**Comparing the Performance of Genetic algorithm with Reinforcement learning for simple game learning problem**

**Problem definition:**

In this project we compare the performance of Evolutionary computing algorithm like genetic algorithm with reinforcement learning like Q-learning for a simple game learning problem, the game learning problem setup is given a graph and a start and end location the goal of the learning algorithm/agent is to find the best possible path from the source to destination. The testing has been done on various graph setups of different sizes, sparseness and even by blocking few nodes/path in the graph. Only a specific version of genetic algorithm and reinforcement learning algorithm is applied to this specific problem and all conclusions and results derived are restricted within this scope and might or might not scale well for other problems.

**Introduction:**

Reinforcement learning is an area of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) concerned with how [software agents](https://en.wikipedia.org/wiki/Software_agent) ought to take [actions](https://en.wikipedia.org/wiki/Action_selection) in an environment in order to maximize some notion of cumulative reward. It basically attempts to maximise the expected sum of rewards as per a pre-defined reward structure obtained by the agent. It does so by learning a value function which is updated using information obtained from the agent’s interactions with its environment. The interactions are dictated by a balance of actions for exploration and exploitation. Exploitation ensures that the agent makes use of (exploits) the already acquired knowledge (value function), while exploration makes sure the agent remains ‘playful’, which might allow it to improve its knowledge.

On the other hand Evolutionary Computing algorithms like Genetic algorithm starts with a population of randomly generated individuals/solutions, and uses the principle of natural selection to discover useful sets of solutions. The selection is usually fitness, with fitter individuals being allowed a higher probability of being selected into the subsequent generation. In order to search the solution-space, a proportion of individuals are subjected to mutation and crossover operations. The hope is that as the generation progress, fitter and fitter individuals get selected into the subsequent generations, which will ultimately deliver optimal or close to optimal solutions.

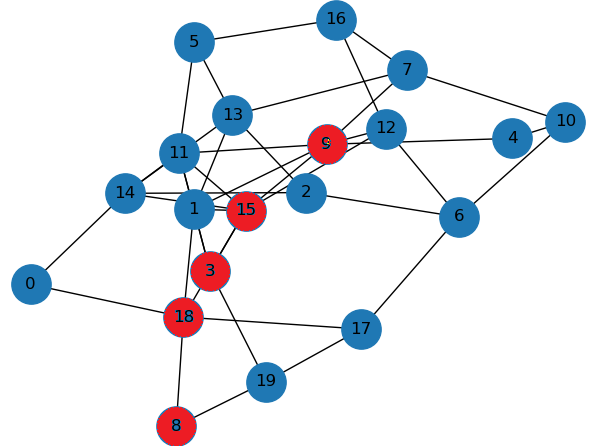
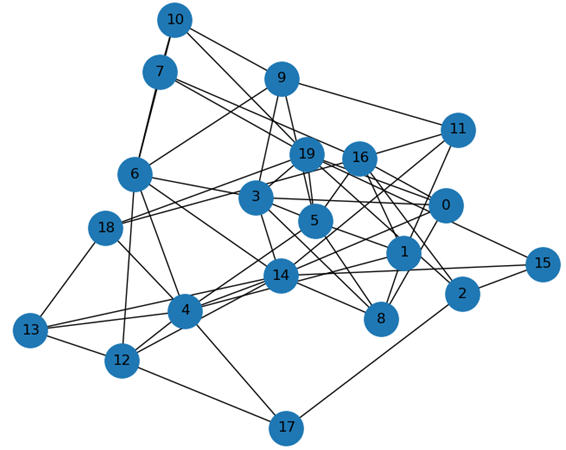
These are the fundamental difference between these two learning algorithms:

* The fundamental operating of the two approaches for both the algorithms is very different. Reinforcement learning uses a relatively well understood and mathematically grounded framework of Markov decision processes, whereas most evolutionary computing algorithms like genetic algorithms etc are largely based on heuristics.
* The value function update in reinforcement learning is a gradient-based update, whereas genetic algorithm generally doesn’t use such gradients.
* Reinforcement learning uses the concept of one agent, and the agent learns by interacting with the environment in different ways. In evolutionary algorithms, they usually start with many "agents" and only the "strong ones survive"
* Reinforcement learning benefits from features such as temporal credit assignment, which can speed up the convergence and other mechanisms such as experience replay, wherein recent agent-environment interaction information is stored and recalled from time to time in order to accelerate learning whereas no such strategies exists in Genetic algorithms.
* Reinforcement learning agent learns both positive and negative actions, but evolutionary algorithms only learns the optimal, and the negative or suboptimal solution information are discarded and lost.

Despite these differences, one may argue that there are certain similarities between the two approaches like they are both nature-inspired, and they both attempt to find goodness of solutions as per some pre-defined notion of ‘goodness’ i.e. Reinforcement Learning and Evolutionary Computing  are two different optimisation techniques that address the same problem: the optimisation of a function, the maximisation of an agent's reward in reinforcement Learning  and the fitness function in evolutionary computing  , respectively, in potentially unknown environments. In terms of utility, recent studies ([Evolution Strategies as a Scalable Alternative to Reinforcement Learning](https://blog.openai.com/evolution-strategies/)) have suggested that GA approaches could serve as a reasonable alternative to RL ones (at least for some problems). However, current research directions indicate that these two algorithms are related in a more subtle, but important sense - in the sense that it is a combination of some versions of these algorithms (along with other supervised and unsupervised algorithms) which could potentially lead to an artificial general intelligence (AGI).

This project aims to compare the performance of these two different strategies on a specific game learning task. The learning agent basically tries to learn and find solutions to the problem.

**Problem Setup:**

Consider the game setup similar to a simple maze problem. Basically maze is a path or collection of paths, typically from an entrance to a goal. The word is used to refer both to branching [tour puzzles](https://en.wikipedia.org/wiki/Tour_puzzle) through which the solver must find a route, and to simpler non-branching patterns that lead unambiguously through a convoluted layout to a goal. The input for this problem is defined as an adjacency matrix which represents connectivity from one location to another. Given a start location, the goal of the algorithm is to find the best path (shortest path in terms of nodes) from the start to the given end location. We also block certain intermediate nodes in the graph which indicate a blockage similar to the maze problem. 

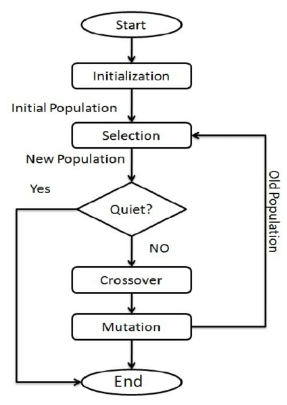
**Figure 1** - **Unblocked Figure 2**- **Blocked**

Consider the graph the above, the task of the learning algorithm in each of the case is to find shortest path from the start node 0 to the end node 19. The first figure is an unblocked case whereas in second case the nodes indicated by red colour is blocked which means the algorithm cannot pass through these marked nodes to reach the destination.

**Algorithms:**

Two different algorithms one belonging to family of evolutionary strategies and one belonging to reinforcement learning have been used. In this research, we use specific version of genetic algorithm, a type of evolutionary computing algorithm and q-learning algorithm which is a type of reinforcement learning to solve the given game learning problem.

**Genetic algorithm:**

The algorithm basically starts with a population of randomly generated individuals/solutions, and uses the principle of natural selection to discover useful sets of solutions. The selection is usually fitness, with fitter individuals being allowed a higher probability of being selected into the subsequent generation. In order to search the solution-space, a proportion of individuals are subjected to mutation and crossover operations.

Input setup: For the given problem we set up the genetic algorithm as a minimization problem, in the given adjacency matrix, a path from one node to other is indicated by 1 or 0 otherwise. We block the nodes by setting up higher weights for each of the edges in the node.

For the corresponding problem setup we elaborate how each of these steps was implemented:

Initial Population: For the initial set of population, given a start node and end node, k population of solutions are generated. The populations are generated by using a random walk from start node until it reaches the end node. Consider the figure 1, an example of random solution from 0th node to 19th node would be [0,8,14,6,3,19]

Fitness Function: The fitness function in this case calculate the distance of the route which is then used to minimize the distance from each node to each other node. In this particular case we assign constant distance as 1 for connections. This should also work if we give real valued weights or distances in numbers. For example the fitness value for the route [0,8,14,6,3,19] in figure 1 would be 5.

Cross over operator: In this process we take two solutions and mix them to find another solution. There are different cross-over operators available in literature like single point, multi-point, order based etc. For the problem we choose to use single point cross-over technique.

For example consider two different route solutions:

**Solution1: [0,7,2,18,3,7,20]**

**Solution2: [0,6,2,18,3,2,18,3,7,20]**

Let the cross-over point be 3.(typically we choose it randomly)

Resultant new solution:

**Solution1: [0,7,2,18,3,2,18,3,7,20]**

**Solution2: [0,6,2,18,3,7,20]**

Mutation operator: We introduce random noise within the solution by changing the values based on certain probability. Typically the mutation probability is kept low.

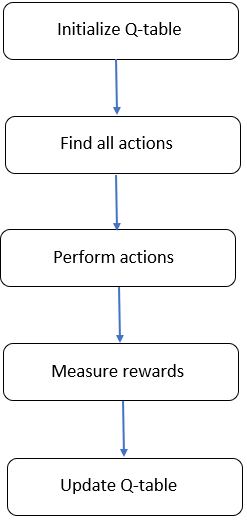
Selection operator: We have used random selection operator based on a probability values, there are other different types of selection operators like tournament, roulette etc which can be tried extending the work.

**Q learning algorithm:**

The Q learning algorithm is the simplest algorithm in reinforcement learning paradigm. It basically maintains a lookup table, called a Q-table, which keeps track of all the states and all possible actions from those states with the associated rewards. During training time we just chose a random node, find all possible actions from that node, calculate the reward associated for those actions and update the Q-table with the calculated reward from that state for each action. This forms the fundamental step of Q learning and is repeated until convergence. At convergence we have the path of maximum rewards from each node to the goal. During test time, the agent simply follows the path of maximum rewards.

Input setup: For the given problem we set up the reinforcement algorithm as a maximization problem, in the given adjacency matrix, a path from one node to the other is indicated by 0 or greater whereas the absence of a path is represented by negative weights. We block the nodes by setting up higher negative weights for each of the edges in the node.

Initialization: We will first make a N x N matrix, where N is the number of states, initialized with zeros. Each action corresponds to an edge from one node to the other. So for each node there will be N possible actions.



Find all actions : Then find all the valid actions from the current state based on the adjacency matrix.

Perform actions : Perform each action.

Measure Rewards : Calculate the rewards for performing that particular action from the current state based on the below formula:

Q\_table[current\_state,action] = Adjacency\_Matrix[current\_state,action] + gamma \* max\_value

where,

Q\_table = an N x N table which contains max reward lookup value from each state

current\_state = current state of the agent or node of the network

action = one of the valid actions at the current state

Adjacency\_Matrix = contains information about the edges between each pair of nodes

gamma = probability of choosing the action with max\_value at each step

max\_value = maximum reward value out of all the actions at the current state

Update Q-table : Update the Q-table with the calculated maximum reward value from each state.

**Experimental Evaluation:**

Origin: 0 Goal: 9

N: 10 Iterations: 1000 Data: data10-1000a.csv

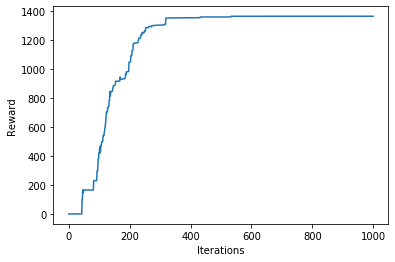
Sparsity: 0.65

Blocked: []

Path length: 3

Shortest path: [0, 1, 2, 9]

Time: 0.0785512924194336



Correct path

------------------------------------------------------------------------

Iteration 0: Best so far is 9 steps for a distance of 8.000000

best\_route [0 4 7 4 5 3 6 2 9]

Iteration 1: Best so far is 7 steps for a distance of 6.000000

best\_route [0 4 5 3 6 2 9]

Iteration 2: Best so far is 5 steps for a distance of 4.000000

best\_route [0 4 5 2 9]

Iteration 4: Best so far is 4 steps for a distance of 3.000000

best\_route [0 1 2 9]

============================================================================

Origin: 0 Goal: 9

N: 10 Iterations: 10 Data: data10-1000a.csv

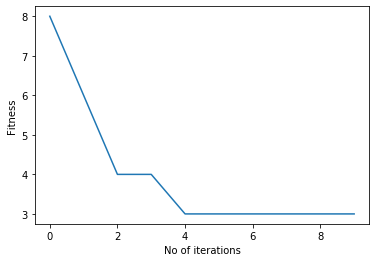
Sparsity: 0.65

Blocked: []

Path length: 3

Shortest path: [0 1 2 9]

Time: 0.05389881134033203



In [92]:

---------------------------------------------------------------------------------------------------------------------------

Origin: 0 Goal: 9

N: 10 Iterations: 1000 Data: data10-1000a.csv

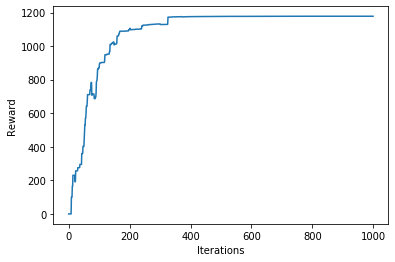
Sparsity: 0.65

Blocked: [1]

Path length: 4

Shortest path: [0, 4, 3, 2, 9]

Time: 0.06761837005615234



Correct path

---------------------------------------------------------------------------

Iteration 0: Best so far is 8 steps for a distance of 7.000000

best\_route [0 4 3 2 5 3 2 9]

Iteration 2: Best so far is 7 steps for a distance of 6.000000

best\_route [0 4 3 6 3 2 9]

Iteration 3: Best so far is 6 steps for a distance of 5.000000

best\_route [0 4 5 3 2 9]

Iteration 4: Best so far is 5 steps for a distance of 4.000000

best\_route [0 4 5 2 9]

============================================================================

Origin: 0 Goal: 9

N: 10 Iterations: 10 Data: data10-1000a.csv

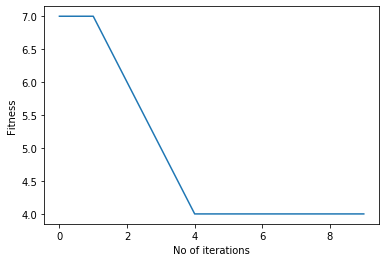
Sparsity: 0.65

Blocked: [1]

Path length: 4

Shortest path: [0 4 6 2 9]

Time: 0.05086398124694824



In [ ]:

-------------------------------------------------------------------------------------------------------------------------------

Origin: 0 Goal: 99

N: 100 Iterations: 6000 Data: data100-6000a.csv

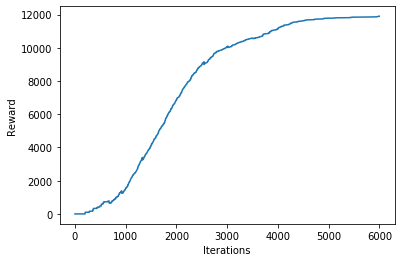
Sparsity: 0.95 gamma: 0.65

Blocked: []

Path length: 3

Shortest path: [0, 29, 17, 99]

Time: 0.6395816802978516



Correct path

In [32]:

Iteration 0: Best so far is 31 steps for a distance of 30.000000

best\_route [ 0 29 17 29 22 96 9 96 76 8 81 40 81 97 28 0 82 0 76 29 17 73 17 6

79 34 29 17 50 17 99]

Iteration 1: Best so far is 20 steps for a distance of 19.000000

best\_route [ 0 82 0 28 1 79 58 87 62 87 47 90 50 2 59 29 17 73 17 99]

Iteration 4: Best so far is 14 steps for a distance of 13.000000

best\_route [ 0 76 29 98 24 70 80 6 23 45 98 73 17 99]

Iteration 5: Best so far is 12 steps for a distance of 11.000000

best\_route [ 0 28 97 2 83 2 9 2 59 29 17 99]

Iteration 6: Best so far is 5 steps for a distance of 4.000000

best\_route [ 0 76 29 17 99]

Iteration 18: Best so far is 4 steps for a distance of 3.000000

best\_route [ 0 29 17 99]

============================================================================

Origin: 0 Goal: 99

N: 100 Iterations: 100 Data: data100-6000a.csv

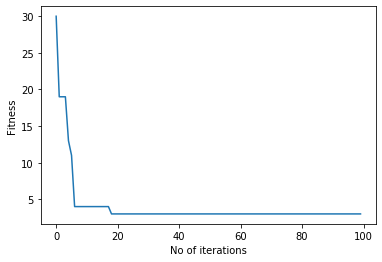
Sparsity: 0.65

Blocked: []

Path length: 3

Shortest path: [ 0 29 17 99]

Time: 3.432793140411377



In [ ]:

-----------------------------------------------------------------------

Origin: 0 Goal: 99

N: 100 Iterations: 10000 Data: data100-6000a.csv

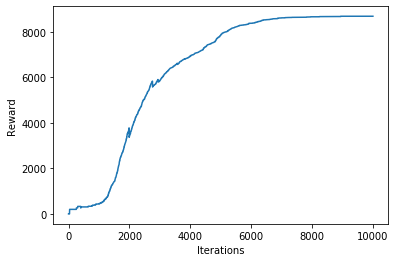
Sparsity: 0.95 gamma: 0.65

Blocked: [17]

Path length: 5

Shortest path: [0, 29, 34, 68, 95, 99]

Time: 3.067688465118408



Correct path

In [32]:

Iteration 0: Best so far is 19 steps for a distance of 18.000000

best\_route [ 0 29 2 38 35 65 35 15 38 2 59 29 13 77 70 26 66 95 99]

Iteration 2: Best so far is 18 steps for a distance of 17.000000

best\_route [ 0 29 59 63 59 2 36 34 24 15 25 22 37 67 26 66 95 99]

Iteration 4: Best so far is 10 steps for a distance of 9.000000

best\_route [ 0 29 98 4 57 70 26 66 95 99]

Iteration 31: Best so far is 9 steps for a distance of 8.000000

best\_route [ 0 82 0 1 91 23 33 95 99]

Iteration 51: Best so far is 7 steps for a distance of 6.000000

best\_route [ 0 1 91 23 33 95 99]

============================================================================

Origin: 0 Goal: 99

N: 100 Iterations: 100 Data: data100-6000a.csv

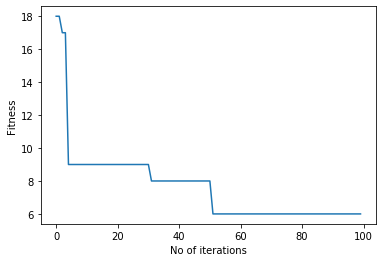
Sparsity: 0.65

Blocked: [17]

Path length: 6

Shortest path: [ 0 1 91 23 33 95 99]

Time: 2.945128917694092



In [ ]:

-----------------------------------------------------------------------

Origin: 0 Goal: 249

N: 250 Iterations: 20000 Data: data250-20000a.csv

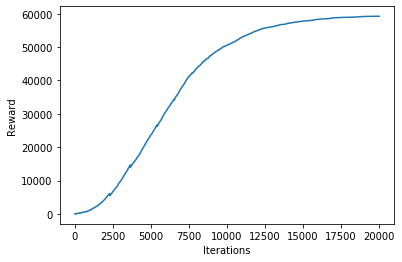
Sparsity: 0.97 gamma: 0.65

Blocked: []

Path length: 4

Shortest path: [0, 158, 10, 55, 249]

Time: 12.722759485244751



Correct path

---------------------------------------------------------------------------

Iteration 0: Best so far is 30 steps for a distance of 29.000000

best\_route [ 0 237 217 241 113 219 120 78 120 248 50 218 243 218 243 66 46 63

161 63 81 201 174 150 167 221 167 19 186 249]

Iteration 3: Best so far is 23 steps for a distance of 22.000000

best\_route [ 0 163 228 215 19 167 221 167 150 174 184 174 201 217 123 22 119 146

40 146 119 70 249]

Iteration 4: Best so far is 19 steps for a distance of 18.000000

best\_route [ 0 158 162 96 72 107 72 107 64 130 181 193 136 234 40 146 119 70

249]

Iteration 5: Best so far is 11 steps for a distance of 10.000000

best\_route [ 0 163 228 215 19 167 221 244 238 15 249]

Iteration 10: Best so far is 8 steps for a distance of 7.000000

best\_route [ 0 33 3 178 241 27 177 249]

Iteration 14: Best so far is 5 steps for a distance of 4.000000

best\_route [ 0 147 115 55 249]

============================================================================

Origin: 0 Goal: 249

N: 250 Iterations: 100 Data: data250-20000a.csv

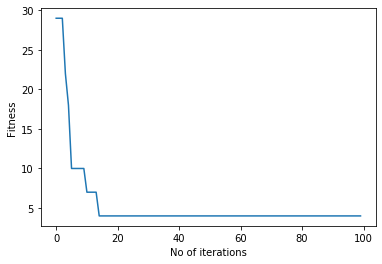
Sparsity: 0.65

Blocked: []

Path length: 4

Shortest path: [ 0 147 115 55 249]

Time: 6.893573760986328



In [ ]:

-----------------------------------------------------------------------

Origin: 0 Goal: 249

N: 250 Iterations: 40000 Data: data250-20000a.csv

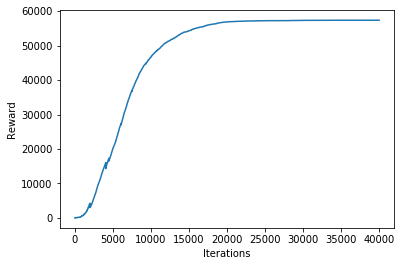
Sparsity: 0.97 gamma: 0.65

Blocked: [158, 10, 55]

Path length: 4

Shortest path: [0, 63, 169, 211, 249]

Time: 26.032241344451904



Correct path

-----------------------------------------------------------------------

Iteration 0: Best so far is 24 steps for a distance of 23.000000

best\_route [ 0 237 75 89 121 190 82 60 86 60 2 60 217 60 138 60 82 56

6 26 54 93 54 249]

Iteration 1: Best so far is 16 steps for a distance of 15.000000

best\_route [ 0 237 75 89 121 190 82 60 86 60 2 60 217 246 186 249]

Iteration 4: Best so far is 13 steps for a distance of 12.000000

best\_route [ 0 158 162 39 162 96 162 244 238 15 238 15 249]

Iteration 7: Best so far is 11 steps for a distance of 10.000000

best\_route [ 0 158 162 39 162 244 238 15 238 15 249]

Iteration 9: Best so far is 9 steps for a distance of 8.000000

best\_route [ 0 150 231 223 240 102 109 15 249]

Iteration 12: Best so far is 7 steps for a distance of 6.000000

best\_route [ 0 158 162 244 238 15 249]

Iteration 17: Best so far is 6 steps for a distance of 5.000000

best\_route [ 0 158 71 222 211 249]

============================================================================

Origin: 0 Goal: 249

N: 250 Iterations: 100 Data: data250-20000a.csv

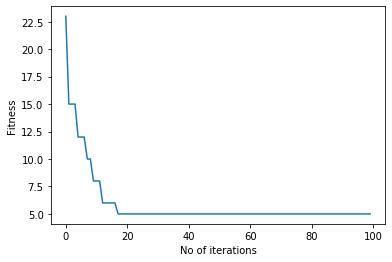
Sparsity: 0.65

Blocked: [147, 115, 55]

Path length: 5

Shortest path: [ 0 158 71 222 211 249]

Time: 2.981276750564575



In [ ]:

-----------------------------------------------------------------------

Origin: 0 Goal: 499

N: 500 Iterations: 70000 Data: data500-70000a.csv

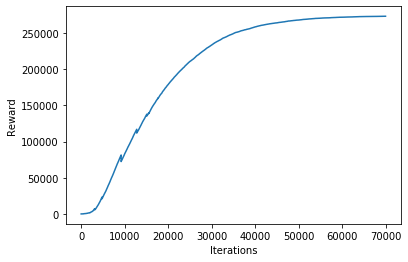
Sparsity: 0.97 gamma: 0.65

Blocked: []

Path length: 3

Shortest path: [0, 118, 354, 499]

Time: 193.21558046340942



Correct path

In [32]:

Iteration 0: Best so far is 58 steps for a distance of 57.000000

best\_route [ 0 154 0 402 153 205 12 431 213 185 481 275 58 181 197 102 173 125

60 494 125 497 51 286 96 286 96 201 20 437 187 95 318 298 24 219

51 429 165 36 165 220 491 44 11 194 142 150 142 75 72 210 56 251

468 12 238 499]

Iteration 1: Best so far is 57 steps for a distance of 56.000000

best\_route [ 0 357 358 301 243 424 486 407 281 272 215 385 474 454 30 221 135 56

29 106 359 372 253 169 58 465 58 16 453 301 367 137 85 137 187 374

353 51 280 186 280 307 419 345 169 299 116 111 245 29 292 107 292 282

147 470 499]

Iteration 2: Best so far is 28 steps for a distance of 27.000000

best\_route [ 0 118 326 270 318 95 498 10 439 370 326 413 412 125 55 226 55 490

55 291 250 331 100 51 99 472 470 499]

Iteration 4: Best so far is 9 steps for a distance of 8.000000

best\_route [ 0 103 295 391 194 406 280 208 499]

Iteration 8: Best so far is 7 steps for a distance of 6.000000

best\_route [ 0 154 183 363 299 414 499]

Iteration 22: Best so far is 6 steps for a distance of 5.000000

best\_route [ 0 57 312 227 453 499]

Iteration 31: Best so far is 4 steps for a distance of 3.000000

best\_route [ 0 118 380 499]

============================================================================

Origin: 0 Goal: 499

N: 500 Iterations: 100 Data: data500-70000a.csv

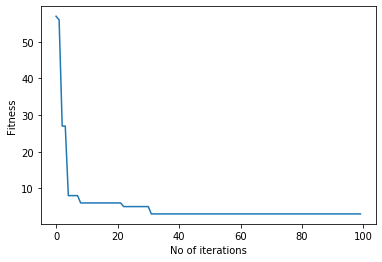
Sparsity: 0.65

Blocked: []

Path length: 3

Shortest path: [ 0 118 380 499]

Time: 4.009281873703003



In [ ]:

-----------------------------------------------------------------------

Origin: 0 Goal: 499

N: 500 Iterations: 70000 Data: data500-70000a.csv

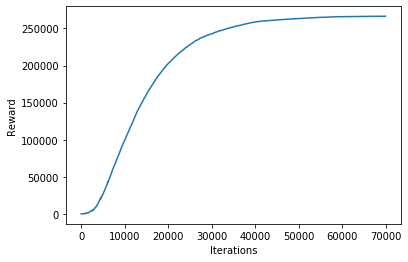
Sparsity: 0.97 gamma: 0.65

Blocked: [118, 397, 227, 453, 429, 104]

Path length: 3

Shortest path: [0, 154, 354, 499]

Time: 194.10709738731384



Correct path

-------------------------------------------------------------------------------------------------------------------------------

Iteration 0: Best so far is 8 steps for a distance of 7.000000

best\_route [ 0 41 220 51 128 125 15 499]

Iteration 24: Best so far is 7 steps for a distance of 6.000000

best\_route [ 0 319 117 123 233 169 499]

Iteration 70: Best so far is 4 steps for a distance of 3.000000

best\_route [ 0 334 441 499]

============================================================================

Origin: 0 Goal: 499

N: 500 Iterations: 100 Data: data500-70000a.csv

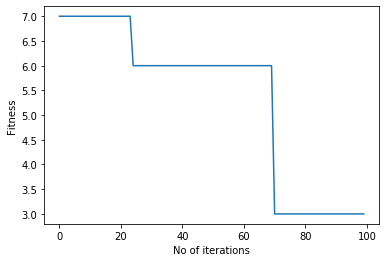
Sparsity: 0.65

Blocked: [118, 397, 227, 453, 429, 104]

Path length: 3

Shortest path: [ 0 334 441 499]

Time: 5.426528453826904



In [ ]:

-------------------------------------------------------------------------------------------------------------------------------

Origin: 0 Goal: 999

N: 1000 Iterations: 100000 Data: data1000-10000.csv

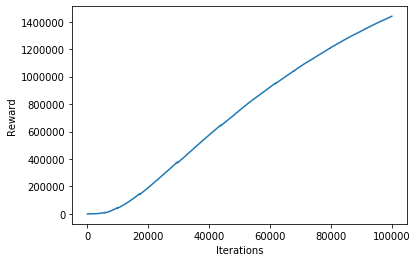
Sparsity: 0.97 gamma: 0.65

Blocked: []

Path length: 2

Shortest path: [0, 155, 999]

Time: 1540.7415761947632



Correct path

-------------------------------------------------------------------------------------------------------------------------------

Iteration 0: Best so far is 102 steps for a distance of 101.000000

best\_route [ 0 825 301 791 614 816 987 486 795 58 583 92 11 843 846 92 675 26

978 56 717 921 961 994 290 174 571 941 306 98 498 814 268 89 925 106

585 218 325 532 577 794 14 926 869 659 930 331 475 143 306 937 829 738

990 445 92 975 831 792 581 72 963 222 272 769 968 249 664 362 820 256

875 331 291 368 86 766 738 392 962 499 272 981 477 961 977 973 943 761

700 353 241 392 962 452 419 993 624 444 839 999]

Iteration 1: Best so far is 42 steps for a distance of 41.000000

best\_route [ 0 560 784 916 525 578 242 269 299 694 241 184 288 184 687 686 150 703

735 718 70 571 941 66 701 793 872 407 242 26 239 609 29 253 185 983

543 678 223 142 259 999]

Iteration 3: Best so far is 20 steps for a distance of 19.000000

best\_route [ 0 560 784 916 525 578 242 269 299 694 241 392 962 452 419 993 624 444

839 999]

Iteration 7: Best so far is 16 steps for a distance of 15.000000

best\_route [ 0 21 155 621 370 877 436 672 638 379 328 740 401 963 155 999]

Iteration 12: Best so far is 12 steps for a distance of 11.000000

best\_route [ 0 991 668 368 874 952 586 369 805 666 259 999]

Iteration 13: Best so far is 4 steps for a distance of 3.000000

best\_route [ 0 21 155 999]

Iteration 87: Best so far is 3 steps for a distance of 2.000000

best\_route [ 0 155 999]

============================================================================

Origin: 0 Goal: 999

N: 1000 Iterations: 100 Data: data1000-10000.csv

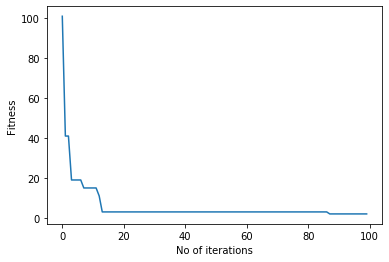
Sparsity: 0.65

Blocked: []

Path length: 2

Shortest path: [ 0 155 999]

Time: 35.36421990394592



In [ ]:

-------------------------------------------------------------------------------------------------------------------------------

Origin: 0 Goal: 999

N: 1000 Iterations: 50000 Data: data1000-10000.csv

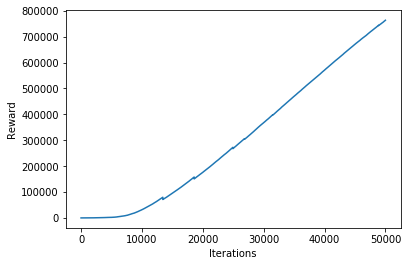
Sparsity: 0.97 gamma: 0.65

Blocked: [155]

Path length: 3

Shortest path: [0, 244, 542, 999]

Time: 1068.1458992958069



Correct path

-------------------------------------------------------------------------------------------------------------------------------

Iteration 0: Best so far is 35 steps for a distance of 34.000000

best\_route [ 0 567 708 810 580 805 901 448 804 192 756 539 181 167 310 122 30 902

538 679 704 565 478 316 620 638 855 42 414 292 181 724 286 251 999]

Iteration 4: Best so far is 16 steps for a distance of 15.000000

best\_route [ 0 567 708 810 580 805 901 448 804 192 756 539 181 167 16 999]

Iteration 6: Best so far is 15 steps for a distance of 14.000000

best\_route [ 0 621 389 706 747 191 284 186 853 891 92 181 167 16 999]

Iteration 12: Best so far is 9 steps for a distance of 8.000000

best\_route [ 0 621 365 700 491 602 662 809 999]

Iteration 27: Best so far is 5 steps for a distance of 4.000000

best\_route [ 0 875 337 632 999]

============================================================================

Origin: 0 Goal: 999

N: 1000 Iterations: 100 Data: data1000-10000.csv

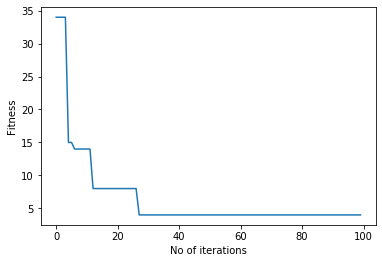
Sparsity: 0.65

Blocked: [155]

Path length: 4

Shortest path: [ 0 875 337 632 999]

Time: 52.747963428497314



In [ ]: