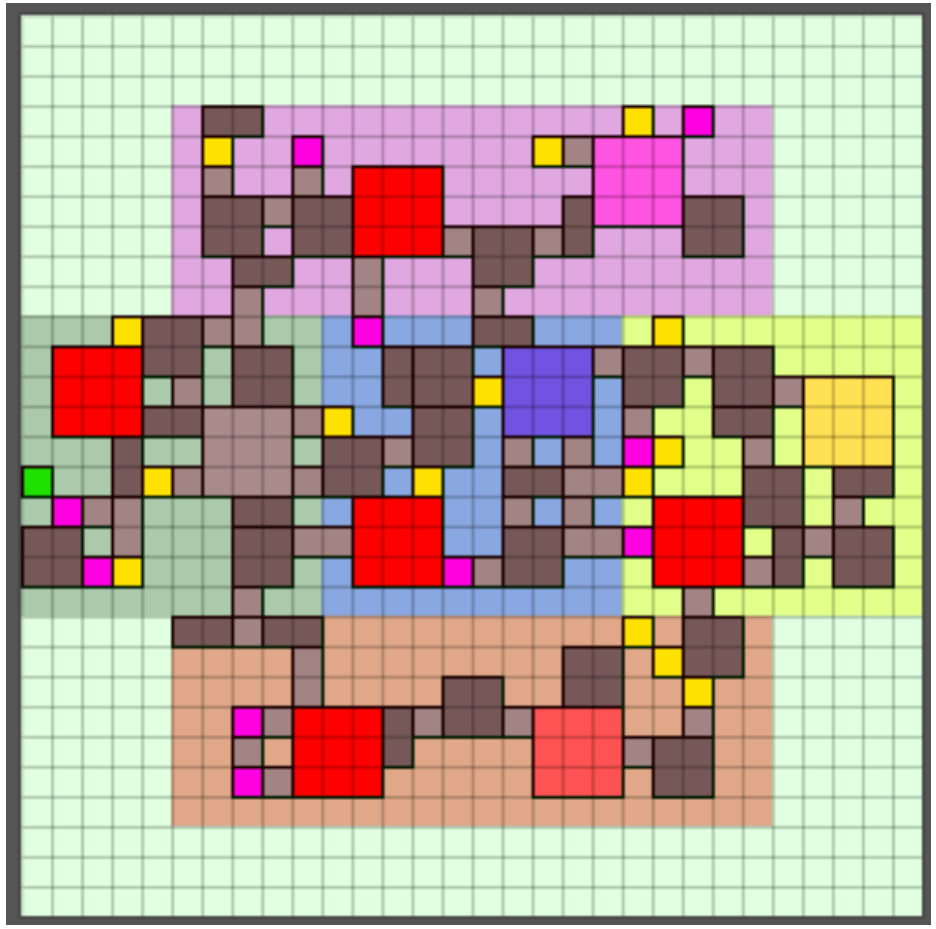
Stage 3

- Creates treasure rooms (yellow)
- Creates event rooms (purple)



Stage 4

- Creates 2x1, 1x2, and 1x1 rooms to connect nearby rooms
- Will not connect rooms if a connection already exists one block away



Stage 5 (in progress)

- Breadth first search through maze, creates list
- Compares full room list to created list, any missing items aren’t connected
- Connect each unconnected room to nearest other room until the map passes the search test

Analysis of Ant-Colony Optimization Algorithms (ACO) for balancing Safety & Distance

Background on ACOs

Solving computational problems by translating/transforming them to path-finding problems on a graph.

Original idea of ACOs: to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food

What I originally wanted to do:

Work on an algorithm that finds the just safest path.

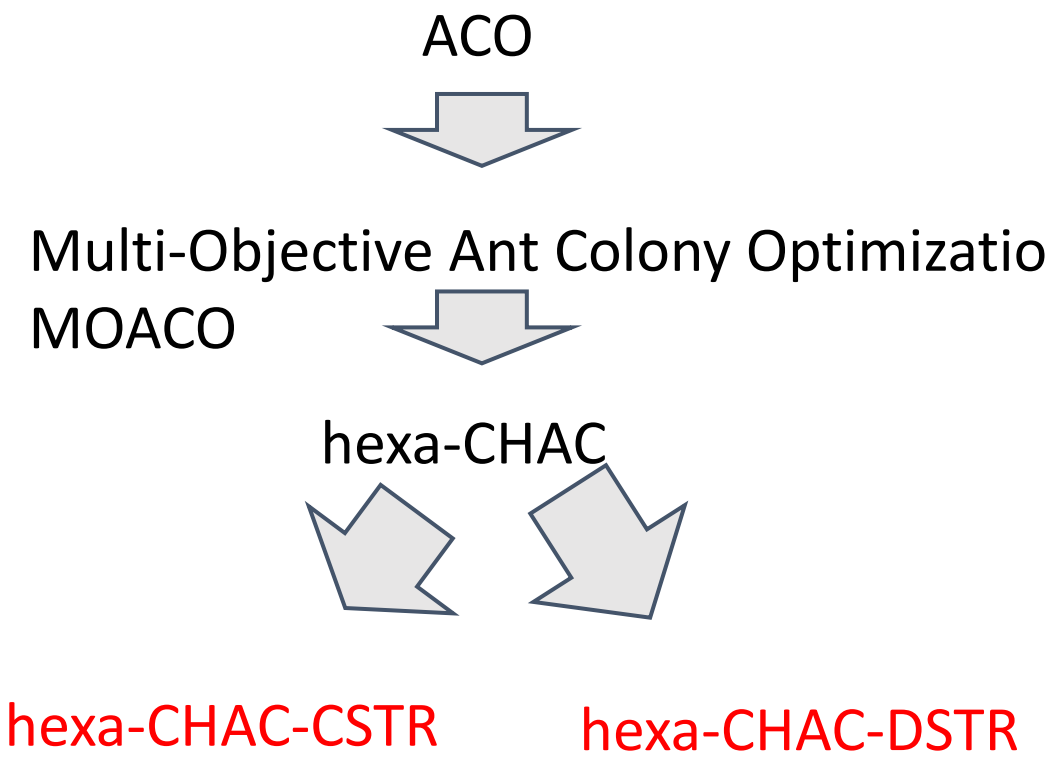
Existing literature did not produce the desired results (i.e. the safest path was not always guaranteed)

Instead, I decided to expand and focus on bicriteria problems where there was slightly more research and information available.

→ minimizing distance + maximizing safety

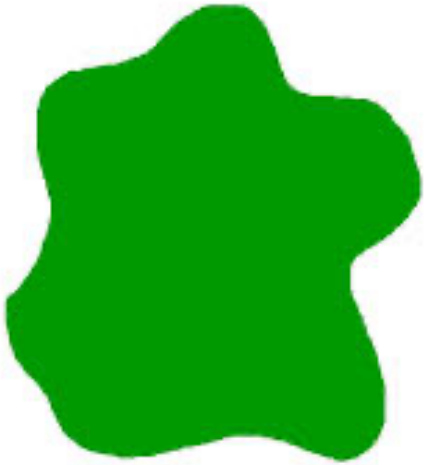
What I have worked on so far

Broken down a family of ACOs to help me narrow my focus:



Algorithms applied to filling Parking Lots

Question:



Suppose we have a randomly shaped area

How can we fill as many parking spaces as possible with the premise of the path not getting blocked?

Why is this interesting?

- In crowded cities, the problem of finding an empty parking space can be so dominant that it often enters everyday conversations of citizens.
- We hope that our exploration of algorithms can bring a novel solution to this headache to commuters.
- Our algorithmic approach is interesting as we are research different ways to find an optimal case where cars are parked and can leave.
- We are learning about different algorithms throughout the process

Things to consider

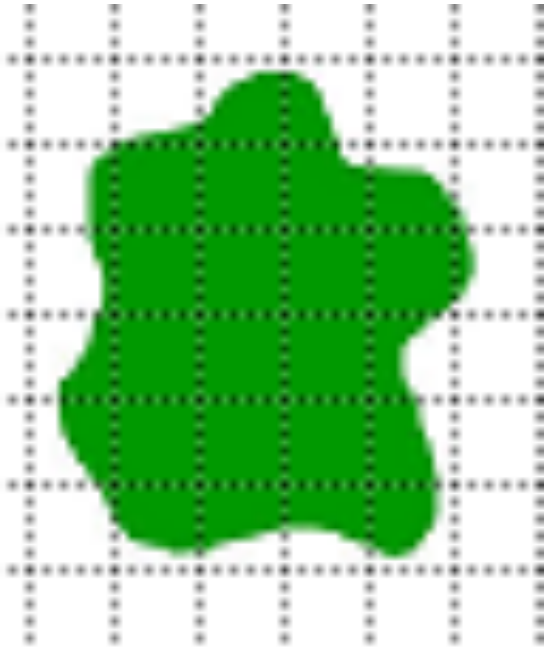
- What algorithms can we use to “park” as many cars (assuming all cars are the same size) in a parking lot so that there is at least one exit (from the area) which is not blocked by a car?
- How should we simulate this problem in a program?
- How should we represent this problem?

Genetic algorithm (GA) basics

- Random -> optimized
- We will use the process of natural selection to reach an optimal parking lot

- **Initialization**
- **Selection**
- **Genetic operators**
- **Termination**

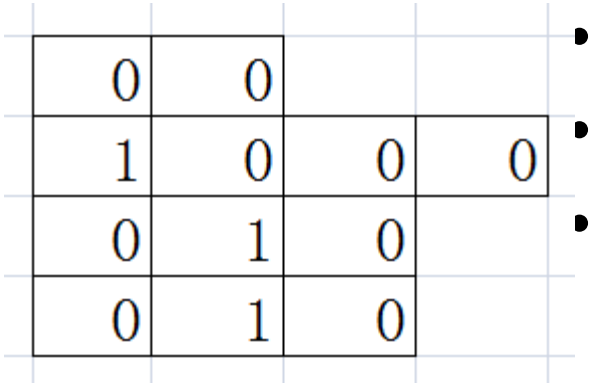
What we have done so far



- **Initialization**
- 0 is an empty space
- 1 is a car occupied space
- K0 – base case (empty parking lot)

0	0		
0	0	0	0
0	0	0	
0	0	0	

How GA mutation works



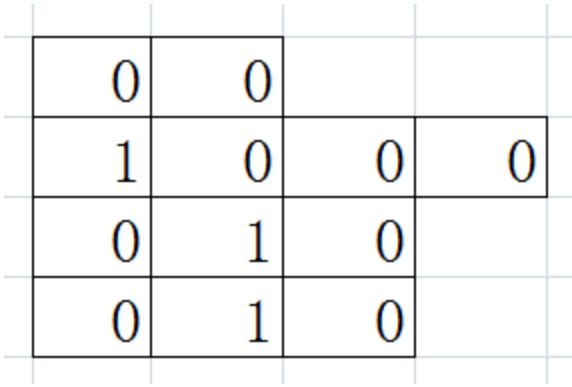
0	0		
1	0	0	0
0	1	0	
0	1	0	

- Evolution operators:
- 0 will evolve to 1 at the rate of P.
- P = Probability
- Cars filling up the parking lot

- Selection:
- Gets rid of unsatisfactory cases – a car cannot leave the parking lot

Pseudocode

- Initialize k0 // empty case
- Evolution-operators（k0) * population；// population = 10
- Store 10 offspring cases to set S1；
- Selection(s1)；// delete cases where a car cannot exit
- K1 = max case（Selection(s1)）；
- Loop:（k1) * population；
- While (Selection(sx) = 0）
- Return k(x-1)



0	0		
1	0	0	0
0	1	0	
0	1	0	

Next steps

- **Looking into other algorithms**

  (keep options open)

- **Implementing a Genetic Algorithm in Python, and writing code to simulate our problem (randomly shaped parking lot, and cars) in Python**

- **Finish writing project report**

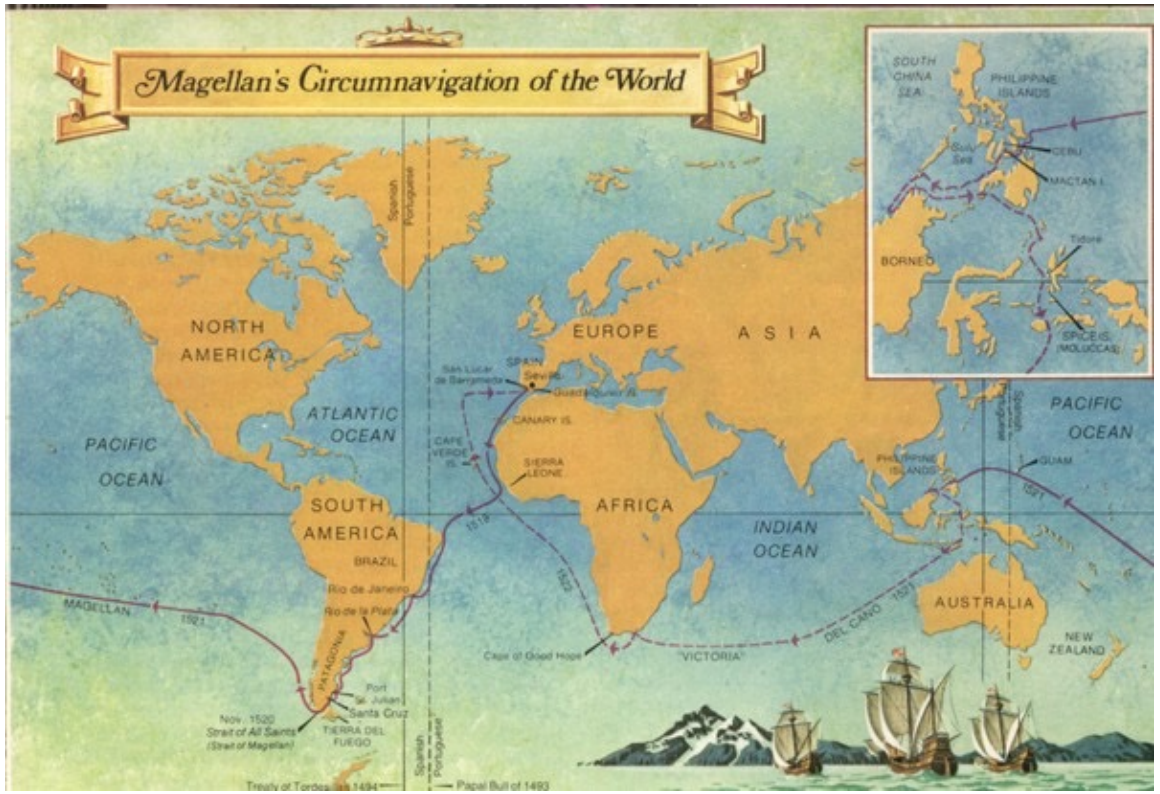
Circumnavigating the World: Relaxation, Food & Adventure

Determining the best way to travel the world based on certain factors

http://png-4.findicons.com/files/icons/750/aquanox_mini/256/globe.png

Data Collection

- Determine rankings of cities/places
- “Best of” Lists- simulate poll data
- Empirical Data
- Compiling lists together



Hawaii	9
Costa Rica	8
Thailand	8
Mexico	7
Seychelles	6
Australia	5
Bali	5
Georgia	5
Greece	5
Jamaica	5
Maldives	5
Puerto Ric	5
Belize	4
Bora Bora	4
Dominicar	4
Fiji	4
Bahamas	3
Cancun	3
Iceland	3
Italy	3
Miami	3
New Zeala	3
St. Lucia	3
Turks & Ca	3
Barbados	2
Cape Cod	2
Charlotte /	2
Florida	2
London	2
Malaysia	2
Rio	2
Santa Fe	2
Sedona, A	2
St. Barts	2
St. Vincent	2
Amsterdar	1
Andalusia	1
Anguilla	1
Aruba	1
Aspen	1

Austin	1
Barcelona	1
Bermuda	1
British Virg	1
Canada	1
Cape Tow	1
Capri	1
Cayman	1
Chiang Ma	1
Cozumel	1
Croatia	1
Cuba	1
Devon	1
Dunton, C	1
Easter Isla	1
Fernando,	1
Flathead, I	1
Galapagos	1
Granada	1
Hamptons	1
Honduras	1
Jaipur, Ind	1
Jenner, CA	1
Kenya(coa	1
LA	1
Marrakech	1
Myanmar	1
Myrtle Bea	1
Nantucket	1
New Delhi	1
New Hamp	1
Norway	1
Oman	1
Oregon	1
Orlando	1
Outer Ban	1
Paris	1
Patagonia	1
Philippines	1
Pitcairn	1
Reykjavik	1

Left-Relaxation
Right-Food

Bangkok	9
New York	9
Tokyo	8
Paris	7
Istanbul	6
London	6
Barcelona	5
Ho Chi Mi	5
Marrakech	5
Mexico Cit	5
New Orlea	5
San Franci	5
Singapore	5
Copenhag	4
Hong Kong	4
Penang	4
Brussels	3
Chicago	3
Melbourn	3
Abergris C	2
Bologna	2
Bombay	2
Buenos Ai	2
Lyon	2
Mumbai	2
Palermo	2
Portland	2
Rio	2
Rome	2
San Sebas	2
Sydney	2
Tel Aviv	2
Austin	1
Berlin	1
Bordeaux	1
Buffalo	1
Cartagena	1
Charlestor	1
Chennai	1
Cincinnati	1

Edam	1
Ensenada	1
Florence	1
Jaipur	1
Kuala Lum	1
Los Angele	1
Louisville	1
Montreal	1
Napa Valle	1
Naples	1
Oaxaca	1
Osaka	1
San Juan	1
Seattle	1
Seoul	1
Shanghai	1
St. Louis	1
Taipei	1

http://travel.nationalgeographic.com/travel/top-10/food-cities/
http://www.travelchannel.com/interests/food-and-drink/photos/editors-pick-our-favorite-food-cities
http://www.foodandwine.com/slideshows/worlds-best-cities-for-street-food
http://www.thrillist.com/eat/nation/the-world-s-best-food-cities
http://www.travelandleisure.com/worlds-best-spas-2014-winners-list
http://www.foodandwine.com/slideshows/worlds-5-best-food-cities/1
http://www.forbes.com/pictures/ehlk45fiii/the-worlds-top-10-cities-for-street-food-3/
http://www.ucityguides.com/cities/top-10-best-food-cities.html
http://www.citiesjournal.com/top-15-best-food-cities-in-the-world/
http://www.travelchannel.com/interests/food-and-drink/photos/best-food-and-wine-cities
http://www.frommers.com/slideshows/818551-the-world-s-best-street-food-12-top-cities#slide837006
http://travel.cnn.com/explorations/eat/worlds-best-cities-foodies-504544
http://www.usatoday.com/story/travel/destinations/10greatplaces/2014/06/27/worlds-best-foodie-cities/11403467/
http://www.virtualtourist.com/press-center/top-ten-street-food-cities
http://www.fiked.com/20-most-relaxing-places-in-the-world/
http://www.huffingtonpost.com/2014/08/31/traveling-alone-10-relaxi_n_3202180.html
http://www.lonelyplanet.com/blog/2013/02/26/lonely-planet-travellers-choice-the-top-destinations-of-2013-part-3/
http://www.smartertravel.com/photo-galleries/editorial/eight-vacations-for-ultimate-relaxation.html?id=102&all=1
http://www.tripadvisor.com/TravelersChoice-Beaches-cTop-g1
http://travel.usnews.com/Rankings/Best_Beaches_in_the_World/
http://travel.nationalgeographic.com/travel/top-10/beaches/
http://www.thrillist.com/travel/nation/the-25-best-beaches-in-the-world
http://www.travelandleisure.com/worlds-best-spas-2014-winners-list
http://www.elle.com/culture/travel-food/g7712/best-spas-in-the-world/?slide=2

Organizing Data

- Just one factor
- Routes that include two factors, three factors, etc. (i.e. Relaxation and Food)

Thailand
Belize
Barcelona
Austin, TX
Australia
Vietnam
Istanbul
Jaipur, Ind
London
Marrakech
Mexico
Paris
Tel Aviv, Is

http://the1andonlyrestaurant.com/wp-content/uploads/2013/04/Thai-Food.jpg

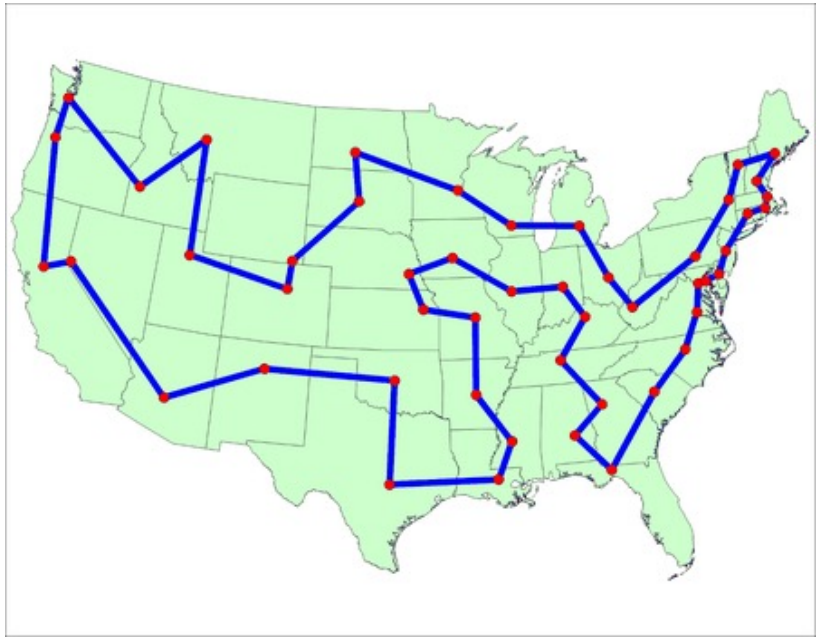
http://www.hiltonhawaiianvillage.com/assets/img/discover/oahu-island-activities/HHV_Oahu-island-activities_Content_Beaches_455x248_x2.jpg

Picking the Path: Traveling Salesman Approach

- Finds shortest path that visits each city once and returns to the original city
- NP-Hard
- O(n^2*2^n) or O(n^4, n^3*2^n)

http://arxiv.org/abs/1412.2437

http://support.sas.com/documentation/cdl/en/ornoaug/65289/HTML/default/viewer.htm#ornoaug_optnet_examples07.htm



Picking the Path: Other Approach

1. Compile list of cities by categories (Relaxation, Adventure, Food, etc.)
2. Break list of cities into regions around the world (such as by continents)
3. Pick a city in each region, once each region has at least one “destination” restart in regions until target number is reached O(n) n being # of stops
4. Connect the dots between the stops within the region, then stops between regions until there is one continuous path with each stop visited once.

General Steps of Algorithm

1. Sort the data for each category (count duplicates, then sort)
2. Create different subcategories across the categories
3. Pick n (with n being the number of stops) destinations based on the given list of wants (i.e Food, Adventure and History, Relaxation and Food and History, etc.)
4. Determine a path from the starting to city that visits each city once and then returns to the origin city in the end

Spelling Correction Algorithms

Spelling Correction Algorithms Why?

- Given the limited amount of data/words/letters present, it seems realistic for amateurs in algorithm study to entirely grasp and understand the process, steps, and subtleties of several spelling correction algorithms.
- Also – spelling correction algorithms are obviously applicable to daily life/regular technology use.
- Even decades after their creation, developers are still working on refining and improving spelling correction algorithms. Improvements to older algorithms are constantly being made, especially given context, pronunciation, and language based additions to existing algorithms.
- There's a variety of different methods to go about doing basically the same thing, making it interesting to compare the pros/cons and differences of existing popularly used algorithms in the field.

The Basics

- Verifiers vs Correctors
- Operations of spell-checking:
  - Document normalization
  - Spelling verification
  - Spelling correction: VARIETY OF TECHNIQUES COME INTO PLAY HERE
    - Algorithm for handling morphology
    - N-gram Model
- Context-Based Correction Algorithms
  - Andrew Golding and Dan Roth's Winnow-based correction algorithm

Topics to Examine

- Three major problems faced when correcting words in text: (1)
  - Non-word detection error: “pattern-matching and n-gram analysis techniques… for detecting strings that do not appear in a given word list”
  - Isolated-Word error correction: “general and application-specific spelling correction techniques”
  - Context-dependent work correction: “natural-language-processing tools or statistical-language models”
- Binary Neural Approach towards spelling correction: (2)
  - phonetic matching
    - Soundex
  - supervised learning
  - associative matching in the AURA neural system
    - Hybrid approach to handle insertion, deletion, substitution, and transposition (double substitution) errors
    - Synthesis of n-gram, soundex, hamming distance scores
- Benefits of restricting dictionaries to a certain number of entries (3)
  - Trade-off of restricting dictionary – sacrificing precision for accuracy
  - Domain-sensitivity

Sources: (1) (2) (3)

Going Forward

- Choose exactly which correction algorithms to compare and focus on. Probably narrow number of algorithms down to 3 to compare.
- Possibly limit research to algorithms working with English.
- Organize current body of research into a cohesive whole, and identify main points of focus.
- Identify main issues, limitations, and possible avenues for improvement still present in this field. (What we concluded after examining all current research)

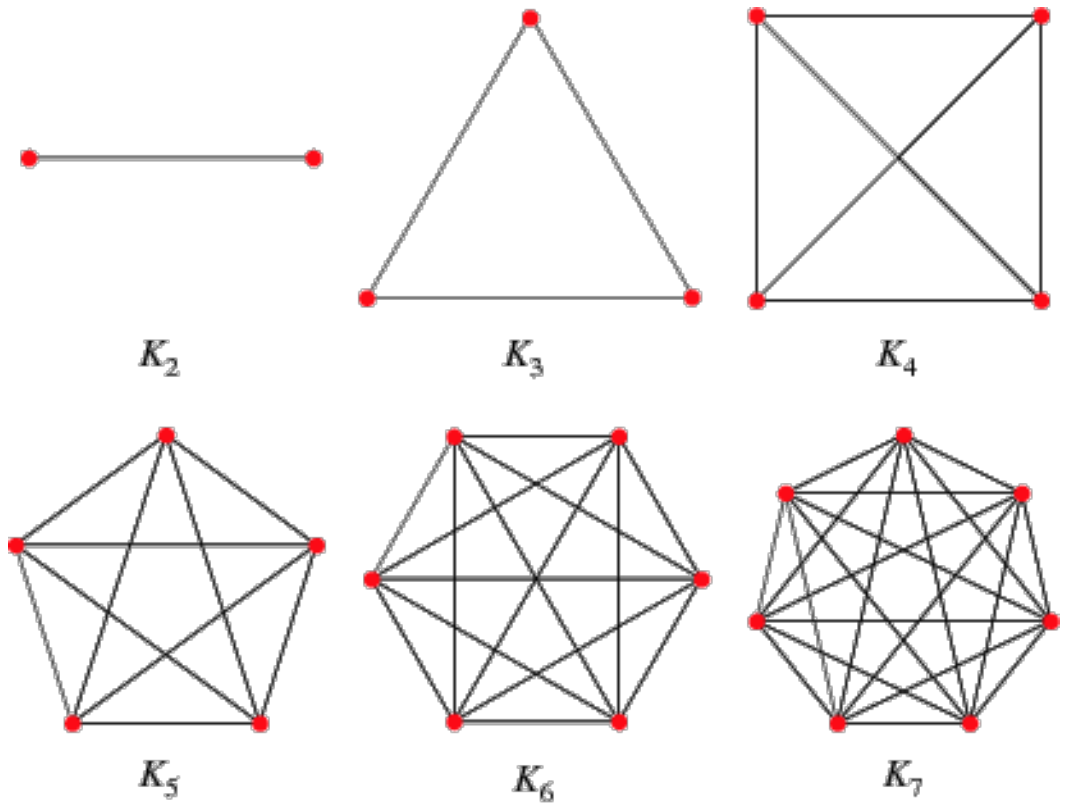
Optimization of Shortest Path Algorithms

Special Cases

- The concept of this optimization is dependent upon special cases as the relate to the number of edges in the graph. In general, my goal was to generalize a relation between the number of edges in a graph and the length of the shortest path. My thought was that less checks could possibly be made if the algorithm limited itself to only certain possible distances.

Inspiration

- In an undirected graph of n nodes, the maximum number of possible edges is $\frac{n(n-1)}{2}$
- When a graph of n nodes has $\frac{n(n-1)}{2}$ edges we know that the length of the shortest path k from node u to v must have a value of $k = 1$, as every node in the graph is connected to each other
- An algorithm could then have a special case set aside for when the graph has $\frac{n(n-1)}{2}$ edges
- I wanted to see if other such cases existed

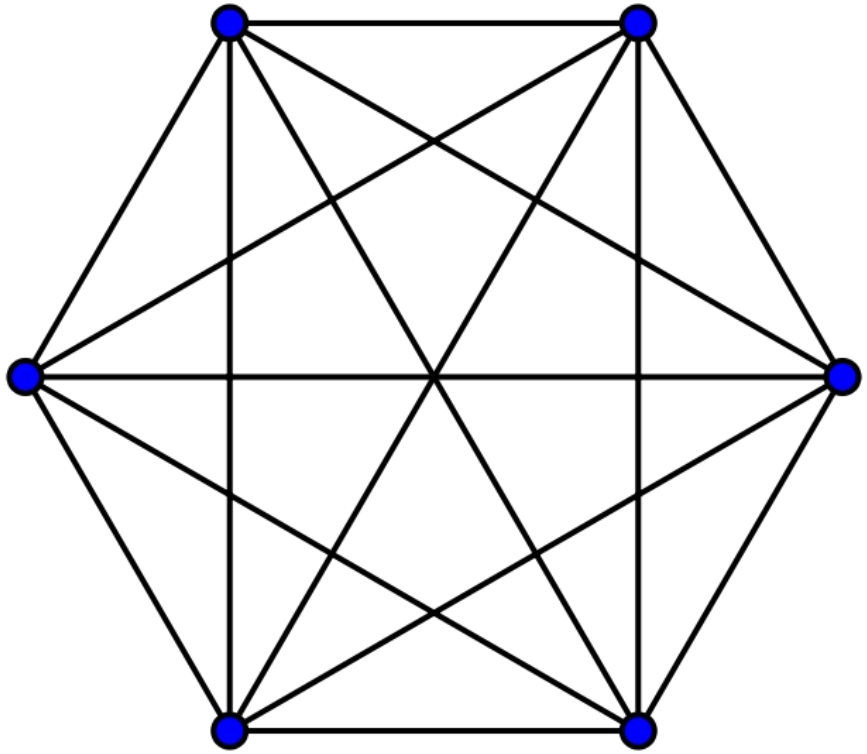


Some Special Cases

Number of Edges in Respect to n	Possible Lengths k of Shortest Path
$\frac{n(n-1)}{2}$	1
$\frac{n(n-1)}{2} - c, 1 \leq c < n-1$	$1, 2$
$\frac{n(n-1)}{2} - (n-1)$	$1, 2, 3$
$\frac{n(n-1)}{2} - n$	$1, 2, 3$
$n+1$	$1 \dots n-2$
n	$1 \dots n-2$
$n-1$	$n-1$

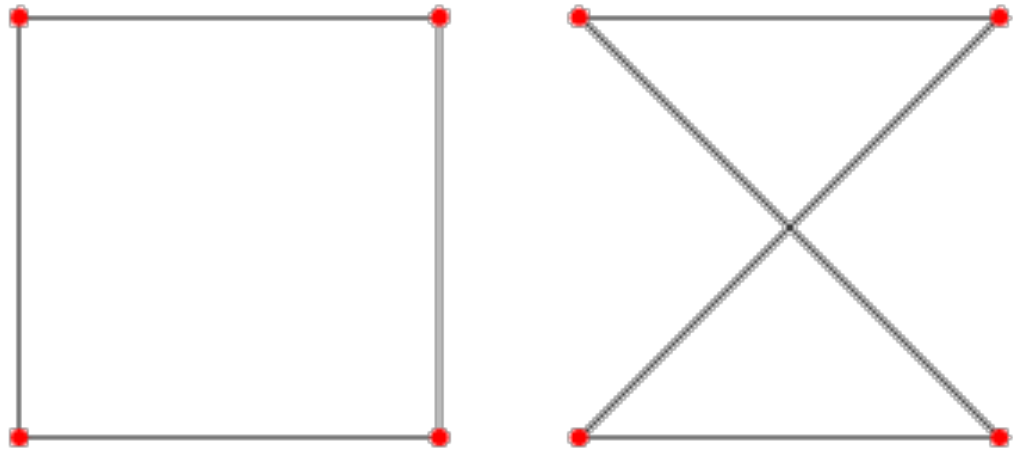
Node Connections

- To bring actual Nodes into the algorithm, I also anticipated that similar cases would exist for the number of edges connected to each node
- For a graph of n nodes, the maximum number of edges which can connect to a single node is $n - 1$
- If trying to find the shortest path from node u to node v, if either u or v is connected to $n - 1$ edges then the shortest path will be of distance $k = 1$



Connections Example

- A graph of n nodes, and edge count $2n - 4$
- Nodes u and v both connect to $n - 2$ edges, neither of which connect to the other
- As neither u and v are not connected, we can determine that the edges connected to each are $(n - 2) \times 2 = 2n - 4$ distinct edges, which is equal to the total number of edges
- This would mean that the length of the shortest path $k = 2$, as no edges are unaccounted for, and both u and v are connected to every other node as they are each connecting to $n - 2$ nodes, or 1 less node than the maximum possible for a single node, and we know that they are not connected to each other



Goals Moving Forward

- Find more cases for number of edges connected to a node
- Further generalize cases by creating ranges
- Try to tie both concepts together (total number of edges and number of edges at each node)
- Somehow break down the problem through these concepts (Recursively, Divide-and-Conquer, etc.)
- Abstract to more types of graphs if possible (Directed, Weighted, etc.)
- Figure out where concepts fit best (Calculation, Verification, etc.)