Graph Guided Genetic Algorithms
Kyle Sweeney
Understanding Genetic Algorithms
Part 1: the Problem

\[ f(x) \]

\[ X = (12.4, \text{"No-go"}, 22, ...) \]
Understanding Genetic Algorithms
Part 2: Just Copy Nature

- Solution == DNA
  - e.g (12.4, “No-go”, 22, ...)

- Fitness function
  - A method for determining how “good” a solution is
  - Can be a score

- Breeding Multiple Generations
  - Combine DNA in different ways
  - E.g (12.4, “No-go”, 22, ...) + (-3, “Go”, “9”) == 2^n possible combinations

- Survival of the Fittest
Application: Genetic Engineering

CGT: 0.123
CGC: 0.334
CGA: 0.462
CGG: 0.111

CGT: 0.121
CGC: 0.235
CGA: 0.461
CGG: 0.213
Solution: Genetic Algorithms to solve Genetic Engineering Problems

- “DNA”: the specific Codon encodings which generate the same Protein
- Fitness Function: $\sum |\text{MinMax(Source)} - \text{MinMax(Target)}|$
- Breeding:
  - Zip Children: for each position, alternate between taking from parents
  - Random Children: randomly choose from parents
  - Half and Half: first half one parent, second half the other
  - ....
Scoring Function in Detail

“Area” Between Curves

+Additional Penalty if Orig and New have different slope directions
How Graphs Made things Different

- *Graph Based Evolutionary Algorithms* by Bryden K.M. et al
- Take a graph and place a potential solution on each vertex
  - The only mating partners for that vertex are its neighbors
  - Choose from potential mates who to mate with
- Only replace parent if child is better than parent
graph = new graph([ list of random permutations of start ])
for i in range 50:
    for v in graph.nodes():
        children = []
        for n in graph.neighbors(v):
            children += breed(n,v,10)
        sort(children)
        if children[0].score < v.score:
            graph.replace(v,children[0])
return sort(graph.nodes())[0]

complexity: $O(V^2B)$ where $O(B)$ is time complexity of Breeding algorithm, in this case $O(N)$ where N is length of solutions.
New Pseudocode

```python
    graph = new graph([ list of random permutations of start ])
    for i in range 50:
        tupes = []
        for v in graph.nodes():
            nodetupes += (graph, v)
        with pool(K) as p:
            replaces = p.map(vertex_prog, nodetupes)
            for x in replaces:
                if x.child not in graph:
                    graph.replace(x.parent, x.child)
    return sort(graph.nodes())[0]

    def vertex_prog(graph, vert):
        children = []
        for neighbor in graph.neighbors(vert):
            children += breed(vert, neighbor)
        children.sort()
        if children[0].score < vert.score:
            return (vert, children[0])
```

New Complexity

- Complexity is essentially the same as before, however with a $\frac{1}{K}$ factor, reducing the runtime by the number of processes being ran.

- For each node -> for each neighbor: $O(V^2)$

- For each node-node pair: $O(B)$

- Total: $O(\frac{V^2B}{K})$
Graphs - variable nodes

- **Caveman Graph**
  - $K$ connected $Q$ cliques in a ring

- **Windmill Graph**
  - $Q$ cliques with all nodes connected to a central node

- **Erdos-Renyi aka GNP**
  - For each possible edge between $N$ nodes has a probability $P$ of existing

- **Watts-Strogatz**
  - $N$ nodes, $K$ edges, with probability $P$ each edge is re-wired
  - Start with ring of $N$ nodes, connect to nearest $K$ neighbors, rewire
Implementation Details

- **Software Libraries**
  - Python3 targeted
  - Graphs generated and manipulated via NetworkX
  - Pypy3 used to execute the program

- **Graph Manipulation Technique**
  - Multi-threaded, Each vertex being processed by a thread

- **Data Collection**
  - 4 Specimens being compared against an e.coli strain
    - *caenorhabditis elegans, Mus musculus, Homo sapien, Saccharomyces cerevisiae*
  - 10 runs averaged in score and time elapsed

- **Context**: First gen solution could have score of 270,000+
Watts-Strogatz: 10 Edges, 10% - E.coli vs Brewer’s Yeast

![Graph showing the relationship between Processes in Pool and Average Time (S) for Watts-strogatz 40,10,0.1.](image-url)
Watts-Strogatz: 10 edges, 10%, 16 Procs-
E.coli vs Brewer’s Yeast
Windmill Graph: 4 Cliques, 16-Procs - E.coli vs Brewer’s Yeast

Windmill Graph

Windmill Graph

Windmill Graph
Erdos-Renyi: 10%, 16-Procs - E.coli vs Brewer’s Yeast
It’s all about Connection - Windmill, 16-Proc

<table>
<thead>
<tr>
<th>Genes</th>
<th>Windmill Graph - 4,25</th>
<th>Windmill Graph - 25,4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Time</strong></td>
<td><strong>Score</strong></td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>stddev</td>
</tr>
<tr>
<td>5</td>
<td>1408.068770</td>
<td>8.699657</td>
</tr>
<tr>
<td>4</td>
<td>1423.719129</td>
<td>10.158205</td>
</tr>
<tr>
<td>3</td>
<td>1439.245480</td>
<td>26.072518</td>
</tr>
<tr>
<td>2</td>
<td>1404.060382</td>
<td>2.853025</td>
</tr>
</tbody>
</table>
## Caveman - 4,25

<table>
<thead>
<tr>
<th>Genes</th>
<th>Time</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>stdev</td>
</tr>
<tr>
<td>5</td>
<td>1397.356041</td>
<td>3.315249</td>
</tr>
<tr>
<td>4</td>
<td>1417.976462</td>
<td>3.782410</td>
</tr>
<tr>
<td>3</td>
<td>1421.361858</td>
<td>3.749360</td>
</tr>
<tr>
<td>2</td>
<td>1406.844888</td>
<td>12.047170</td>
</tr>
</tbody>
</table>

## Caveman - 25,4

<table>
<thead>
<tr>
<th>Genes</th>
<th>Time</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>stdev</td>
</tr>
<tr>
<td>5</td>
<td>365.723654</td>
<td>6.032161</td>
</tr>
<tr>
<td>4</td>
<td>363.854922</td>
<td>0.903344</td>
</tr>
<tr>
<td>3</td>
<td>366.806325</td>
<td>2.963881</td>
</tr>
<tr>
<td>2</td>
<td>375.012293</td>
<td>3.576178</td>
</tr>
<tr>
<td>Watts-strogatz - 40,20,0.1</td>
<td>Watts-strogatz - 40,10,0.1</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Time</strong></td>
<td><strong>Score</strong></td>
<td><strong>Time</strong></td>
</tr>
<tr>
<td>Genes</td>
<td>mean</td>
<td>stdev</td>
</tr>
<tr>
<td>5</td>
<td>471.01908</td>
<td>1.914677</td>
</tr>
<tr>
<td>4</td>
<td>477.796109</td>
<td>5.220690</td>
</tr>
<tr>
<td>3</td>
<td>473.214193</td>
<td>2.311429</td>
</tr>
<tr>
<td>2</td>
<td>470.439868</td>
<td>6.602775</td>
</tr>
</tbody>
</table>
Generalized Results

- Diminishing Returns Relationship between Time and Quality of Score
- Rather large Variability in solutions between different Graphs
- Graphs overall impact runtime by changing number of possible breeding pairs
- Watts-Strogatz Seems to be ideal in having lowest runtime and lower scores
Conclusions

- Simple Mapping of distributing work to different tasks may decrease gains as a result of increased overhead managing processes
- Not all graphs are ideally suited to any Genetic Algorithm problem