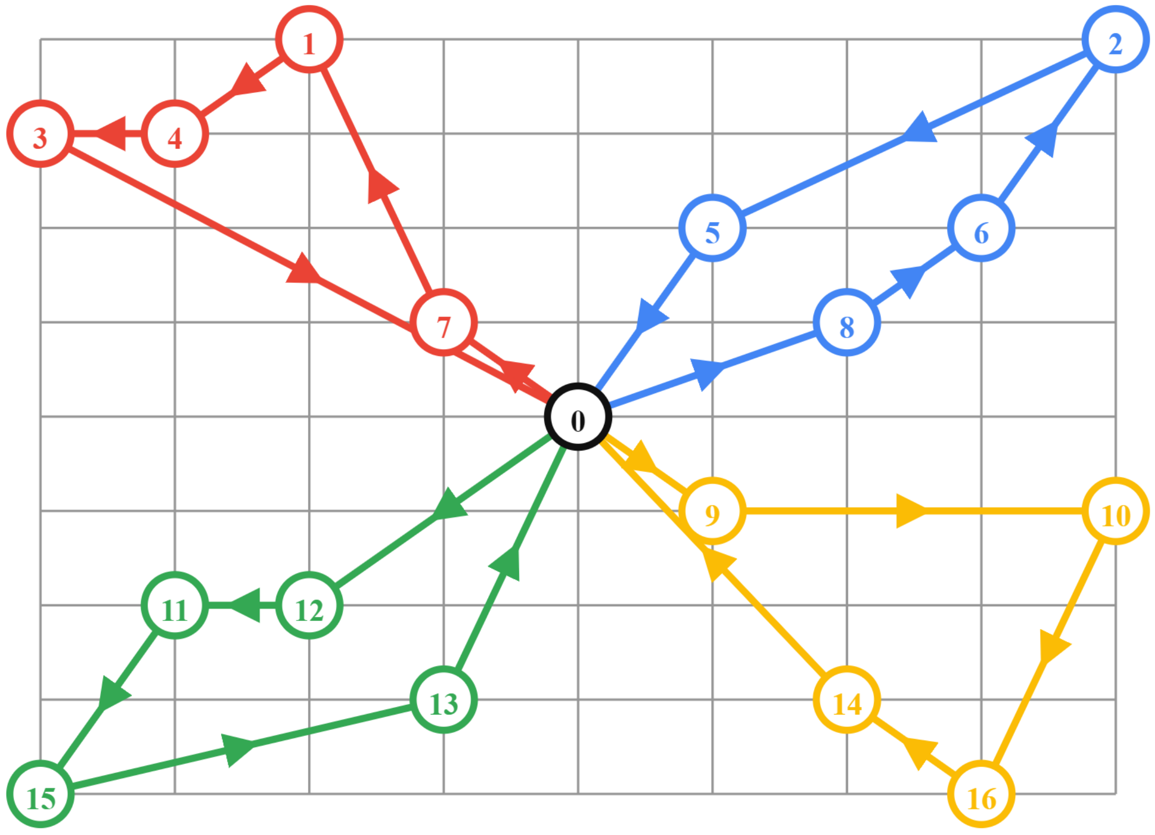
Adrian Low

Multi-Agent Genetic Algorithm for Vehicle Routing Problem

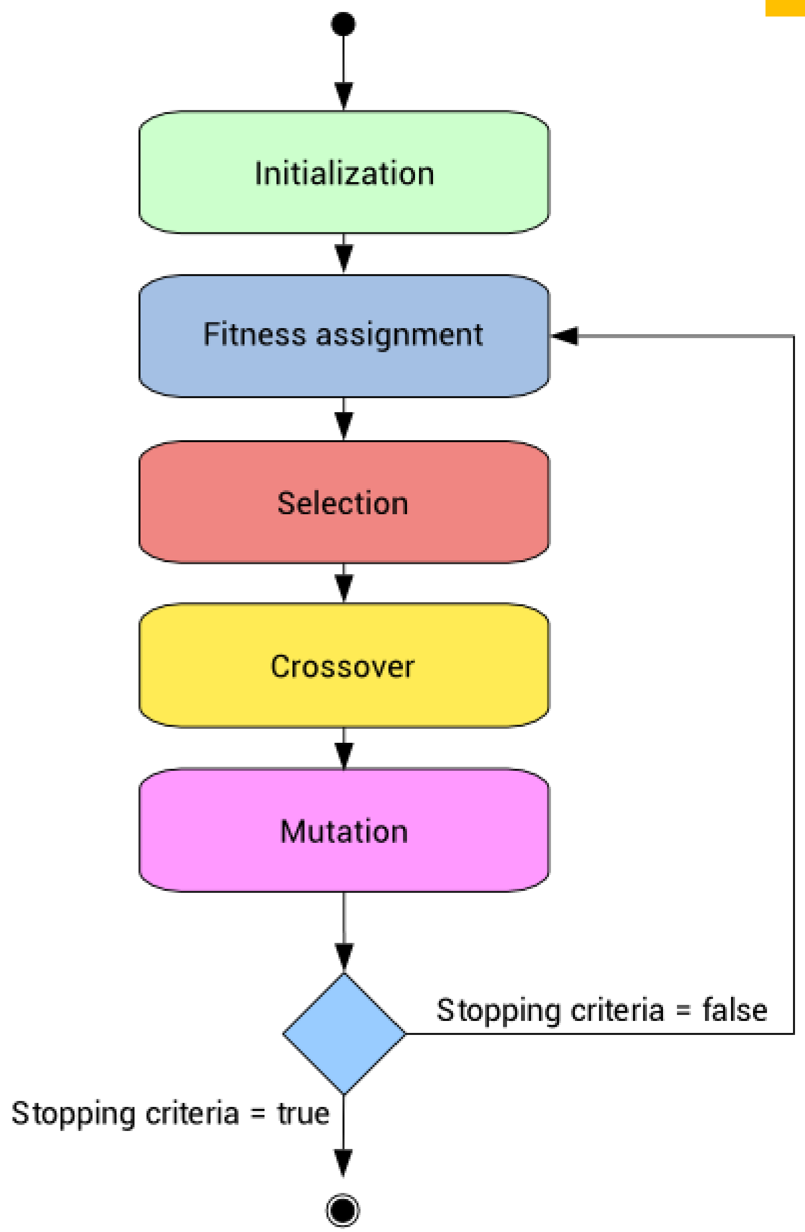
Vehicle Routing Problem

- Problem: to find optimal routes for multiple vehicles visiting a set of locations
- Extension of the Traveling Salesman Problem (TSP)
- Distance, Carrying Capacity, Time Windows, Multiple Depots, etc.



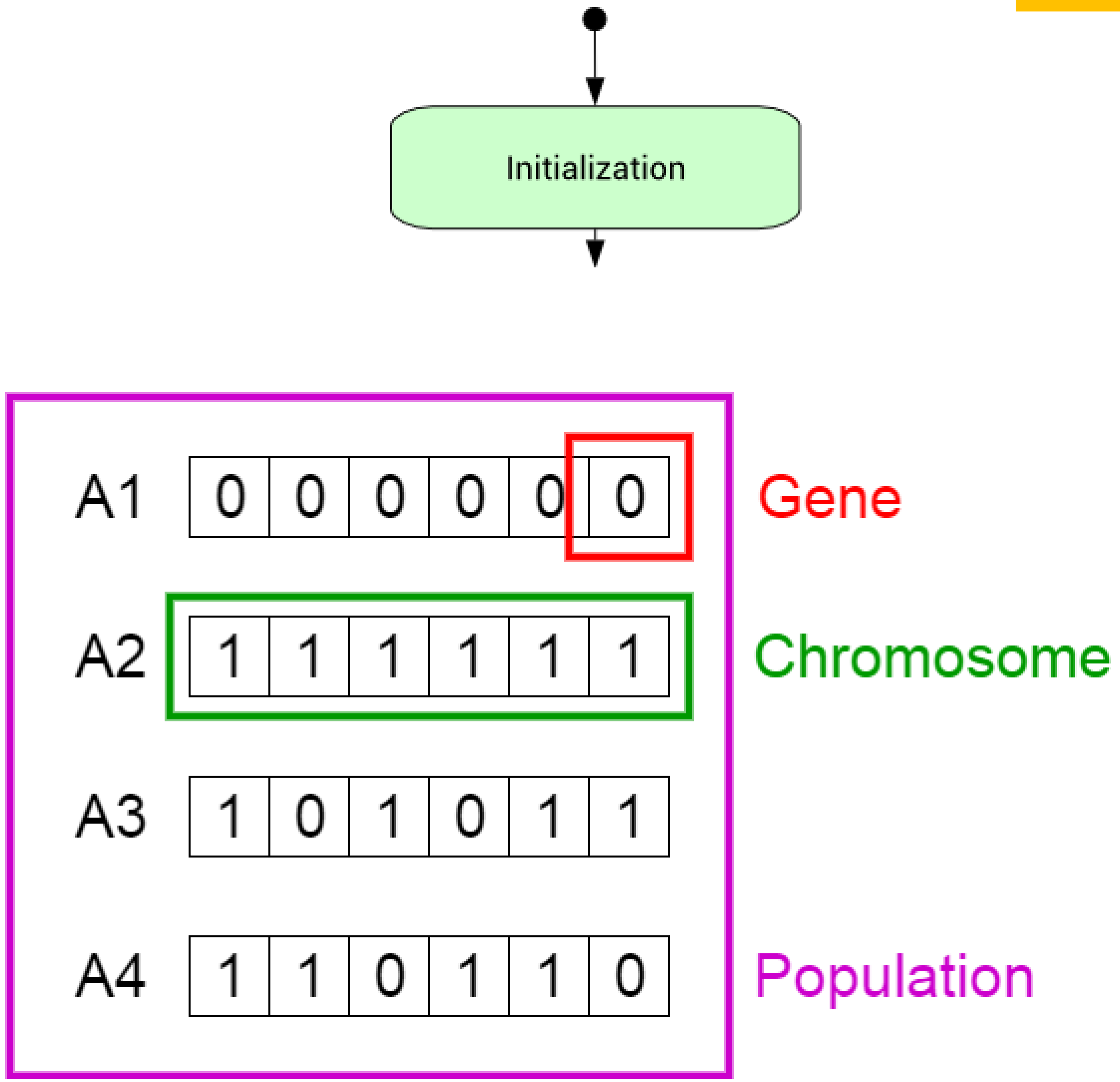
Genetic Algorithms (GA)

- A search heuristic based on evolution and natural selection in Nature
- Fittest individuals reproduce next generation
- 5 phases



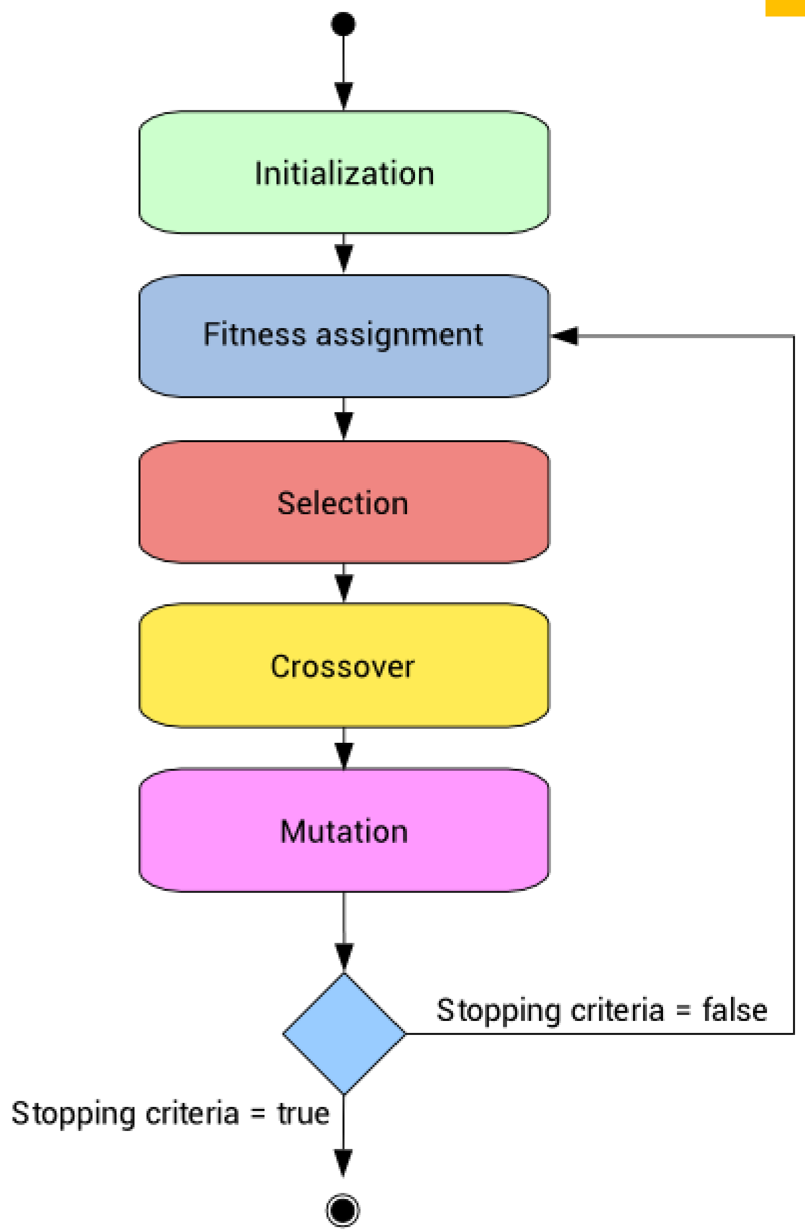
GA: Initialization

- Creates a population of random chromosomes (in a string)
- Each chromosome represents a potential solution
- Genes in the chromosome represent a set of parameters



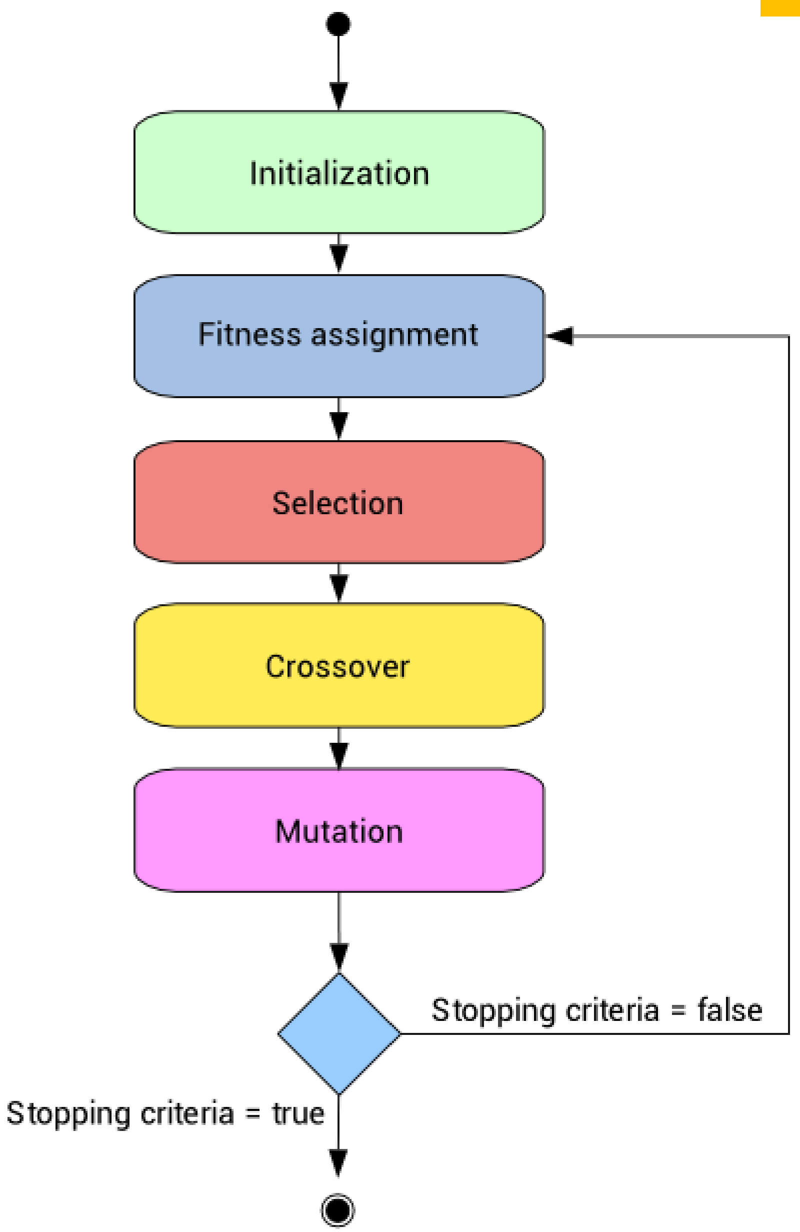
GA: Fitness, Selection, Crossover

- A fitness function gives a fitness score to each individual
- Selection chooses individuals to reproduce based on fitness
- Crossover exchanges genes of parents to produce offspring



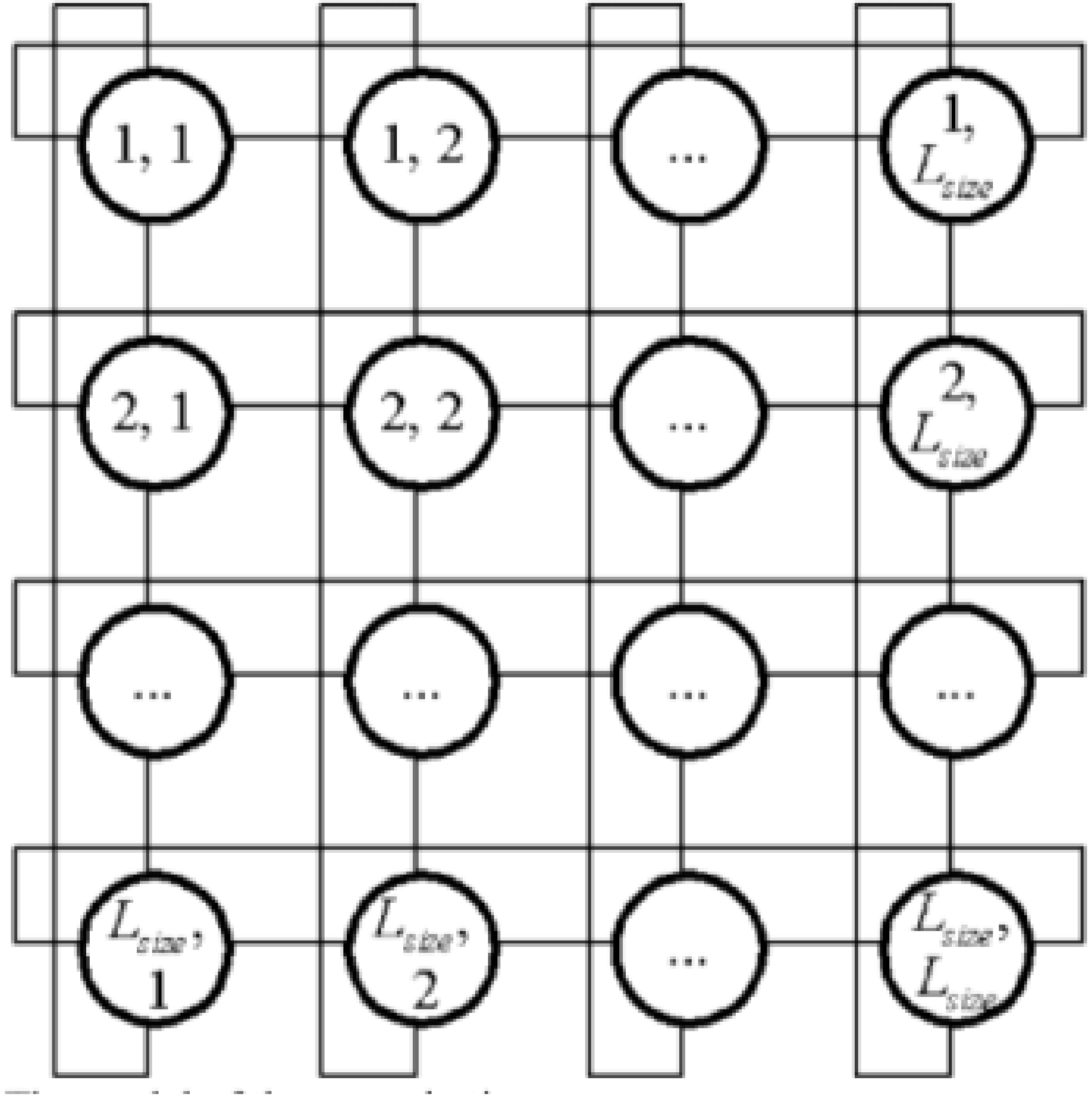
GA: Mutation, Termination

- Mutation selects individuals at a low probability and modifies genes
- After mutation, fitness gets reevaluated
- Selection, Crossover, and Mutation repeats until a stopping point



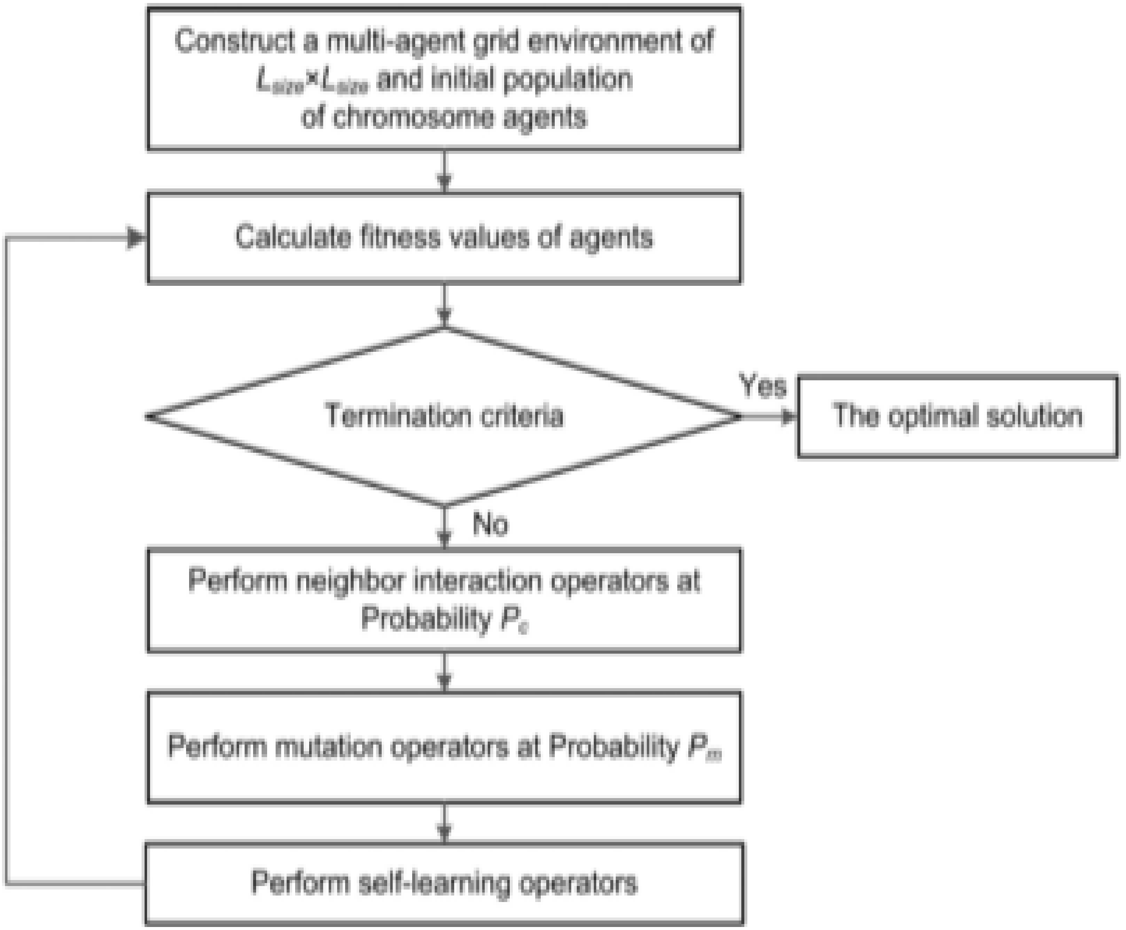
Multi-Agent Genetic Algorithm (MAGA)

- Combines Agent-based systems and Genetic Algorithms
- Agent – an autonomous computational individual with properties and actions
- Agents live in a lattice environment and interacts with its neighbors in a Von Neumann neighborhood
- Each agent represents a potential solution and has energy (fitness) based on that solution
- Based off Cellular Genetic Algorithms



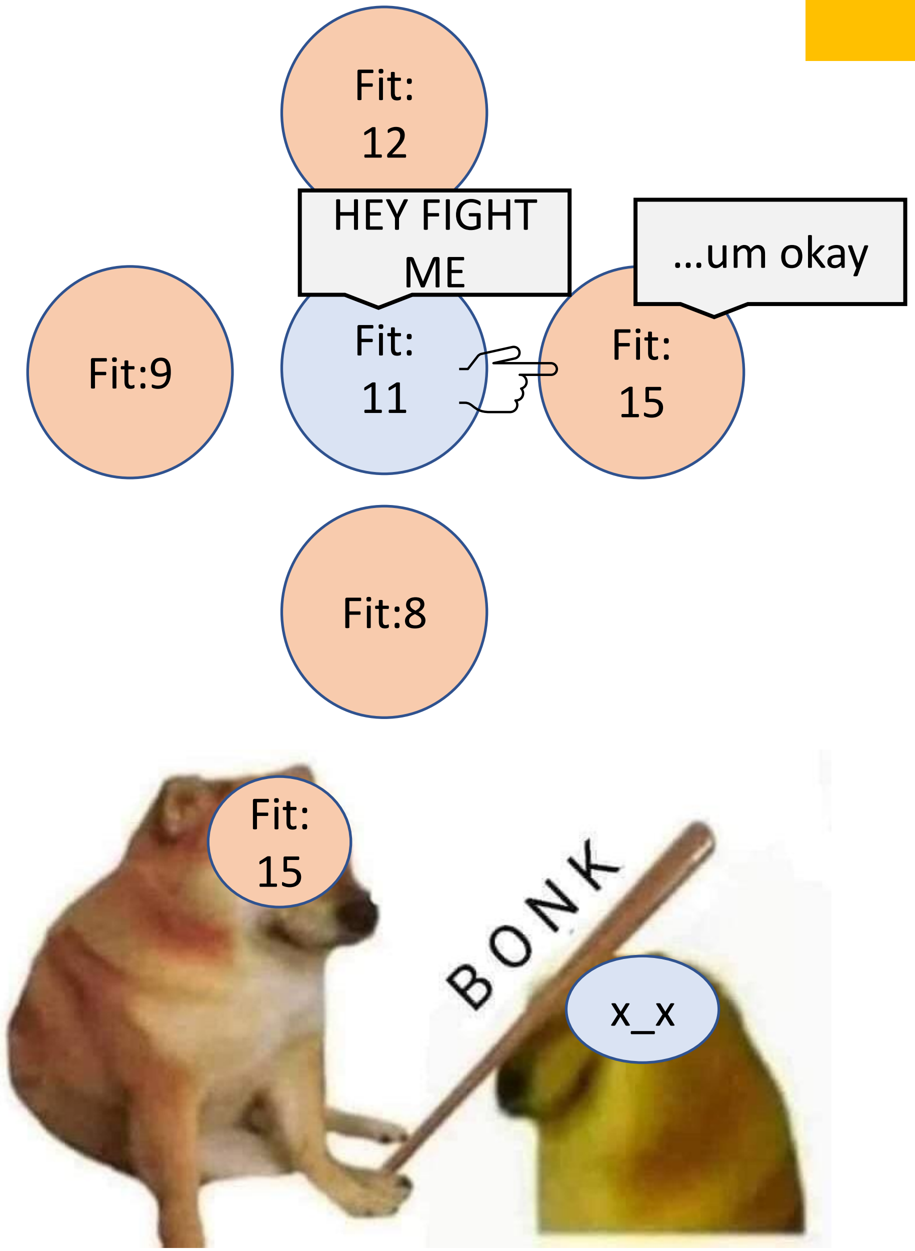
MAGA Framework

- Initialization: Creates grid of agents
- Neighbor Interaction Operators: Competition and Crossover
- Mutation: same as GA
- Self-learning operator: selected crossover and mutation
- Repeat until stopping condition



Neighborhood Interaction

- Each agent looks at its neighbors and finds the highest fitness agent
- If the neighbor has a higher fitness than the agent, the agent dies
- The agent is replaced by crossover using itself and the neighbor that killed it as parents
- If the neighbor has a lower fitness than the agent, do nothing



Self-Learning Operator

- Takes the best agents in each generation
- Each agent constructs its own MAGA performing same operations to find a new optimal agent
- Simulates an agent knowing the problem and trying to improve itself

Why use MAGA?

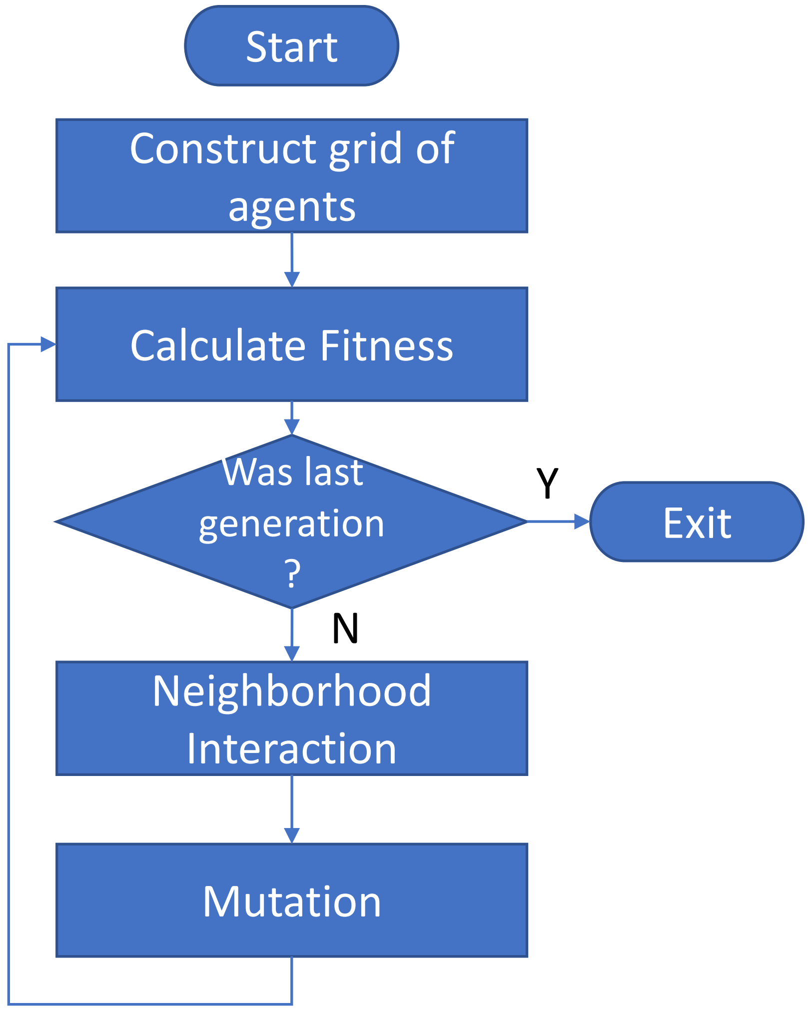
- Proven to overcome early and local convergence problems of traditional GAs
- Good scalability in terms of computational cost
- Co-evolution in an environment gives a better reflection of evolution in Nature
- Genetic algorithms proven to converge on a solution quicker than other methods like Ant-Colony Optimization for TSP

Goals

- Implement MAGA for the Vehicle Routing Problem
- Create a chromosome that allows algorithm to find optimum number of vehicles to divide route with
- Most other approaches have a fixed number of vehicles to run algorithm with
- Explore parameters affecting traversal of solution space (crossover type, mutation)

My Implementation: Flowchart

- Initialization: Creates grid of agents
- Neighbor Interaction Operators: Each agent finds neighbor with the highest fitness and performs crossover if it dies
- Crossover done one of two ways
- Mutation: each agent has a chance of mutating
- Self-learning operator: not implemented due to complexity
- Repeat for fixed number of generations

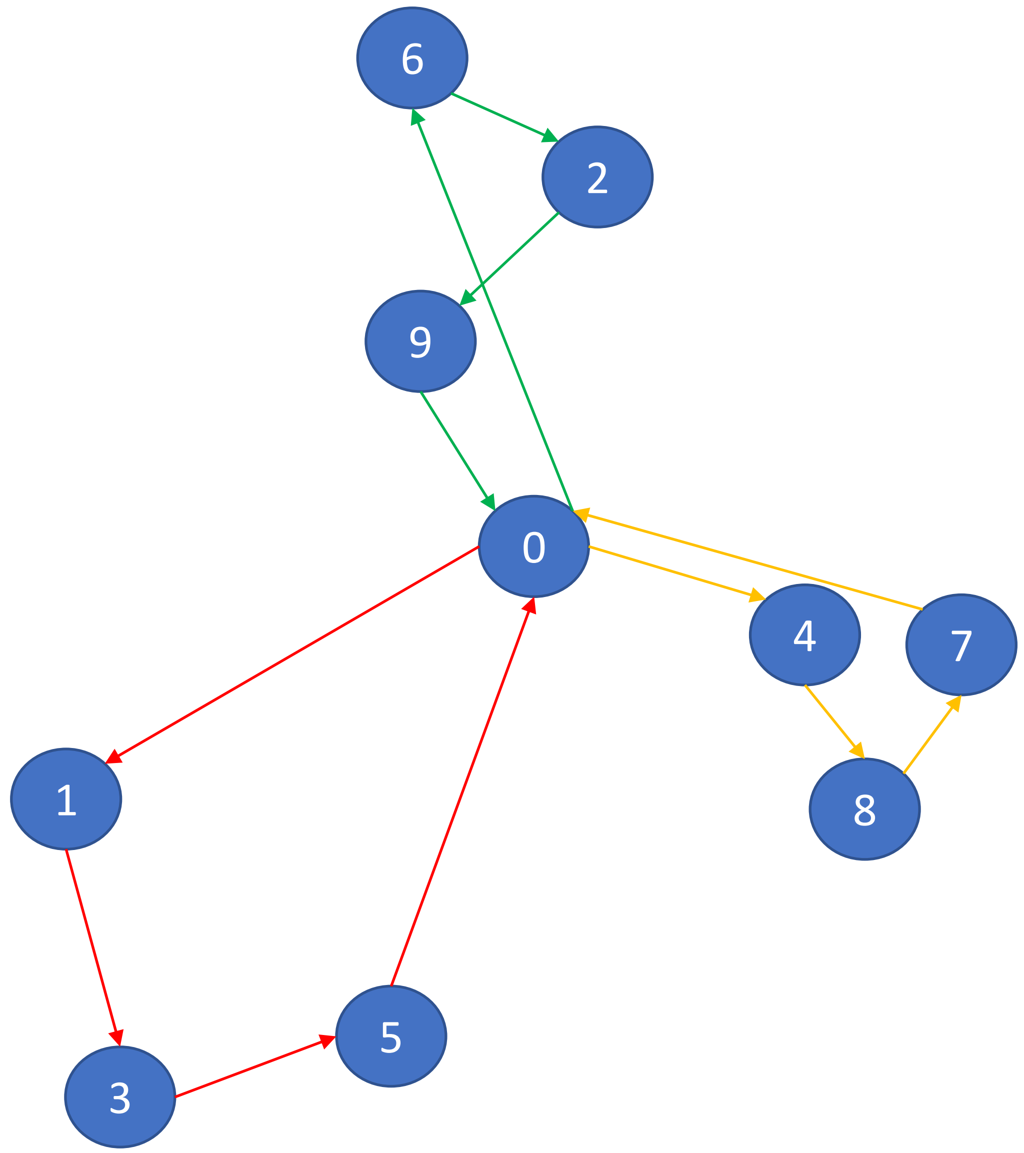


My Implementation: Chromosome

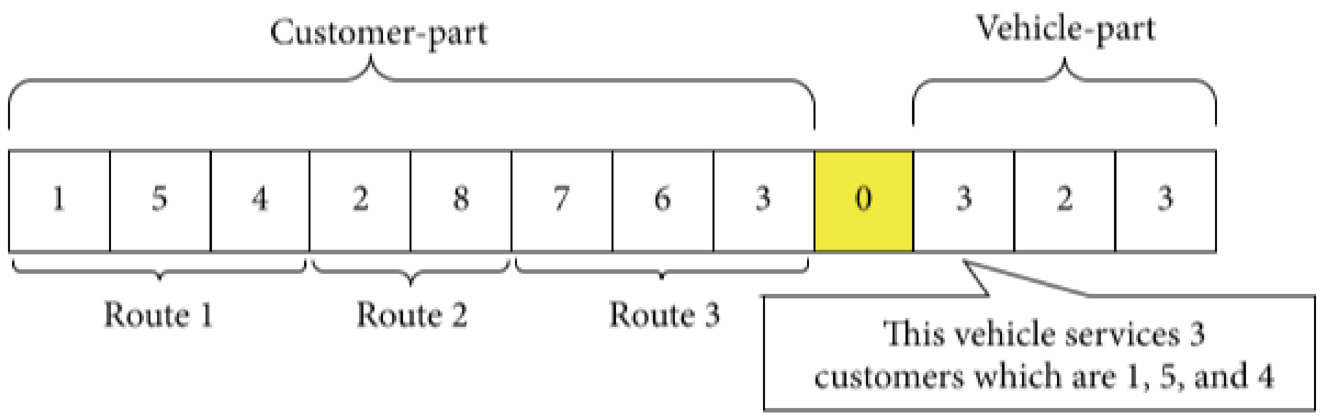
- Chromosome represented by a string of integers
- Each gene is an integer represented a stop to visit
- 0’s divide the route amongst another vehicle
- [1, 3, 5, 6, 2, 9, 4, 8, 7] represents a single vehicle going to stops in order from left to right
- [1, 3, 5, 6, 2, 0, 9, 4, 8, 7] represents two vehicles:
- first one goes to [1, 3, 5, 6, 2] second one goes to [9, 4, 8, 7]

My Implementation: Chromosome

- [1, 3, 5, 0, 6, 2, 9, 0, 4, 8, 7]
- All routes start and end at central depot (represented by 0)



Example of Other Chromosomes

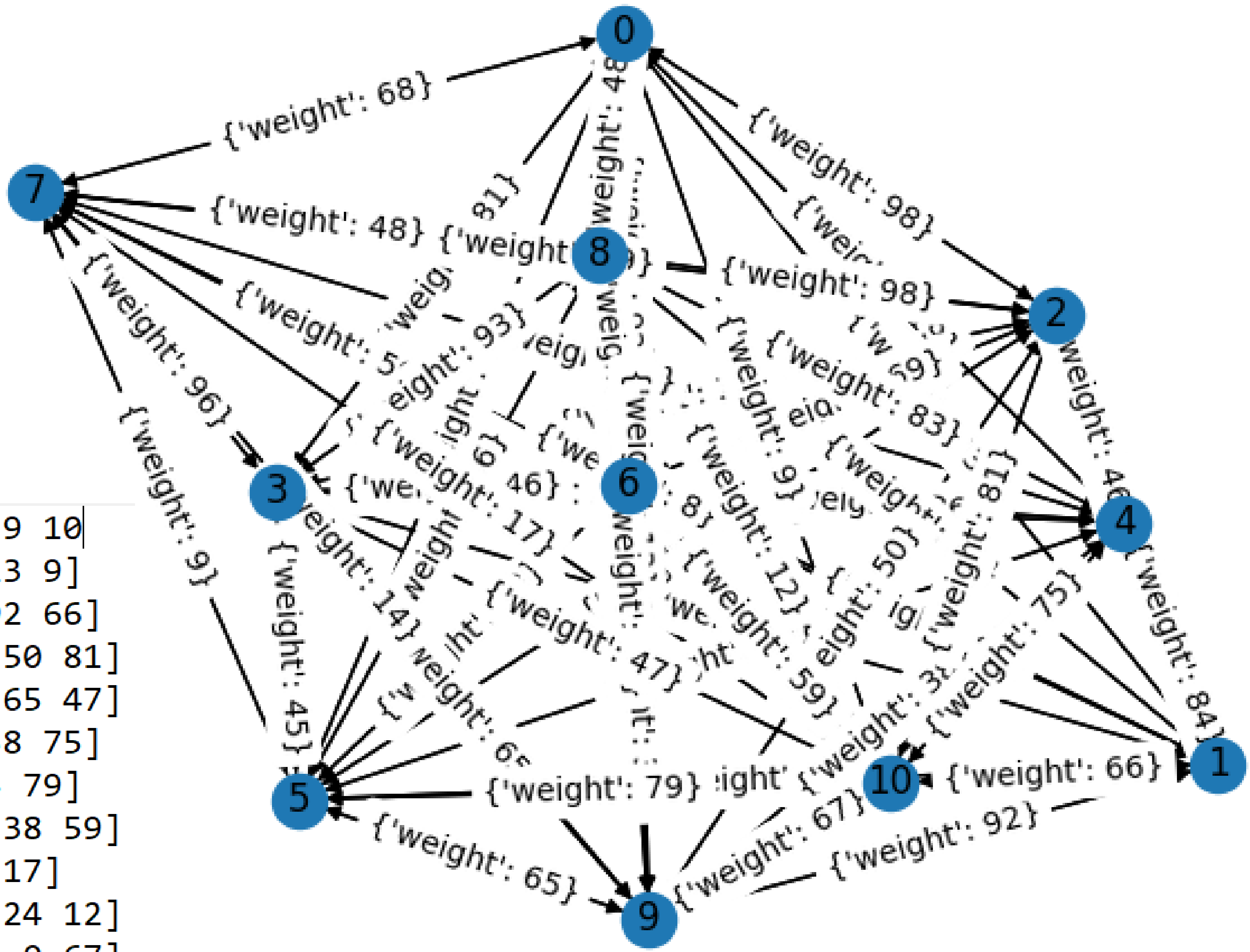


My Implementation: Fitness

- Fitness represents total cost of visiting all stops
- Highest fitness means lowest cost
- Calculated by sum of weights between stops traversed
- Weights: 1-100 can represent distance and time
- Weights stored in an Adjacency Matrix
- Symmetric matrix for simplicity

	0	1	2	3	4	5	6	7	8	9	10
0	0	13	98	81	19	6	92	68	48	13	9
1	13	0	66	46	84	41	69	8	23	92	66
2	98	66	0	14	46	48	69	29	98	50	81
3	81	46	14	0	67	45	46	96	93	65	47
4	19	84	46	67	0	30	36	6	83	38	75
5	6	41	48	45	30	0	73	9	36	65	79
6	92	69	69	46	36	73	0	59	92	38	59
7	68	8	29	96	6	9	59	0	48	14	17
8	48	23	98	93	83	36	92	48	0	24	12
9	13	92	50	65	38	65	38	14	24	0	67
10	9	66	81	47	75	79	59	17	12	67	0

- Chromosome = [1 3 2 4]
- Fitness = cost(0, 1) + cost(1, 3) + cost(3, 2) + cost(2, 4) + cost(4, 0)



	0	1	2	3	4	5	6	7	8	9	10
0	[0	13	98	81	19	6	92	68	48	13	9]
1	[13	0	66	46	84	41	69	8	23	92	66]
2	[98	66	0	14	46	48	69	29	98	50	81]
3	[81	46	14	0	67	45	46	96	93	65	47]
4	[19	84	46	67	0	30	36	6	83	38	75]
5	[6	41	48	45	30	0	73	9	36	65	79]
6	[92	69	69	46	36	73	0	59	92	38	59]
7	[68	8	29	96	6	9	59	0	48	14	17]
8	[48	23	98	93	83	36	92	48	0	24	12]
9	[13	92	50	65	38	65	38	14	24	0	67]
10	[9	66	81	47	75	79	59	17	12	67	0]

My Implementation: Neighborhood Interaction

- Each agent finds neighbor with highest-fitness
- If fitness of agent >= neighbor fitness: do nothing
- Else: Perform crossover with both agent’s chromosomes one of two ways at P(c) to replace agent
- Method 1: uses agent as the first parent in algorithm
- Method 2: uses neighbor as the first parent in algorithm
- In other MAGA research, order of parents mattered in crossover, thus the two methods

My Implementation: Crossover

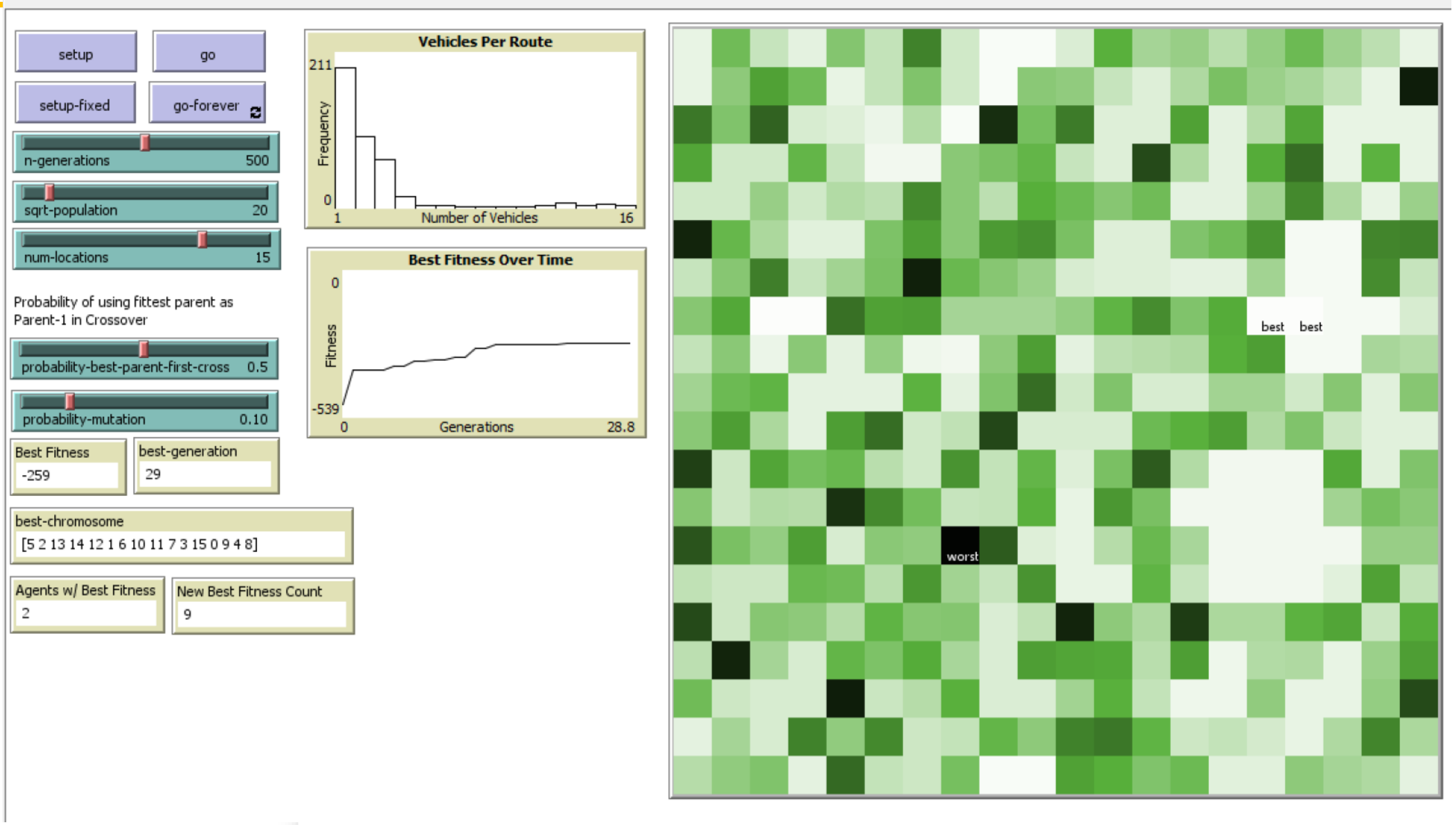
- CX2 crossover proposed by Hussain et al. in 2017
- Method compared against other best known crossover methods for TSP
- For my version:
  - Remove 0's from chromosomes remembering indexes
  - Perform CX2 creating two offspring
  - Place 0's into offspring at same index as higher fitness parent
  - Calculate fitness of new chromosomes and select best to replace agent

My Implementation: Mutation

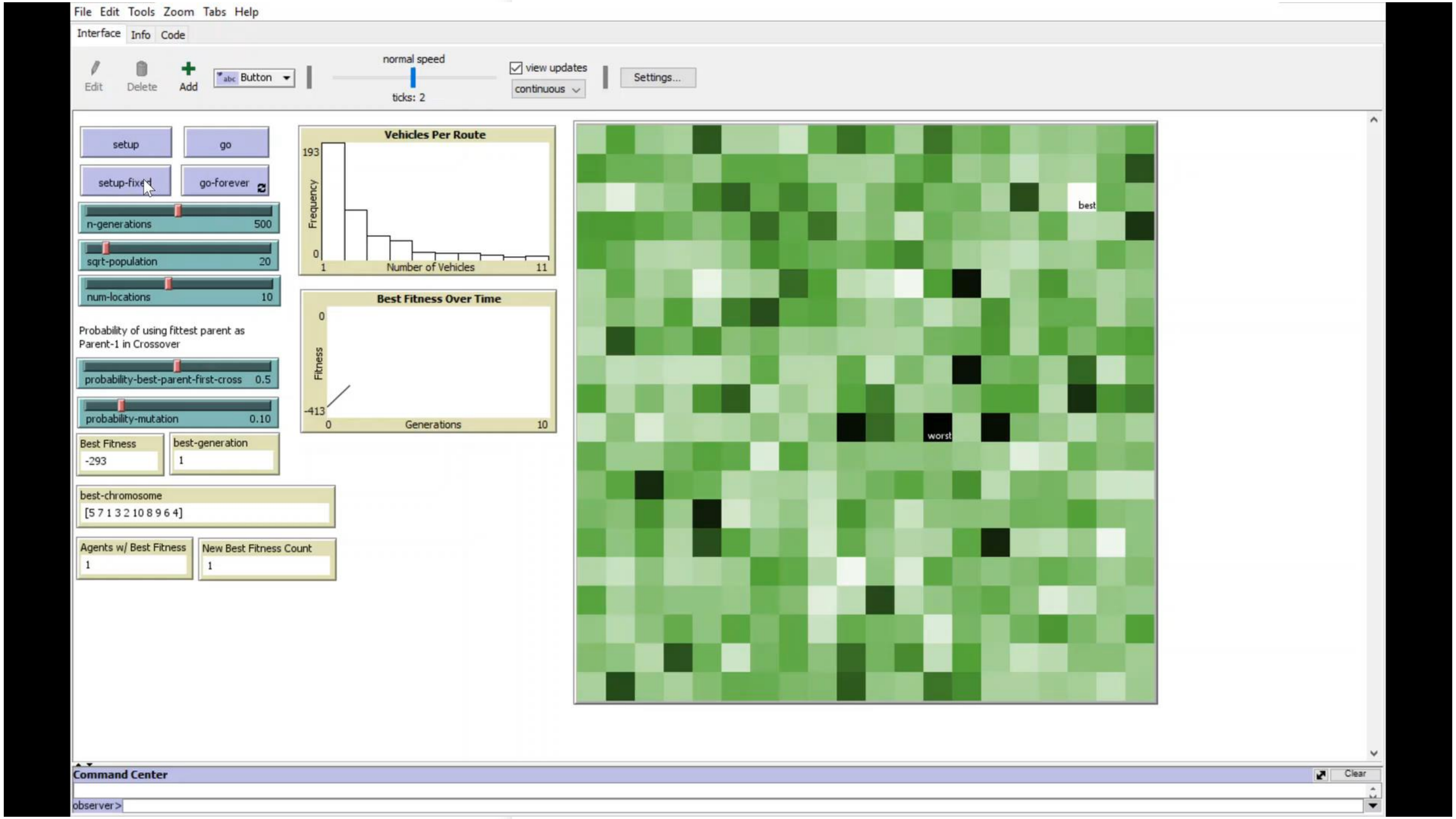
- Insert a random amount of 0’s into chromosome at random points
- [1, 3, 5, 6, 7, 2, 4] -> [1, 0, 3, 5, 6, 0, 7, 2, 4]
- Traditional GA swaps bits

My Implementation: NetLogo

- Programmed in NetLogo
- Easy domain to program agents
- Good visualizations
- Ability to run experiments varying parameters with BehaviorSpace



Demo

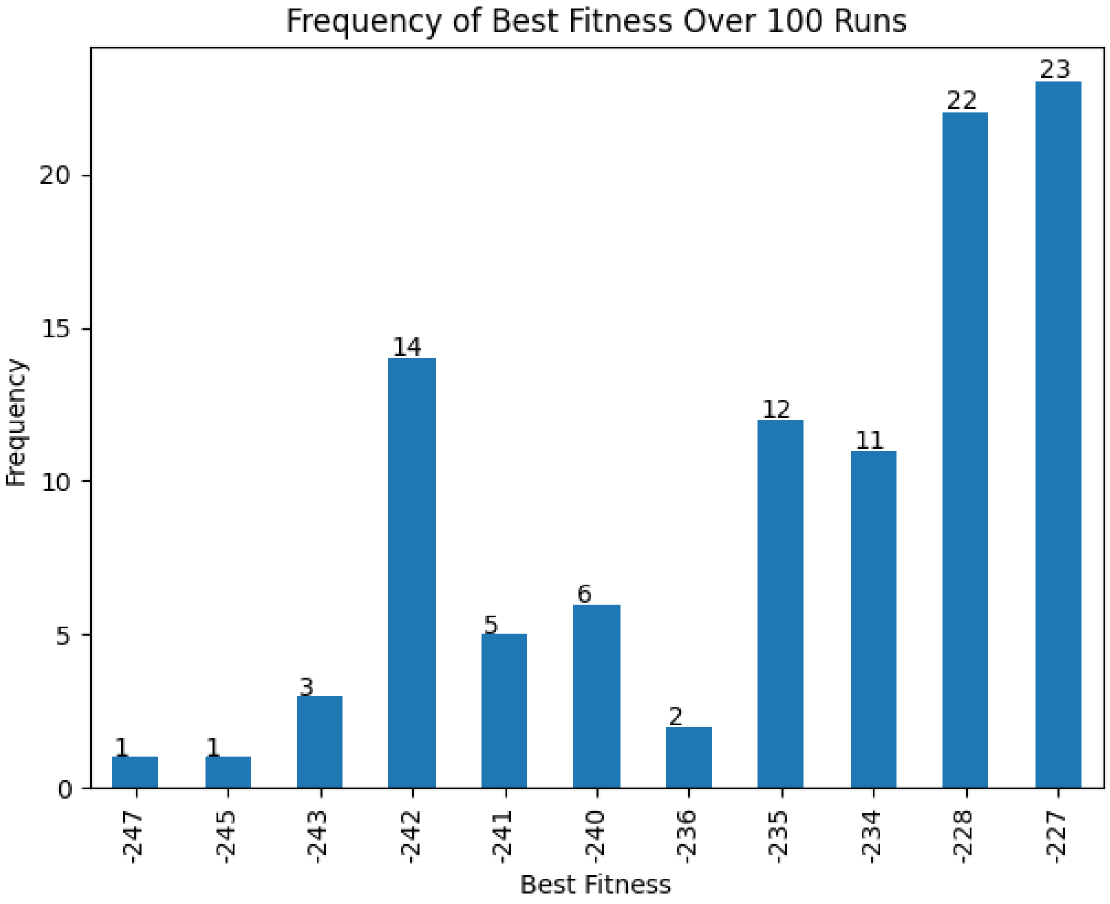

File Edit Tools Zoom Tabs Help
Interface Info Code
Edit Delete Add Button
normal speed
ticks: 2
view updates
continuous
Settings...
setup
go
setup-fixed
go-forever
n-generations 500
sqrt-population 20
num-locations 10
Probability of using fittest parent as Parent-1 in Crossover
probability-best-parent-first-cross 0.5
probability-mutation 0.10
Best Fitness
-293
best-generation
1
best-chromosome
[5 7 1 3 2 10 8 9 6 4]
Agents w/ Best Fitness
1
New Best Fitness Count
1
Vehicles Per Route
193
Frequency
0
1
Number of Vehicles
11
Best Fitness Over Time
0
Fitness
-413
0
Generations
10
best
worst
Command Center
Clear
observer>

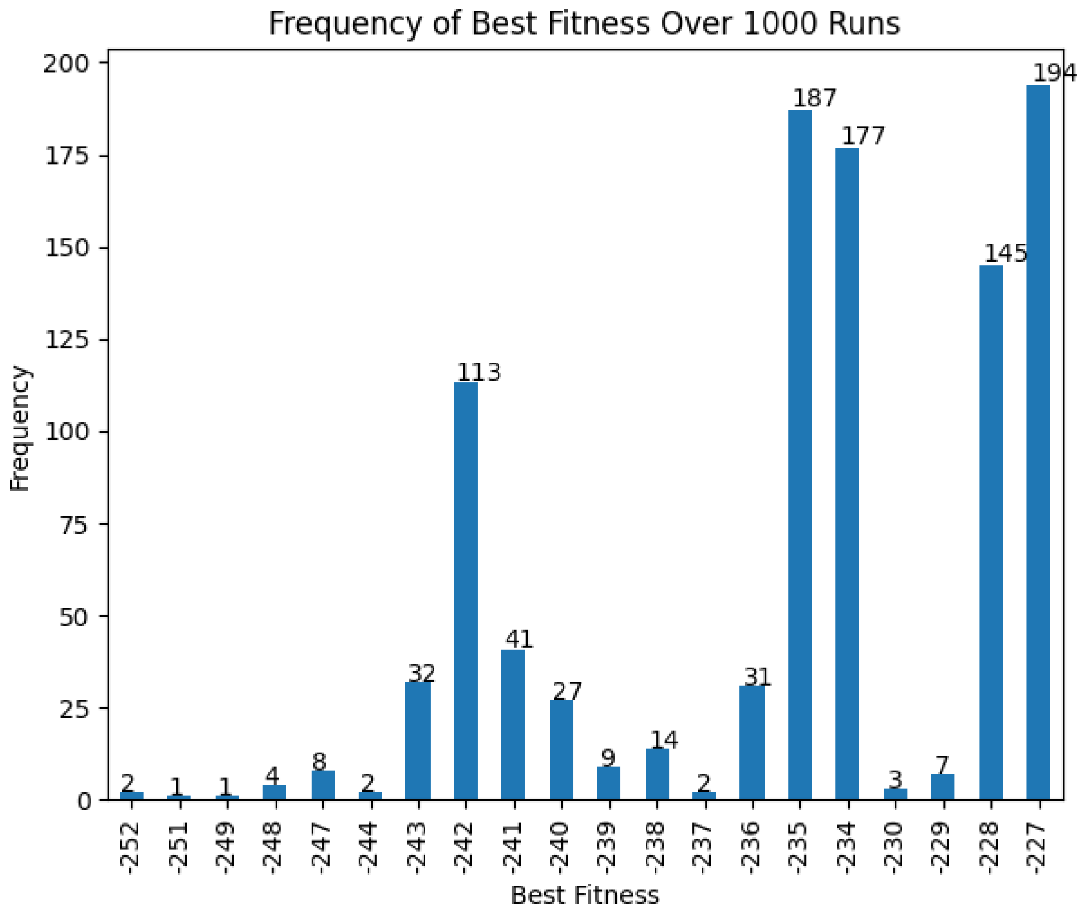
Results

- Data Collected using same adjacency matrix from earlier
- Best Fitness = -227
- Possible Chromosomes:
  - [5, 0, 10, 8, 1, 7, 4, 2, 3, 6, 9]
  - [9, 6, 3, 2, 4, 7, 1, 8, 10, 0, 5] reversed
  - [5, 0, 9, 6, 3, 2, 4, 7, 1, 8, 10] 5 still on separate route
  - [10, 8, 1, 7, 4, 2, 3, 6, 9, 0, 5] reversed

	0	1	2	3	4	5	6	7	8	9	10
0	[0	13	98	81	19	6	92	68	48	13	9]
1	[13	0	66	46	84	41	69	8	23	92	66]
2	[98	66	0	14	46	48	69	29	98	50	81]
3	[81	46	14	0	67	45	46	96	93	65	47]
4	[19	84	46	67	0	30	36	6	83	38	75]
5	[6	41	48	45	30	0	73	9	36	65	79]
6	[92	69	69	46	36	73	0	59	92	38	59]
7	[68	8	29	96	6	9	59	0	48	14	17]
8	[48	23	98	93	83	36	92	48	0	24	12]
9	[13	92	50	65	38	65	38	14	24	0	67]
10	[9	66	81	47	75	79	59	17	12	67	0]

Results: 10 stops, 500 generations, P(best-parent-first)=0.5, P(mutation)=0.1

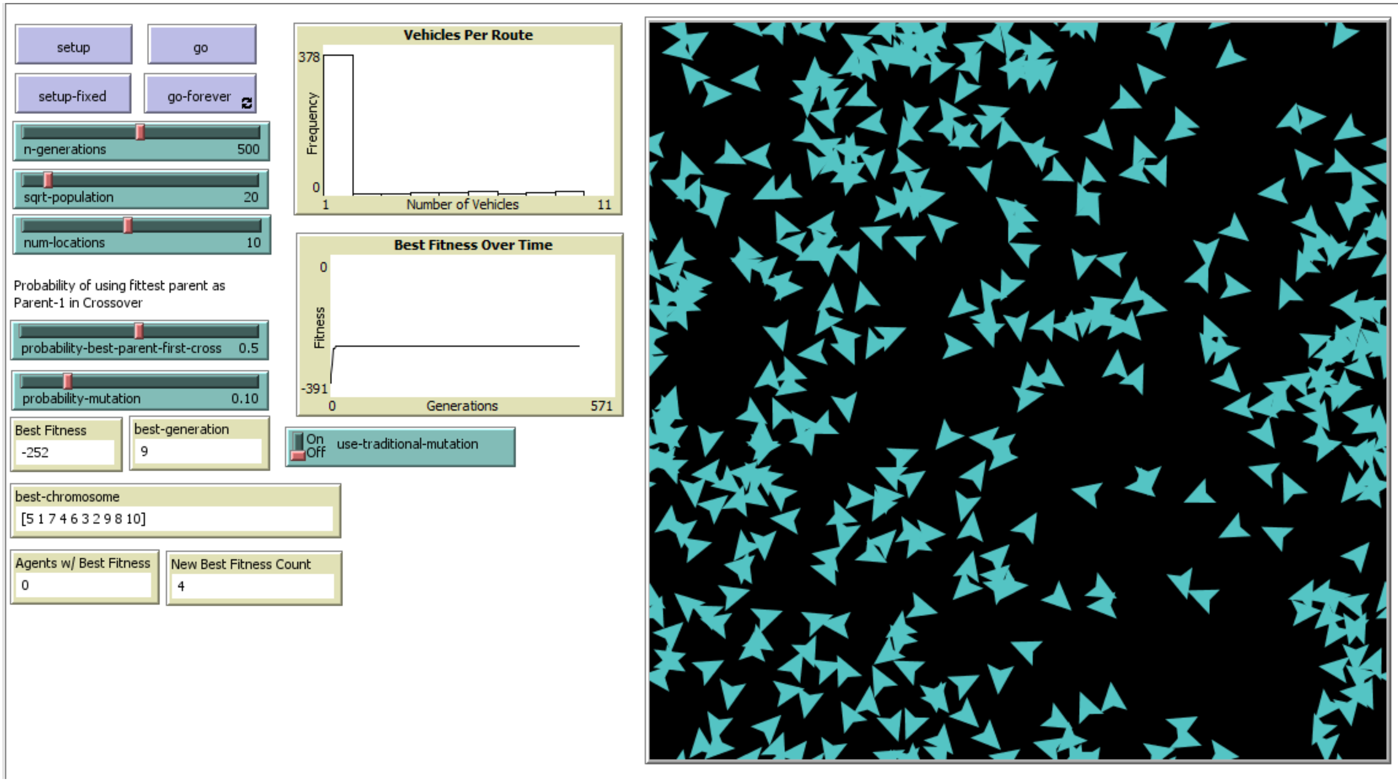




Tends towards best solution, worst-fitness=-894

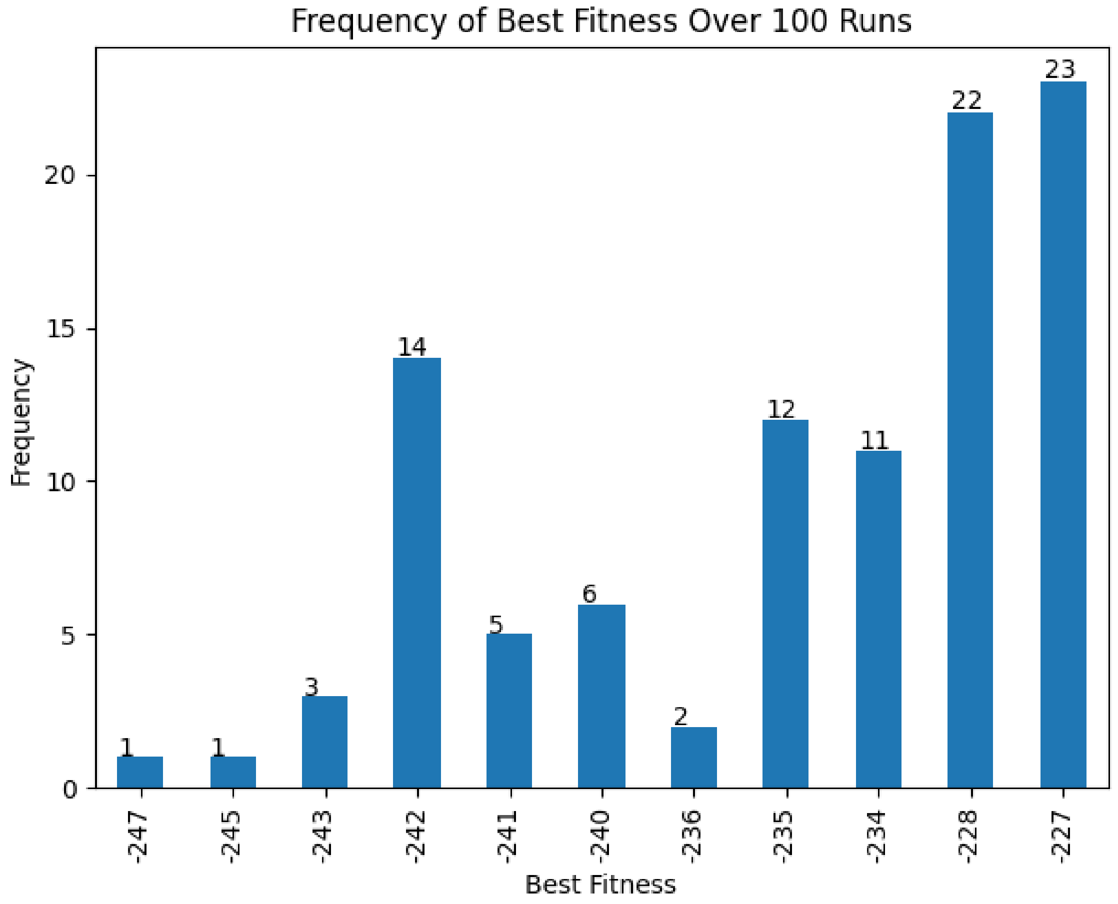
Traditional GA with Tournament Selection

- Removed lattice environment
- Selection from entire population
- Tournament selection size 15
- Same crossover and mutation

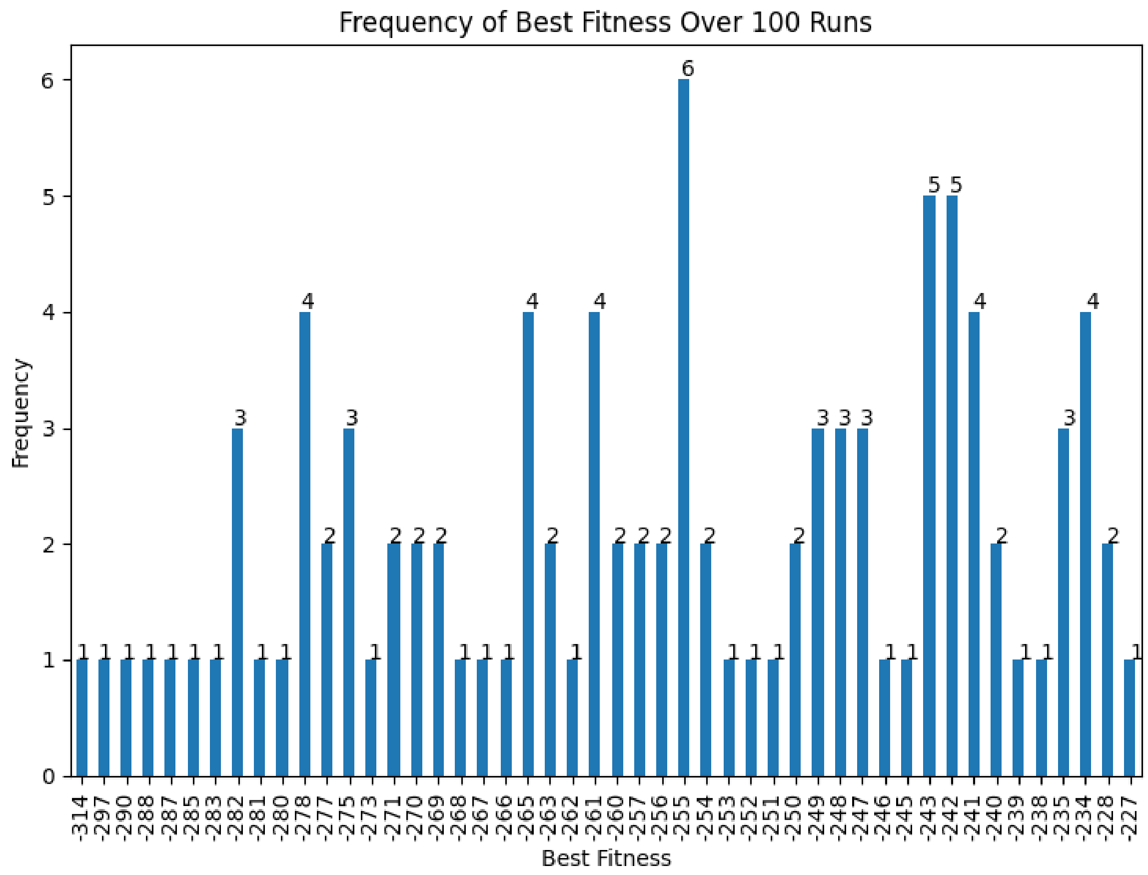


Comparison with same parameters

MAGA

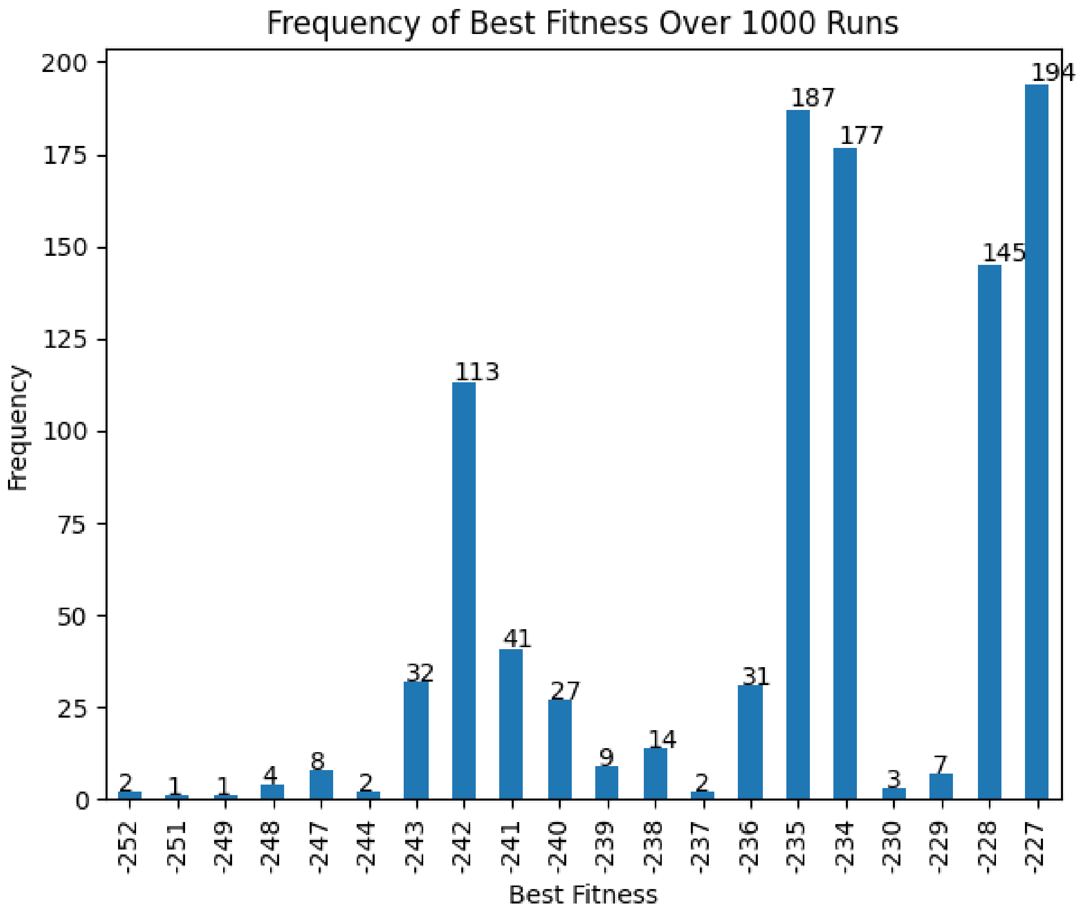


GA – Tournament Selection

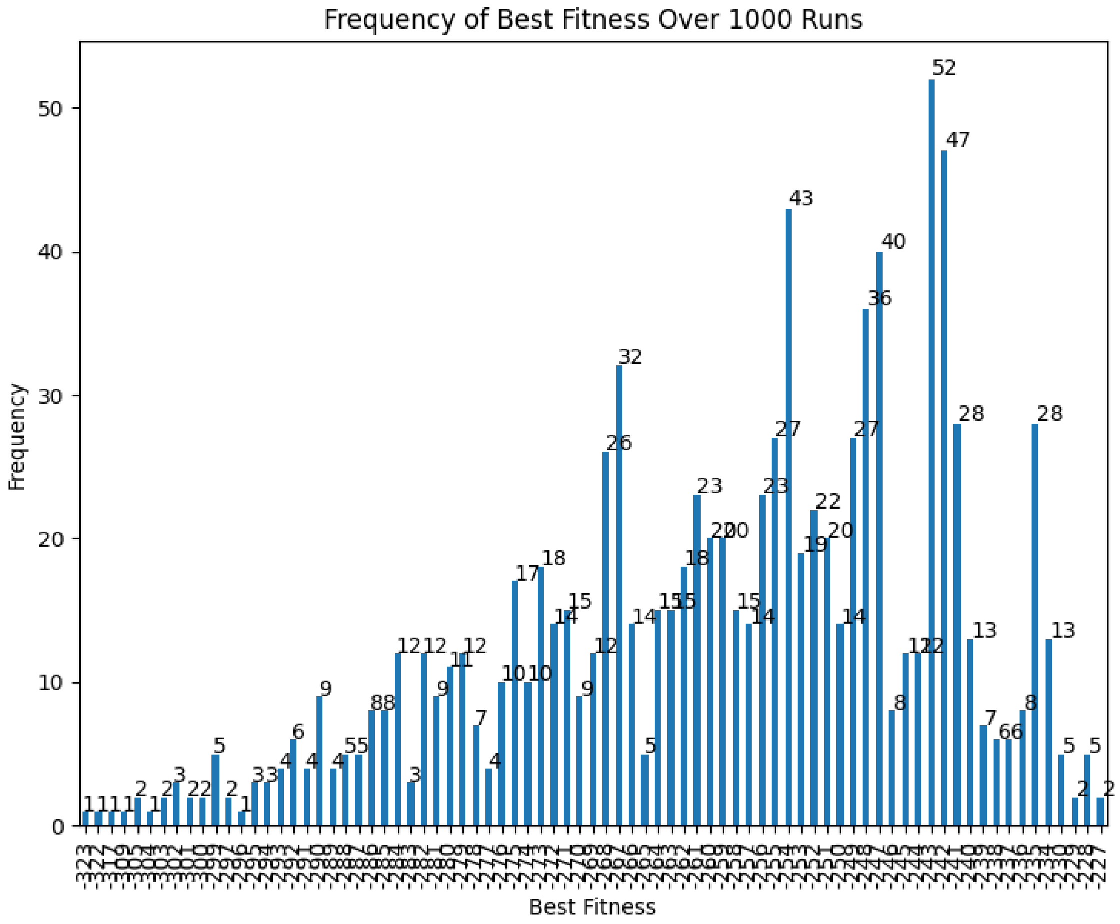


Comparison with same parameters

MAGA

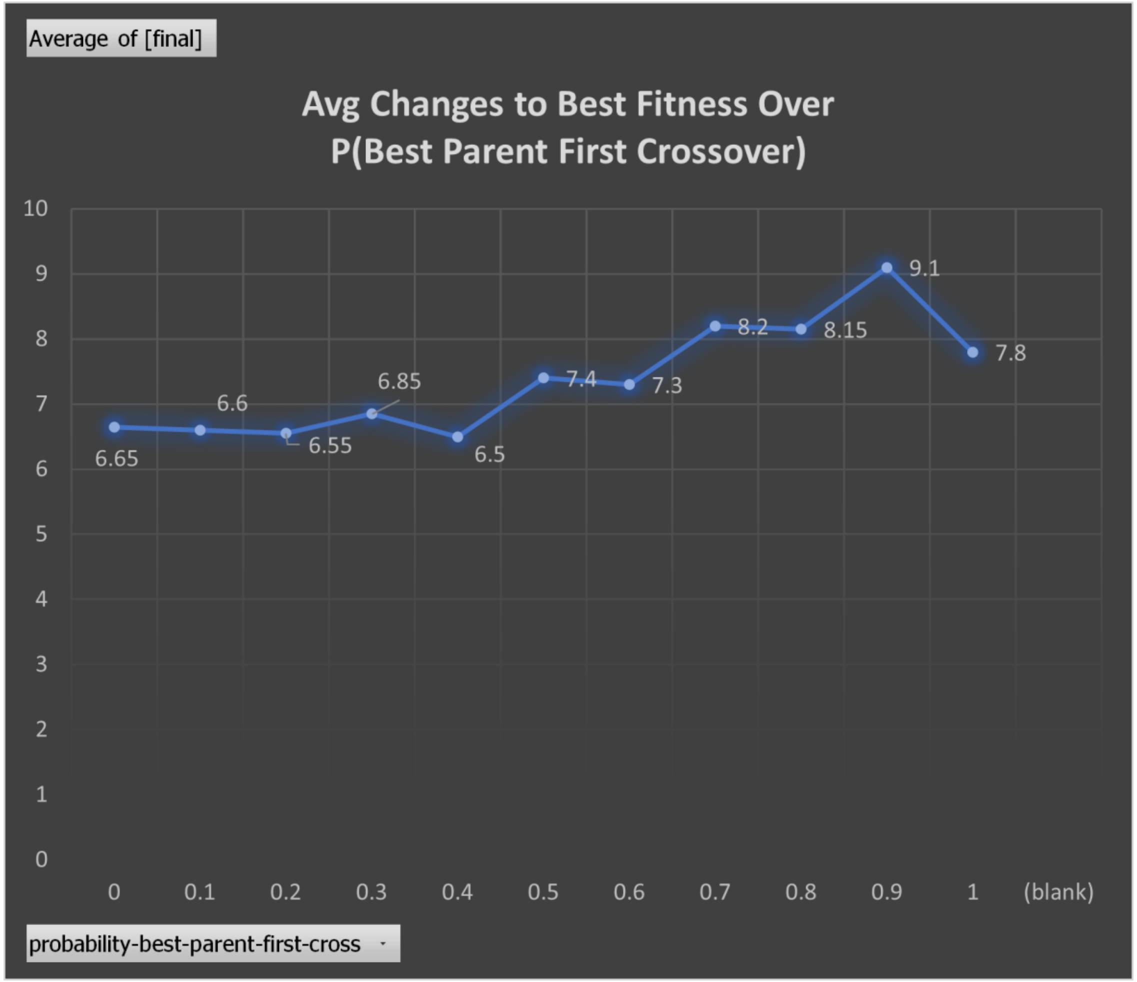


GA – Tournament Selection

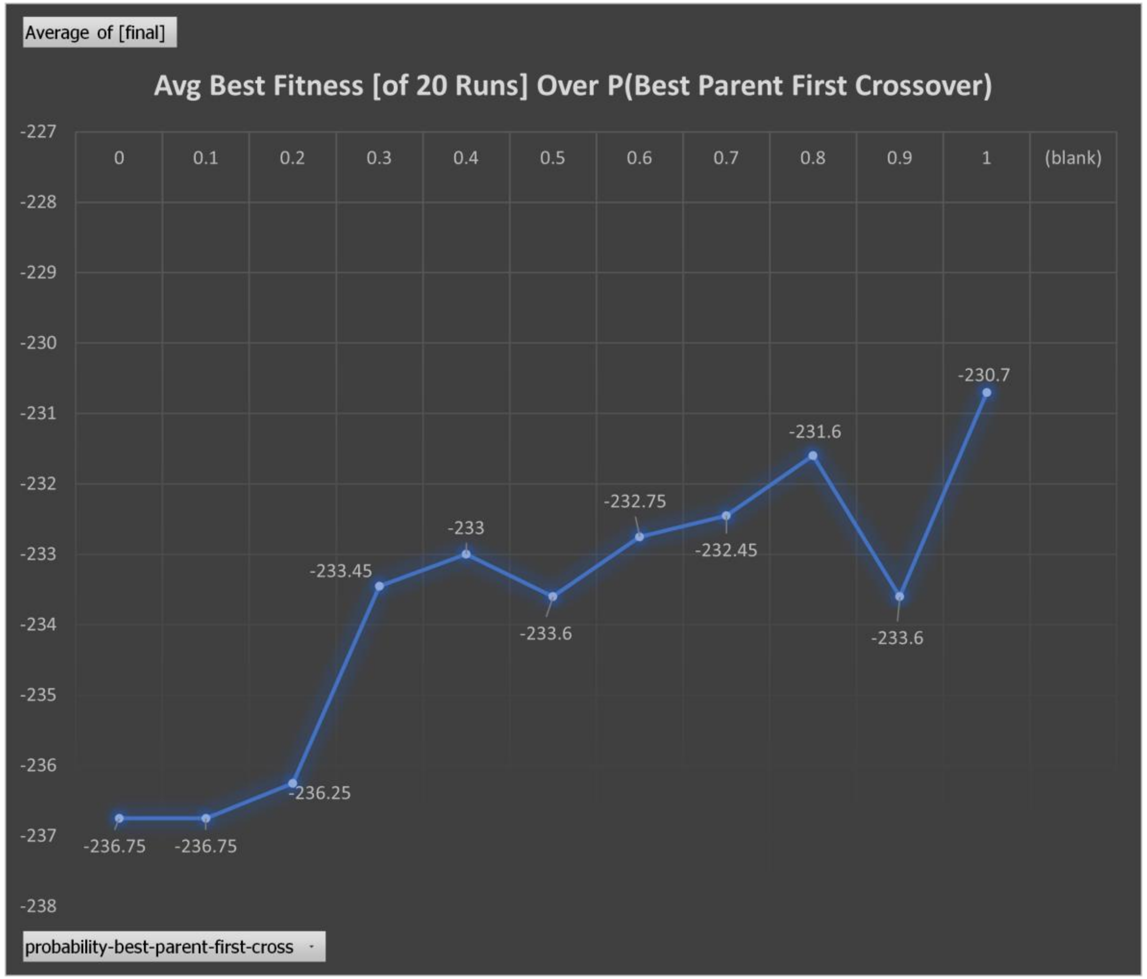


<20 minutes vs 1 hour+

Results: 10 stops, 500 generations, P(mutation)=0.1

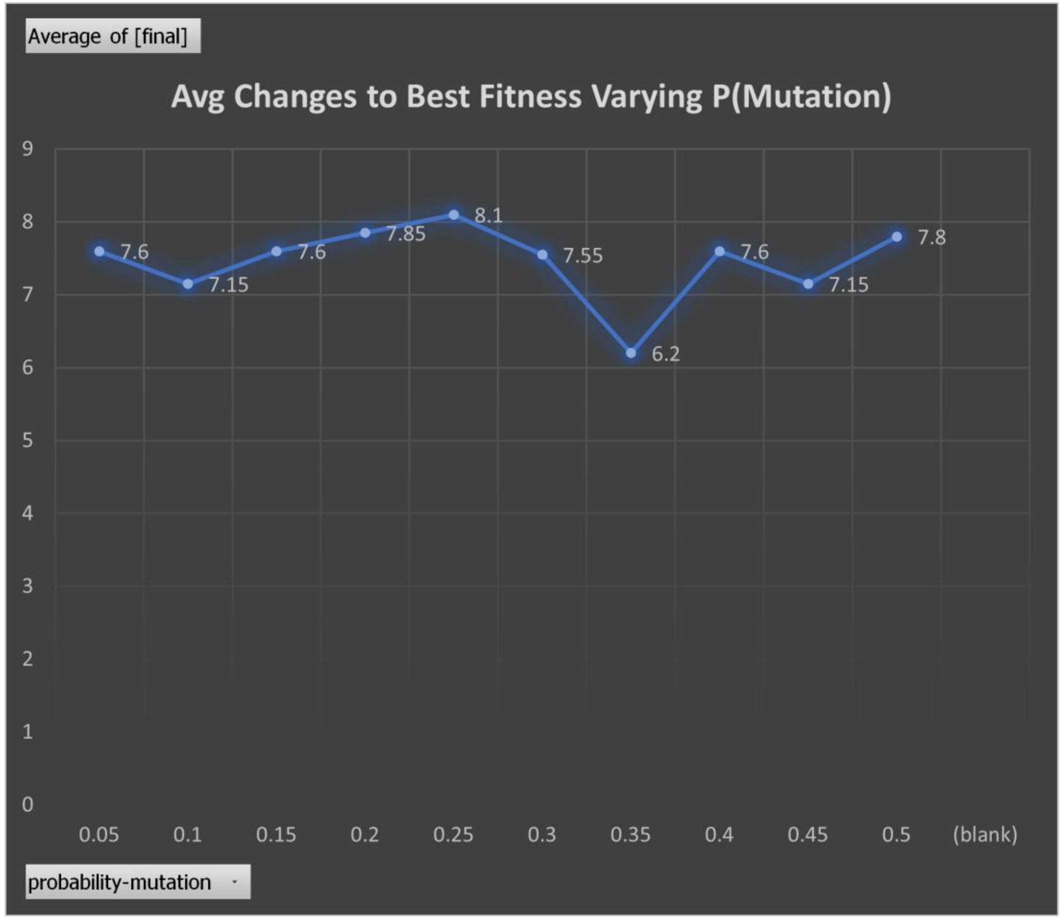


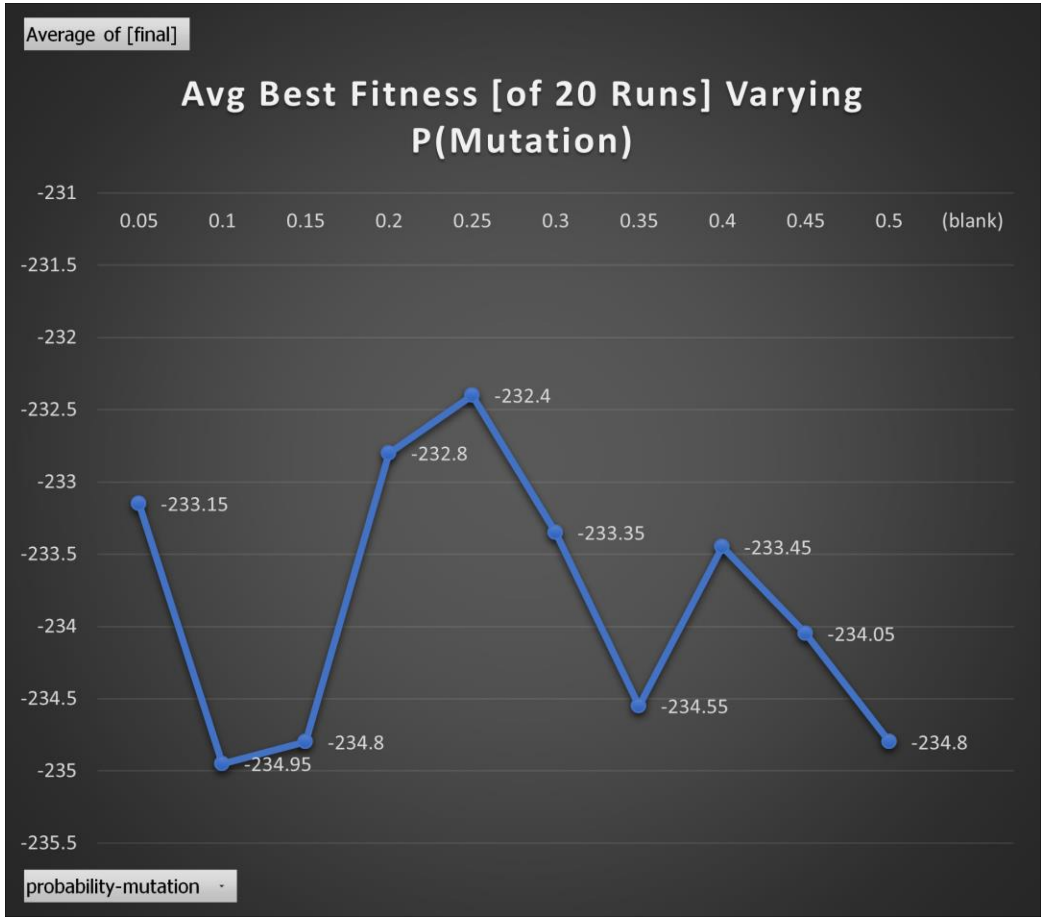
Exploration not very affected



Exploitation seems to increase when fitter parent is used as Parent #1 in CX2

Results: 10 stops, 500 generations, P(best-parent-first)=0.5

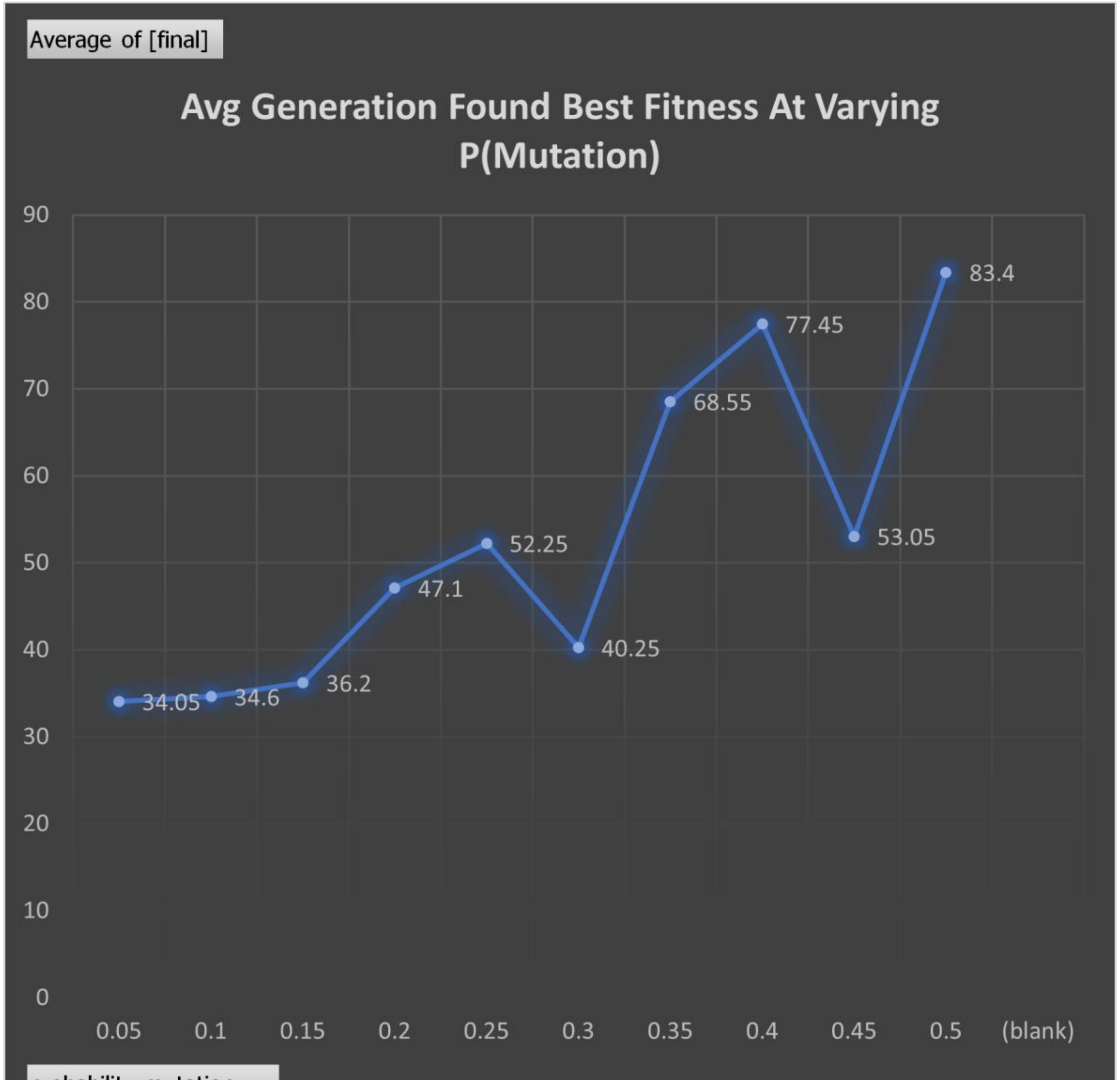




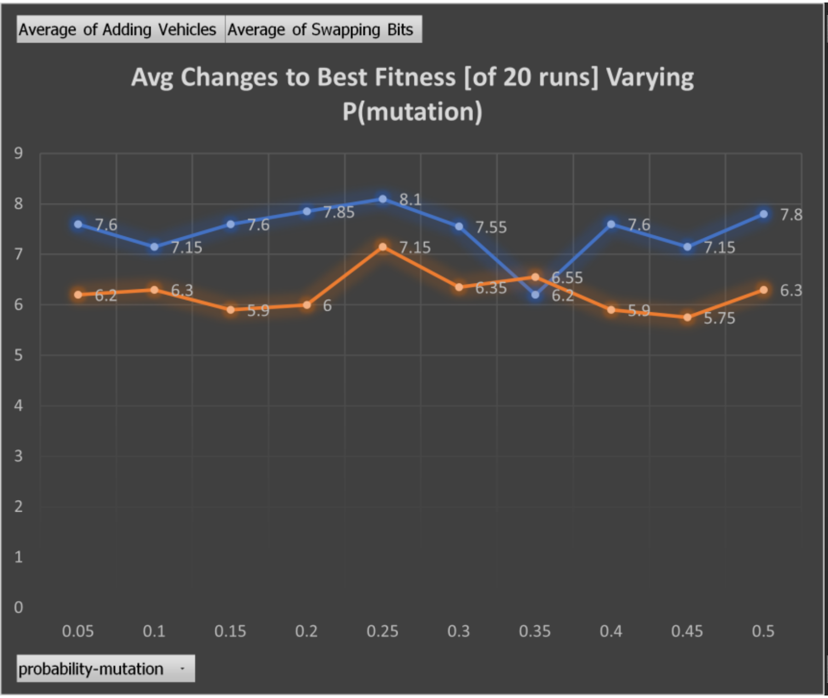
For given constraints, mutation seems to have little affect on finding good and diverse solutions

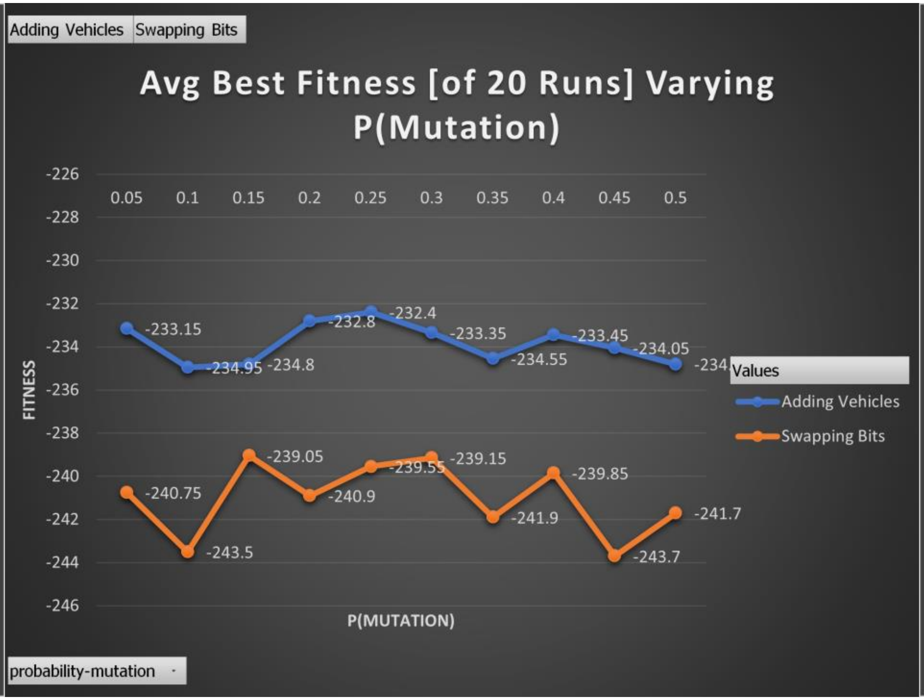
Results: 500 generations, P(best-parent-first)=0.5

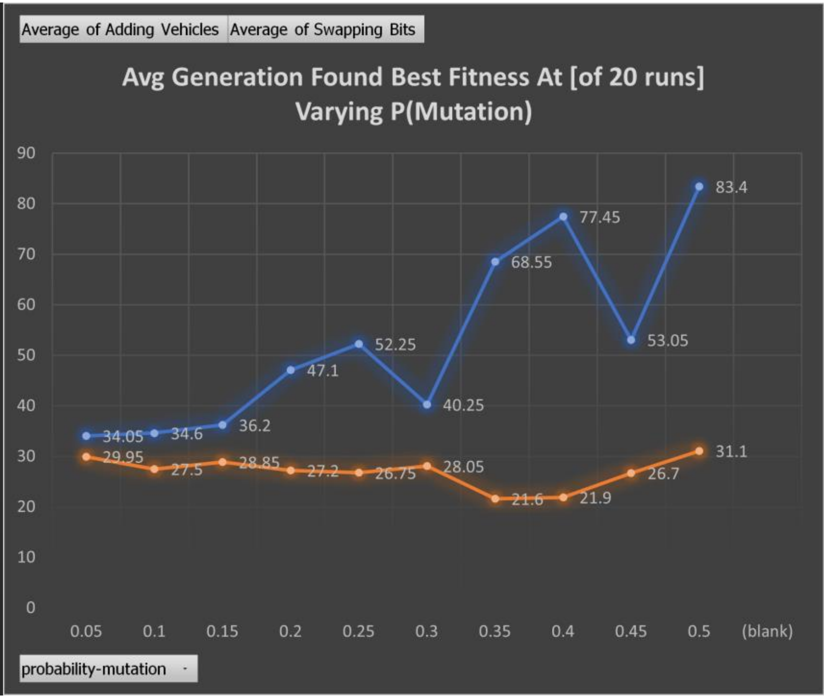
- As P(mutation) increases, the best fitness is found in later generations
- Maybe exploration is affected?
- Mutation helps system avoid converging on a solution too early



Comparing Traditional Mutation







Traditional Mutation of swapping bits causes system to converge on a solution much faster on a solution that is on average not the best

Conclusions

- Working MAGA program for VRP
- More effective than GA
- Successful in seeing ideal number of vehicles emerge
- CX2 algorithm produces higher fitness when higher fitness parent is used as Parent 1
- Mutation algorithm prevents early convergence

Future Work

- Collect and analyze data with larger number of stops
    - Solution space may be limited with only 10 stops
    - Potential to see wider range in number of changes to best fitness
- Add self-learning operator for full MAGA implementation
- Analyze diffusion of information across grid
- Compare run-time to exhaustive search
    - Tried to program exhaustive search in Python and did not have enough RAM ☹

References

- Alhanjouri, Mohammed & Alfarra, Belal. (2011). Ant Colony versus Genetic Algorithm based on Travelling Salesman Problem. International Journal of Computer Technology and Applications. 2. 570-578.
- Drezewski, Rafal & Woźniak, Piotr & Siwik, Leszek. (2009). Agent-Based Evolutionary System for Traveling Salesman Problem. 5572. 34-41. 10.1007/978-3-642-02319-4_5.
- Peng, Chong & Wu, Guanglin & Liao, T. & Wang, Hedong. (2019). Research on multi-agent genetic algorithm based on tabu search for the job shop scheduling problem. PLOS ONE. 14. e0223182. 10.1371/journal.pone.0223182.
- Mitchell, M., Tisue, S. and Wilensky, U. (2012). NetLogo Robby the Robot model. http://ccl.northwestern.edu/netlogo/models/RobbytheRobot. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.
- Wei Zhou, Tingxin Song, Fei He, Xi Liu, "Multiobjective Vehicle Routing Problem with Route Balance Based on Genetic Algorithm", Discrete Dynamics in Nature and Society, vol. 2013, Article ID 325686, 9 pages, 2013. https://doi.org/10.1155/2013/325686
- Zhong, Weicai & Liu, Jing & Xue, Mingzhi & Jiao, Licheng. (2004). A Multiagent Genetic Algorithm for Global Numerical Optimization. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society. 34. 1128-41. 10.1109/TSMCB.2003.821456.