1. Greedy algorithm

The original unsorted list at a test size of 500 has a total Euclidean distance of 327.23386716221. After the algorithm has ran its Euclidean distance goes to 0

The original unsorted list at a test size of 100 has a total Euclidean distance of 64.571629852396. After the algorithm has ran its Euclidean distance goes to 0

For both sample sizes this shows that the overall Euclidean distance is getting smaller therefore the algorithm is working.

Ten runs at 100 colours:

--- 0.0040018558502197266 seconds for Greedy ---

--- 0.005005598068237305 seconds for Greedy ---

--- 0.0030045509338378906 seconds for Greedy ---

--- 0.0030019283294677734 seconds for Greedy ---

--- 0.004004001617431641 seconds for Greedy ---

--- 0.00400233268737793 seconds for Greedy ---

--- 0.005003213882446289 seconds for Greedy ---

--- 0.0030014514923095703 seconds for Greedy ---

--- 0.0040018558502197266 seconds for Greedy ---

--- 0.003001689910888672 seconds for Greedy ---

Average produced = 0.0038028478622437

Ten runs at 500 colours:

--- 0.05428814888000488 seconds for Greedy ---

--- 0.0610353946685791 seconds for Greedy ---

--- 0.05803394317626953 seconds for Greedy ---

--- 0.07204008102416992 seconds for Greedy ---

--- 0.05303072929382324 seconds for Greedy ---

--- 0.05603146553039551 seconds for Greedy ---

--- 0.0540313720703125 seconds for Greedy ---

--- 0.05402994155883789 seconds for Greedy ---

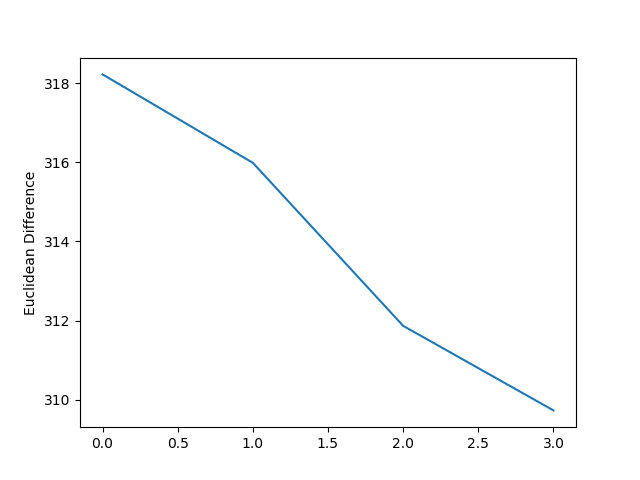
--- 0.05803322792053223 seconds for Greedy ---

--- 0.054032087326049805 seconds for Greedy ---

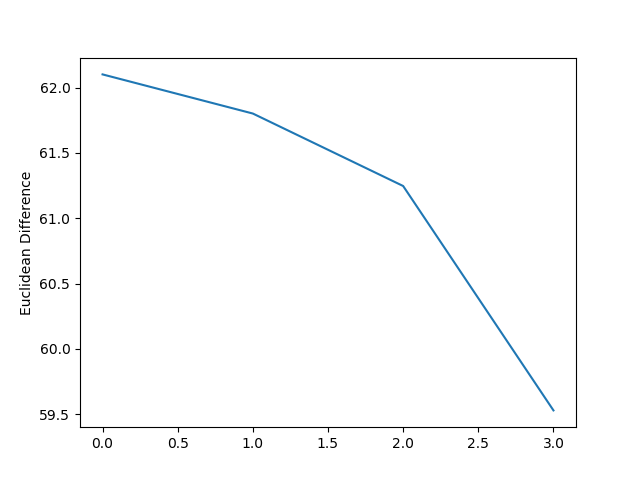
Average produced = 0.057458639144897

1. Hill Climb (single run)

500 test size:



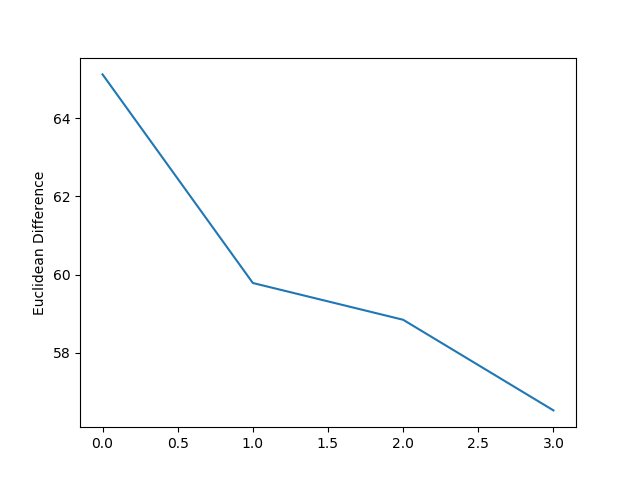
100 test size:



We used the simple neighborhood move operator that choses two random values in the array and then swaps those values. This operator was the simplest to implement and used the least computational power. As the graphs show there is a steady decrease in the total Euclidean distance, thus getting closer to the overall solution.

1. Multi Run Hill climb

100 Test Run:

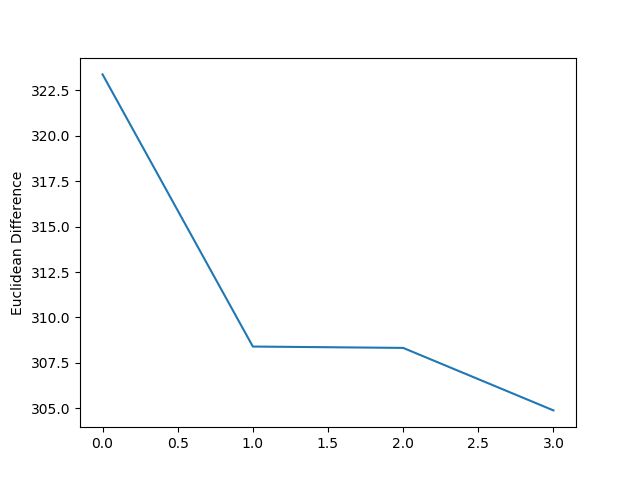


Median - 64.41129416846285

Mean - 64.762615579671

SD - 1.3630962400064

500 Test Run



Median - 326.36406586873

Mean - 326.34296160568

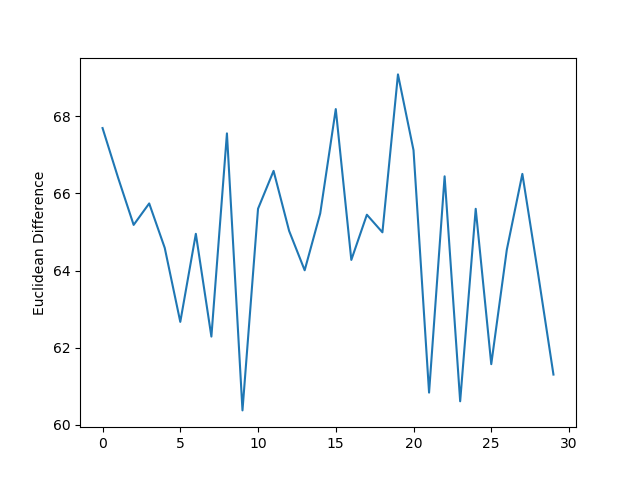
SD - 3.438583352991

As the graphs show there is a steady decrease in the total Euclidean distance, thus getting closer to the overall solution. There is a stepper decline compared to the single run hill climb. This is because we are comparing solutions that are at different points along the solution curve and comparing the totals to find the global optima as opposed to starting in a single position and only searching for that local optima.

1. Our Search (Tabu Search)

We used the Tabu search as our own algorithm. A Tabu search starts with a random solution s. It then gets a random set of N’ neighbors of s. It then looks to see which one of those neighbors is the next best solution. The Tabu search will always make a change even if it is worse than the current best solution. The Tabu search also has a list of Tabu moves in which the algorithm will not go to. This list can be previous best solutions, so the algorithm doesn't go back on itself.

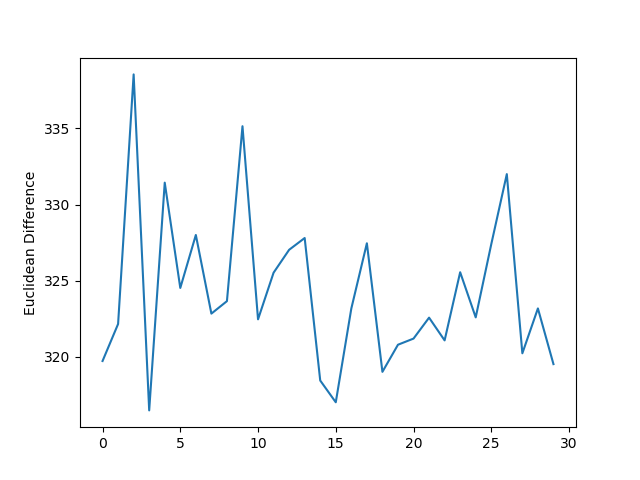
We used the simple neighborhood move operator that choses two random values in the array and then swaps those values. This operator was the simplest to implement and used the least computational power.

100 Test Run

Median - 65.106353367913

Mean - 64.821415412943

SD - 2.3050446213672

500 Test Run

Median - 323.00771066762

Mean - 324.21998415479

SD - 5.1654483550546

1. Summary

I do not think that our search was as efficient as the multi-run hill climbing but better than singe run hill climb and greedy as it was able to compare previous solutions and improve based on where it had been before through the use of search history.