Traveling Salesman Problem (TSP) Approximate Solution

By: Samuel Snyder & Andrew Bailey

Hill Climbers Pseudocode (1/2)

```
fun hillClimbers():
    s ← random initial solution
    best ← bestNeighbor(s)
    while length(best) < length(s) do:
        s ← best
        sNeighbors ← get neighbors of s
        best ← get best neighbor from neighbors
    return s
```

Hill Climbers Pseudocode (2/2)

```
fun getNeighbors(s):
    neighbors ← []
    for each vertex in s:
        n ← swap two vertices
        Neighbors ← append n
    return neighbors
```

```
fun getBestNeighbors(neighbors):
    best ← neighbors[0]
    for n in neighbors:
        len ← length(neighbor)
        best ← neighbor if len <
            length(best)
    return best
```

Runtime Analysis of Hill Climbers

The hill climbers algorithm is a greedy approach, where out of a list of neighbors, it chooses the shortest path. The stochastic (random) part of the algorithm occurs outside, where in this instance, it is run k times. Our application of hill climbers runs in O(n), given the while loop and for loops called sequentially.

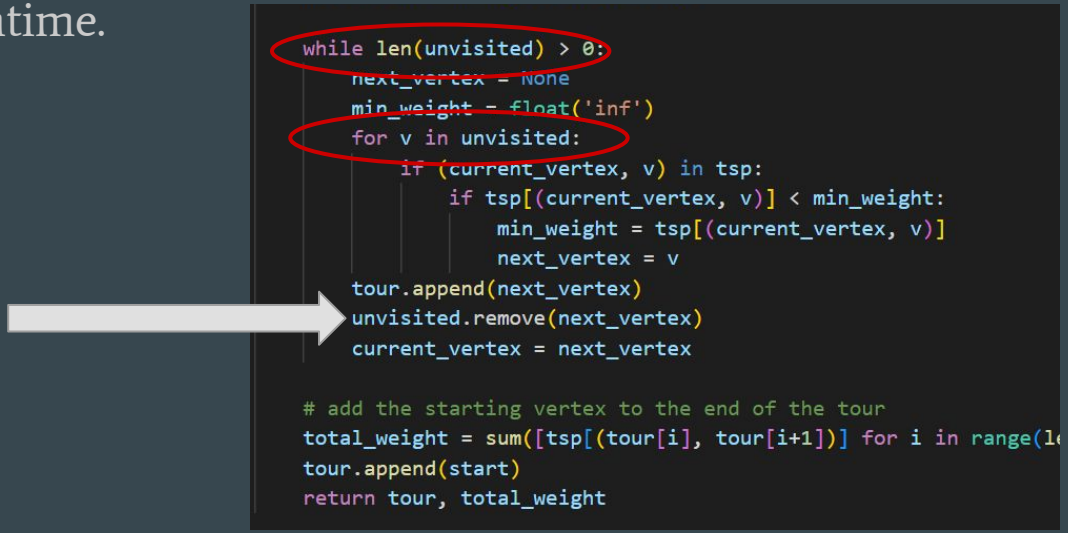
```
def hillClimbing(tsp, vertices, s):
    currSolution = randomSolution(vertices, s) # initial solution
    currLength = calculate(tsp, currSolution)
    neighbors = getNeighbors(currSolution)
    bestNeighbor, bestLength = getBestNeighbor(tsp, neighbors)
    while bestLength < currLength:
        currSolution = bestNeighbor
        currLength = bestLength
        neighbors = getNeighbors(currSolution)
        bestNeighbor, bestLength = getBestNeighbor(tsp, neighbors)

    return currSolution, currLength
```

Nearest Neighbor Pseudocode

```
fun nearestNeighbor(vertices, start):
    notVisited← copy of vertices, without start
    tour ← [] with only start
    while length(notVisited) > 0:
        for v in unvisited:
            if current and v in our tsp AND less than minWeight:
                minWeight ← weight(current, v)
                next = v
        tour ← append next
        notVisited ← remove next
        current ← next
    return tour, length(tour)
```

Runtime Analysis of Nearest Neighbor

An even more greedy approach than hill climbers, nearest neighbor chooses a random start point and loops over all neighbors of a vertex, choosing the nearest. Loops through all vertices. Each iteration of the outer while loop removes a vertex from the unvisited list, meaning one less iteration of the inner for loop each time (n= 20; 20, 19, 18, ...). This gives it an O(n) runtime.



Not-Optimal

```
8 28
0 1 926.974
0 2 802.029
0 3 32.391
0 4 86.102
0 5 334.62
0 6 222.973
0 7 328.56
1 2 274.376
1 3 405.247
1 4 455.476
1 5 694.863
1 6 771.772
1 7 136.132
2 3 807.359
2 4 540.484
2 5 627.947
2 6 891.424
2 7 927.76
3 4 120.32
3 5 902.19
3 6 636.726
3 7 348.929
4 5 164.104
4 6 78.061
4 7 609.047
5 6 938.167
5 7 694.479
6 7 626.596
```

On specific iterations, approximate will not return optimal results.

When starting on vertex 6, the shortest path is 2522.105, while the optimal path is 1884.913, which can be achieved when starting from multiple other vertices.

The optimality of the program relies on its random starting point, so more iterations means greater chance of near-optimality.

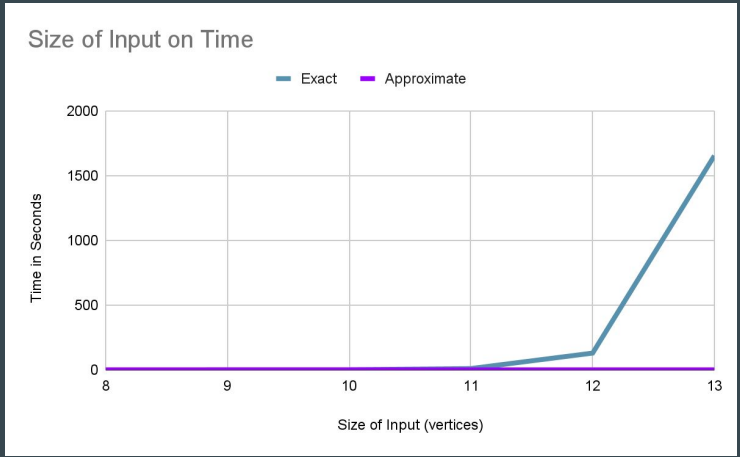
```
2522.105
6 4 0 3 7 1 2 5 6
```

```
1884.913
5 4 6 0 3 7 1 2 5
```

```
1884.9130000000002
4 6 0 3 7 1 2 5 4
```

Wall Clock Runtime Analysis

Compared to the exact solution in the graph, approximate looks near linear. For smaller inputs (tests 1 & 2), the time is similar, but deviates greatly for greater inputs (where test 5 is on input of size 13). The drawback is that it does not find the shortest path (1989 vs. 2444)

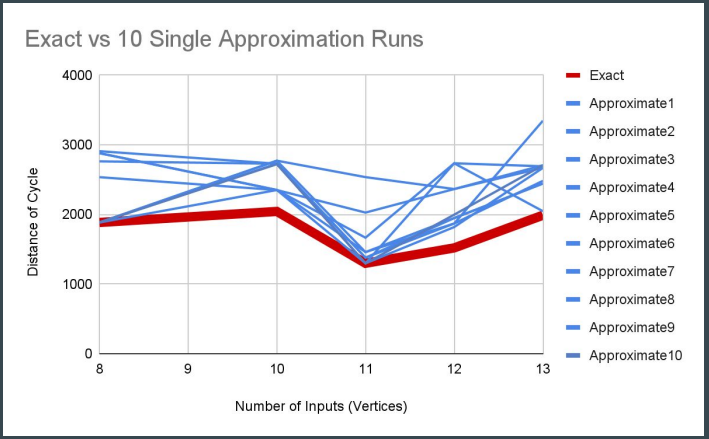



```
(base) sameulsnyder@MacBook-Pro-2 exact_solution
Running test case: test_cases/test_1.txt
1884.913
2 1 7 3 0 6 4 5 2

real    0m0.046s
user    0m0.031s
sys     0m0.011s
Done running test case: test_cases/test_1.txt

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

Running test case: test_cases/test_2.txt
2043.3399999999997
4 2 8 9 0 6 7 1 3 5 4

real    0m0.931s
user    0m0.918s
sys     0m0.009s
Done running test case: test_cases/test_2.txt

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

Running test case: test_cases/test_3.txt
1298.612
8 2 7 4 3 1 10 0 9 5 6 8

real    0m10.135s
user    0m10.038s
sys     0m0.038s
Done running test case: test_cases/test_3.txt

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

Running test case: test_cases/test_4.txt
1519.59
4 5 7 0 11 2 6 3 1 9 10 8 4

real    2m9.335s
user    2m8.116s
sys     0m0.338s
Done running test case: test_cases/test_4.txt

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

Running test case: test_cases/test_5.txt
1989.452
3 9 8 10 12 0 7 4 11 6 2 5 1 3

real    26m50.013s
user    26m40.570s
sys     0m2.623s
Done running test case: test_cases/test_5.txt

------------------------------------
```

```
(base) sameulsnyder@MacBook-Pro-2 approximate_s
Running test case: test_cases/test_1.txt
1884.9130000000002
4 6 0 3 7 1 2 5 4

real    0m0.028s
user    0m0.016s
sys     0m0.009s
Done running test case: test_cases/test_1.txt

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

Running test case: test_cases/test_2.txt
2729.7709999999997
7 1 8 2 4 5 0 6 9 3 7

real    0m0.025s
user    0m0.016s
sys     0m0.008s
Done running test case: test_cases/test_2.txt

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

Running test case: test_cases/test_3.txt
1666.296
7 4 9 5 6 8 2 3 1 10 0 7

real    0m0.025s
user    0m0.015s
sys     0m0.009s
Done running test case: test_cases/test_3.txt

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

Running test case: test_cases/test_4.txt
1819.1800000000003
7 5 4 9 1 3 8 10 6 2 11 0 7

real    0m0.027s
user    0m0.017s
sys     0m0.009s
Done running test case: test_cases/test_4.txt

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

Running test case: test_cases/test_5.txt
2444.406
9 8 6 11 4 7 0 1 5 2 12 10 3 9

real    0m0.023s
user    0m0.014s
sys     0m0.007s
Done running test case: test_cases/test_5.txt

------------------------------------
```

Results Analysis

Our nearest neighbors algorithm relied on the randomness of that starting position to make the best choices for the cycle. As such, the variance of the data is high but will sometimes come up with nearly the exact solution.




```
(base) sameulsnyder@MacBook-Pro-2 exact_solution
Running test case: test_cases/test_1.txt
1884.913
2 1 7 3 6 4 5 2

real    0m0.046s
user    0m0.031s
sys     0m0.011s
Done running test case: test_cases/test_1.txt

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

Running test case: test_cases/test_2.txt
2043.3399999999997
4 2 8 0 6 7 1 3 5 4

real    0m0.931s
user    0m0.918s
sys     0m0.009s
Done running test case: test_cases/test_2.txt

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

Running test case: test_cases/test_3.txt
1298.612
8 2 7 4 3 1 10 0 9 5 6 8

real    0m10.135s
user    0m10.038s
sys     0m0.038s
Done running test case: test_cases/test_3.txt

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

Running test case: test_cases/test_4.txt
1519.59
4 5 7 0 11 2 6 3 1 9 10 8 4

real    2m9.335s
user    2m8.116s
sys     0m0.338s
Done running test case: test_cases/test_4.txt

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

Running test case: test_cases/test_5.txt
1989.452
3 9 8 10 12 0 7 4 11 6 2 5 1 3

real    26m50.013s
user    26m40.570s
sys     0m2.623s
Done running test case: test_cases/test_5.txt

---------------------------------------
```

```
(base) sameulsnyder@MacBook-Pro-2 approximate_s
Running test case: test_cases/test_1.txt
1884.9130000000002
4 6 0 3 7 1 5 2

real    0m0.028s
user    0m0.016s
sys     0m0.009s
Done running test case: test_cases/test_1.txt

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

Running test case: test_cases/test_2.txt
2729.7709999999997
7 1 8 2 4 5 0 6 3 9 7

real    0m0.025s
user    0m0.016s
sys     0m0.008s
Done running test case: test_cases/test_2.txt

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

Running test case: test_cases/test_3.txt
1666.296
7 4 9 5 6 8 2 3 1 10 0 7

real    0m0.025s
user    0m0.015s
sys     0m0.009s
Done running test case: test_cases/test_3.txt

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

Running test case: test_cases/test_4.txt
1819.1800000000003
7 5 4 9 1 3 8 10 6 2 11 0 7

real    0m0.027s
user    0m0.017s
sys     0m0.009s
Done running test case: test_cases/test_4.txt

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

Running test case: test_cases/test_5.txt
2444.406
9 8 6 11 4 7 0 1 5 2 12 10 3 9

real    0m0.023s
user    0m0.014s
sys     0m0.007s
Done running test case: test_cases/test_5.txt

---------------------------------------
```

Results Analysis Quick Fix

In order to combat the heavy outliers from the previous graph, we make use of the polynomial runtime to run the nearest neighbours program a constant more amount of times (in this case 100 more times so it would be $O(100n) \rightarrow O(n)$)

This way we can get closer to finding the optimal starting position of the algorithm by sacrificing a marginal amount of speed on larger inputs.