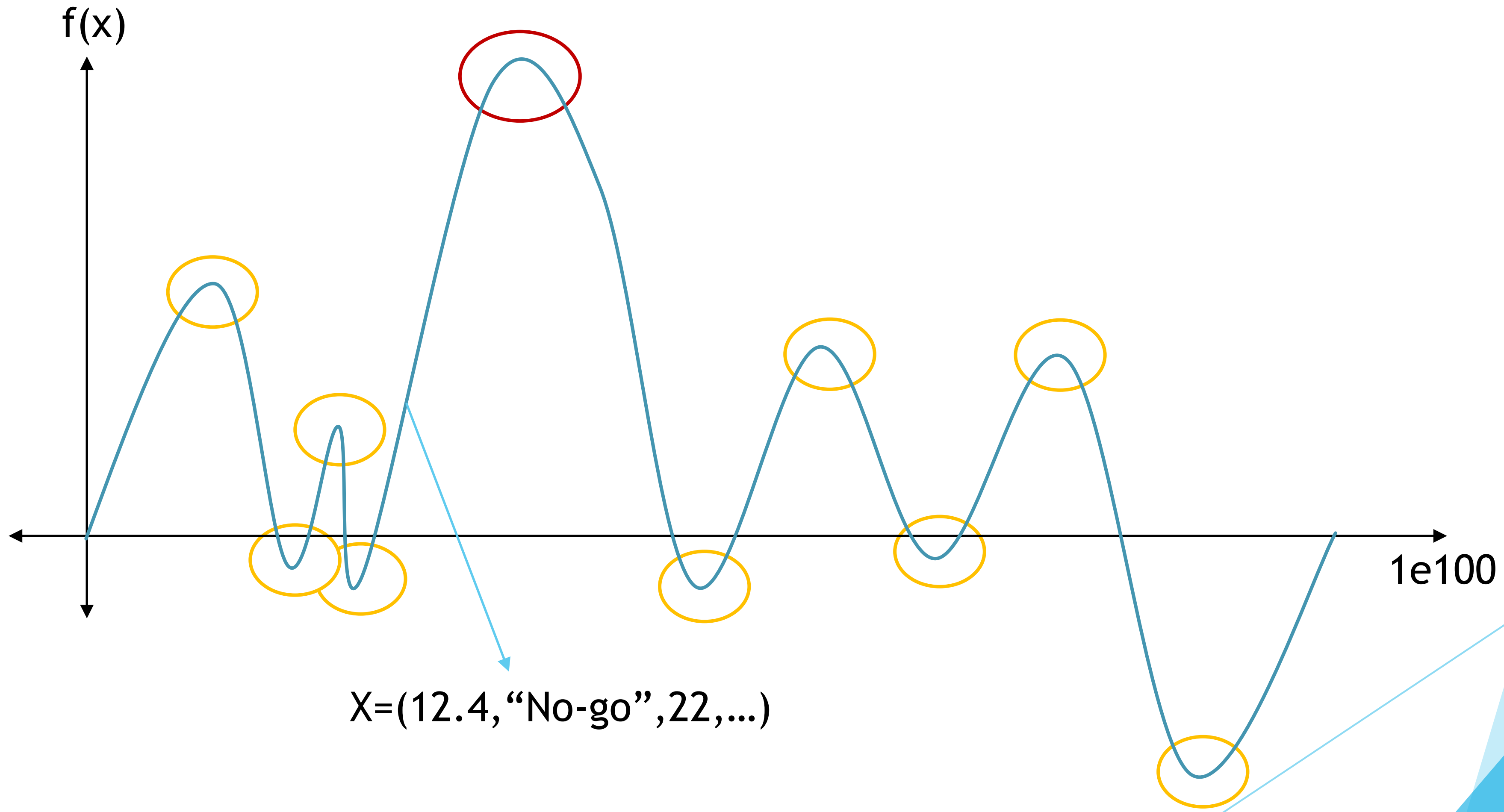
Graph Guided Genetic Algorithms

Kyle Sweeney

Understanding Genetic Algorithms

Part 1: the Problem

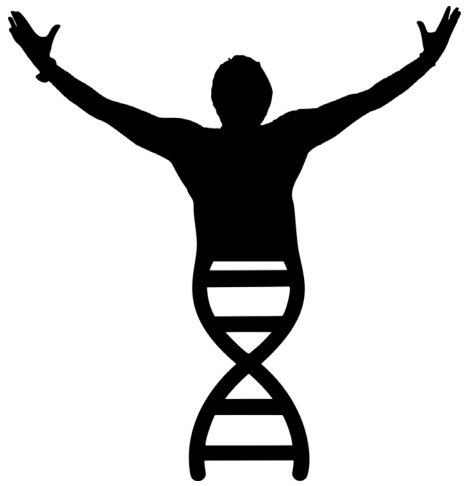


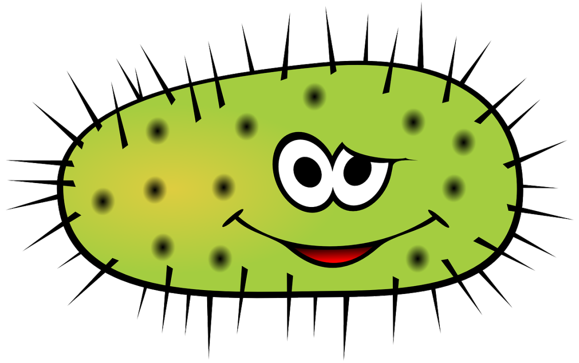
Understanding Genetic Algorithms

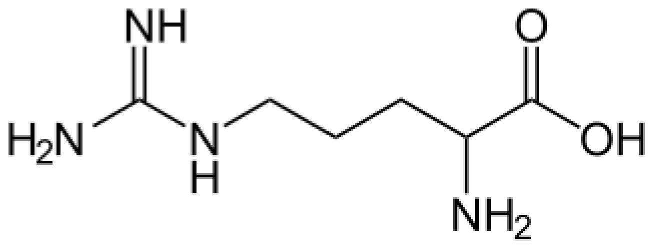
Part 2: Just Copy Nature

- 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^X 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

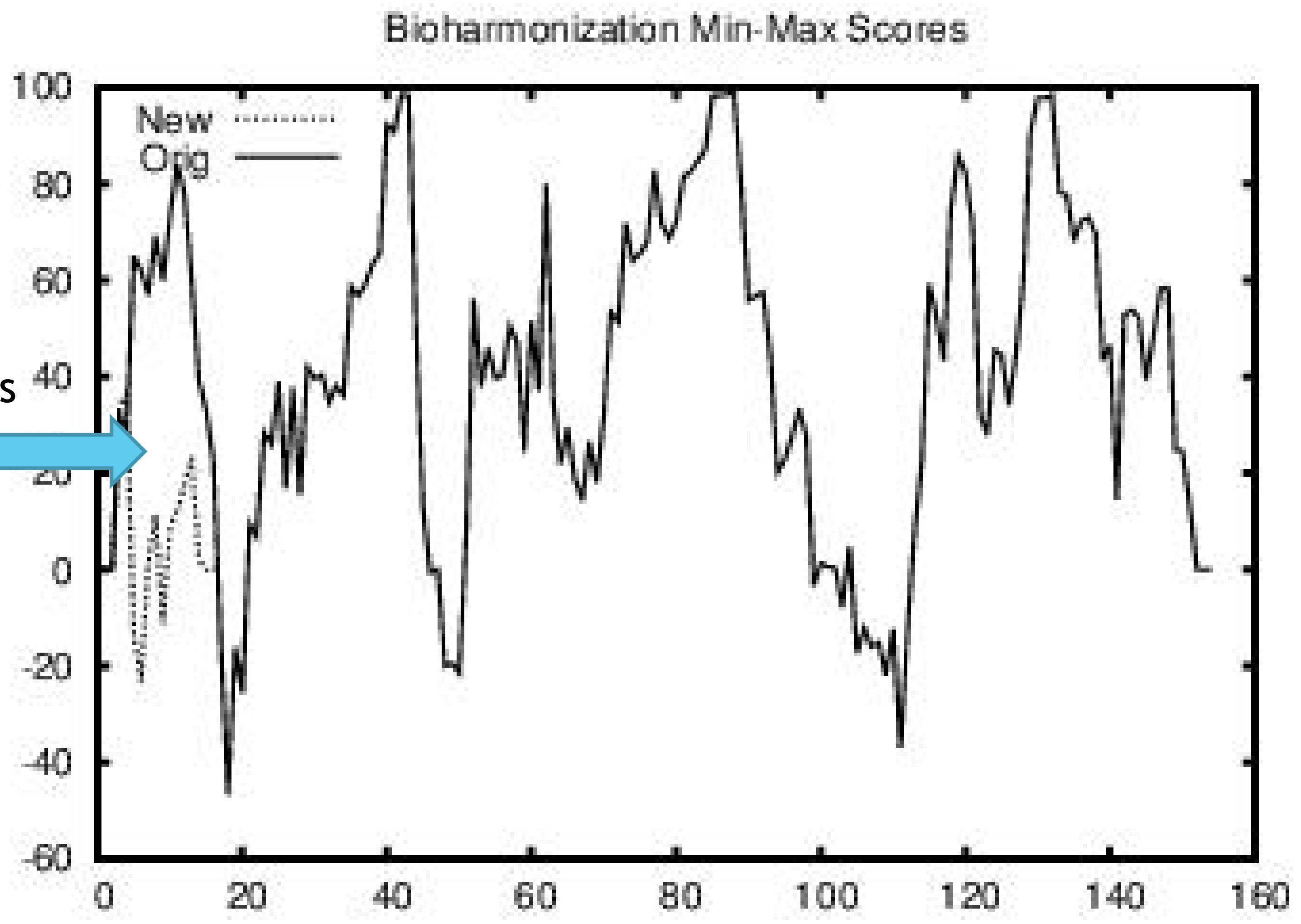
Solution: Genetic Algorithms to solve Genetic Engineering Problems

- "DNA": the specific Codon encodings which generate the same Protein
- Fitness Function: $\sum |\mathrm{MinMax}(\mathrm{Source}) - \mathrm{MinMax}(\mathrm{Target})|$
- 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

Old-Pseudocode

```
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

```
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: $O(\frac{V^2B}{K})$

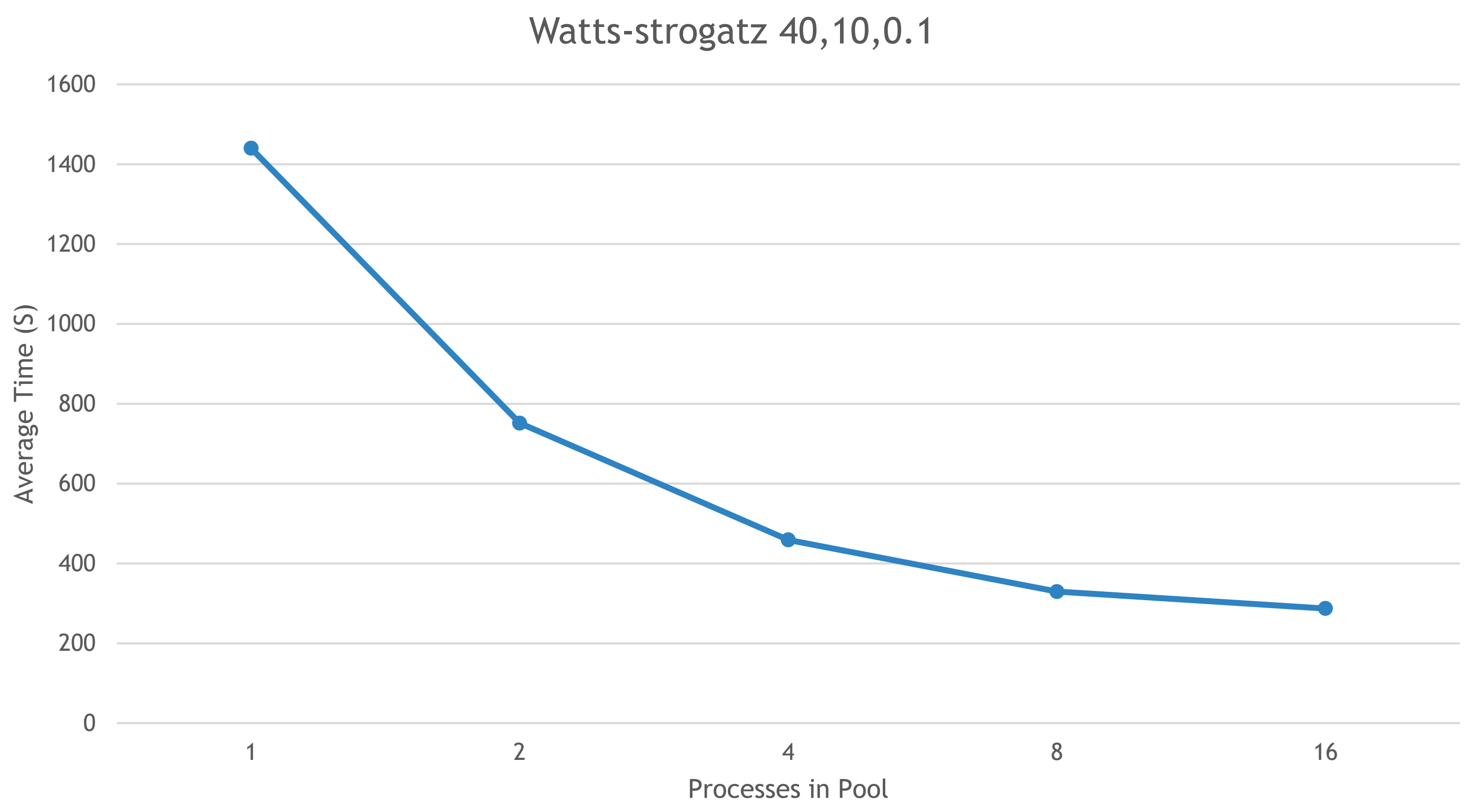
Graphs – variable nodes

- 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

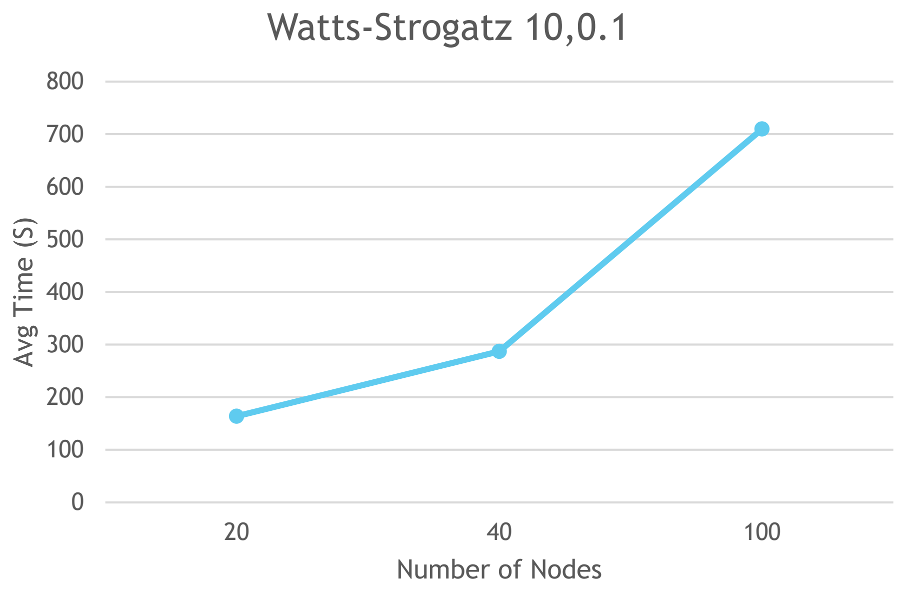
Implementation Details

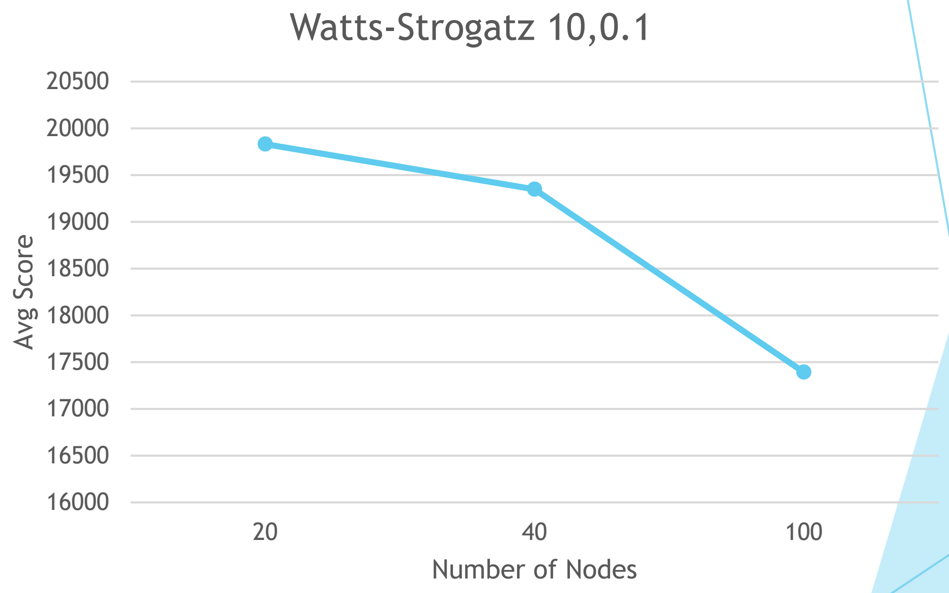
- Software Libraries
  - Python3 targeted
  - Graphs generated and manipulated via NetworkX
  - Pypy3 used to execute the program
- Graph Manipulation Technique
  - Multi-threaded, Each vertex being processed by a thread
- Data Collection
  - 4 Specimens being compared against an e.coli strain
    - 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

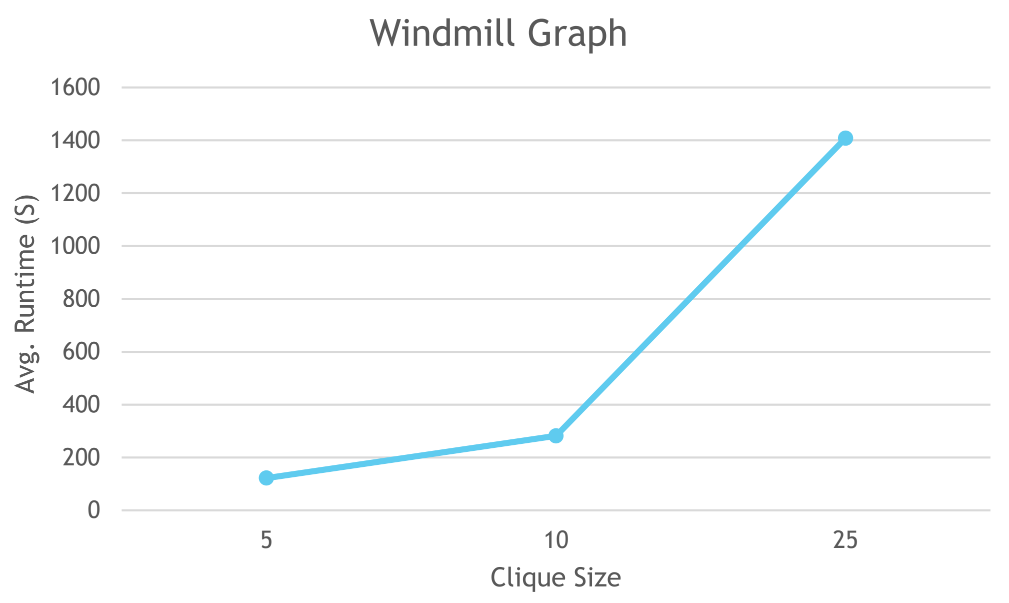


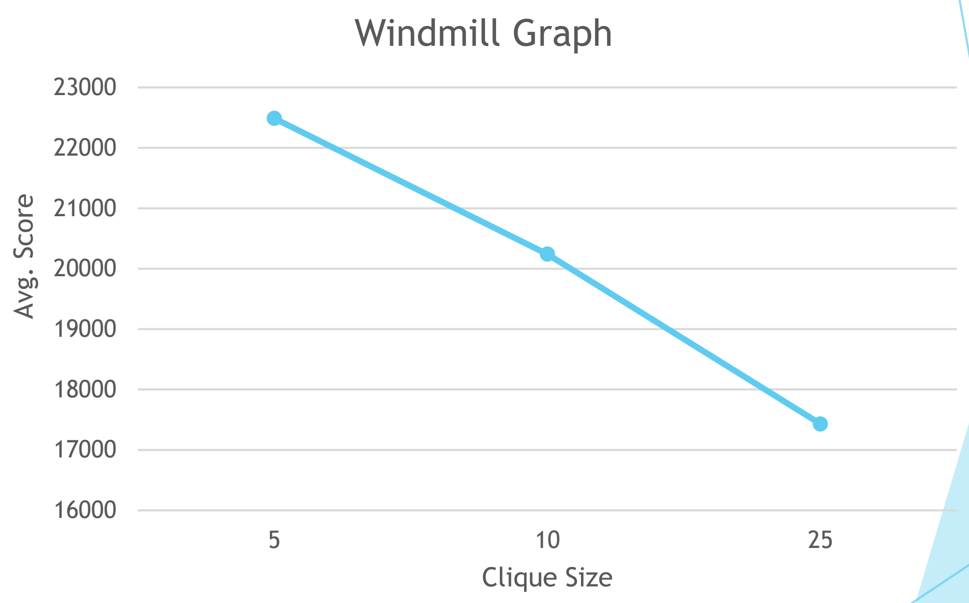
Watts-Strogatz: 10 edges, 10%, 16 Procs- E.coli vs Brewer’s Yeast



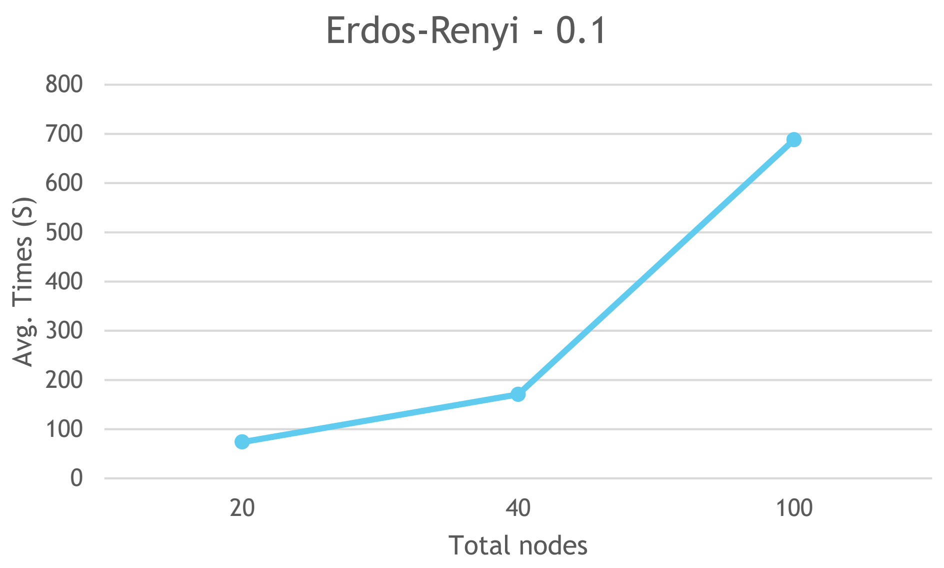


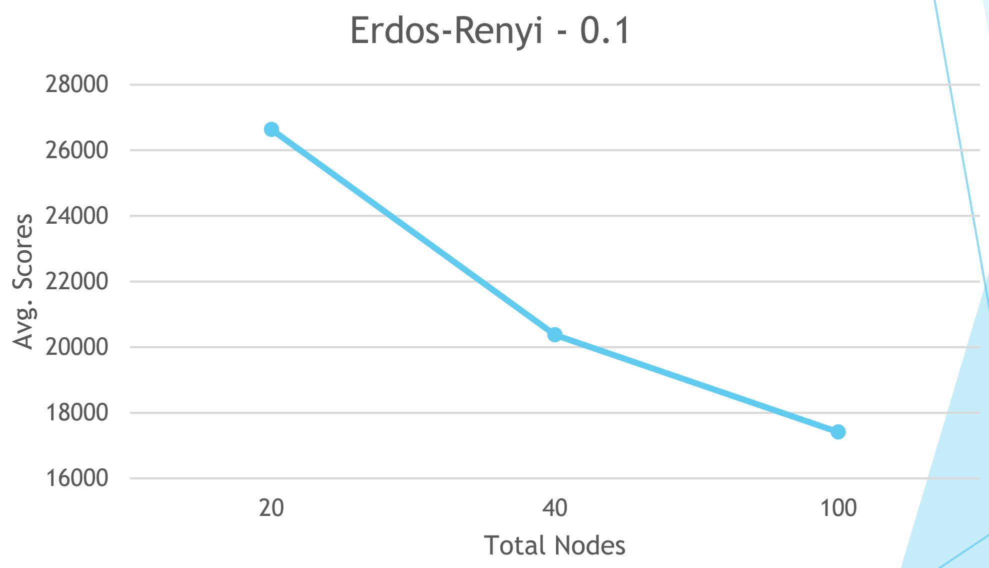
Windmill Graph: 4 Cliques, 16-Procs – E.coli vs Brewer’s Yeast





Erdos-Renyi: 10%, 16-Procs – E.coli vs Brewer’s Yeast





It’s all about Connection – Windmill, 16-Proc

Windmill Graph - 4,25				
	Time		*Score*	
Genes	mean	stdev	mean	stdev
5	1408.068770	8.699657	17428.424	1111.514012
4	1423.719129	10.158205	21811.427	1624.714577
3	1439.245480	26.072518	18648.071	841.697471
2	1404.060382	2.853025	20139.532	1090.583855

Windmill Graph - 25,4				
	Time		*Score*	
Genes	mean	stdev	mean	stdev
5	469.180785	9.633604	20847.582	1406.669398
4	495.512529	3.884894	25048.655	1934.238458
3	466.830144	13.005787	23982.811	1560.534256
2	456.034971	2.652946	26518.976	2373.275543

It’s all about Connection – Caveman, 16-Proc

Caveman - 4,25					**Caveman - 25,4**				
	Time		*Score*			*Time*		*Score*	
Genes	mean	stdev	mean	stdev	Genes	mean	stdev	mean	stdev
5	1397.356041	3.315249	17379.616	559.891502	5	365.723654	6.032161	24305.905	1874.371394
4	1417.976462	3.782410	21811.167	1155.276183	4	363.854922	0.903344	29282.458	1242.619206
3	1421.361858	3.749360	19056.142	905.003859	3	366.806325	2.963881	25348.137	1583.727080
2	1406.844888	12.047170	21306.424	1237.06641	2	375.012293	3.576178	27629.355	957.840066

It's All about Connection – Watts-strogatz, 16-Procs

watts-strogatz - 40,20,0.1					watts-strogatz - 40,10,0.1				
	Time		*Score*			*Time*		*Score*	
Genes	mean	stdev	mean	stdev	Genes	mean	stdev	mean	stdev
5	471.01908	1.914677	17617.438	1750.497095	5	287.126895	4.592964	19348.722	1898.003673
4	477.796109	5.220690	23033.722	1135.957858	4	288.292479	2.427174	23230.660	880.696387
3	473.214193	2.311429	19182.586	1310.839698	3	288.253836	2.194757	20261.781	1514.381834
2	470.439868	6.602775	21481.47	1274.895345	2	284.449048	1.707648	22243.756	1684.05801

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