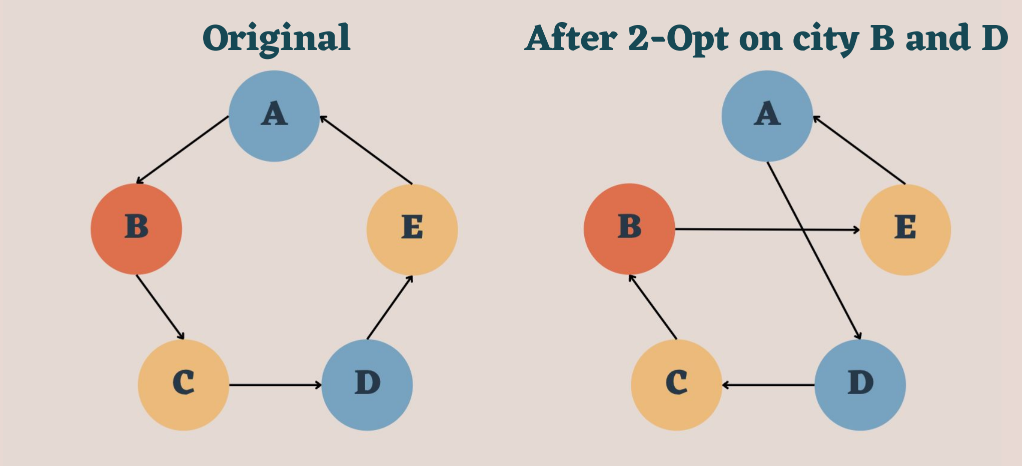
The 2-Opt Algorithm

Efficient Solutions to the Traveling Salesman Problem

Group 1

What is 2-Opt?

2-Opt is a TSP local search algorithm, proposed by Croes in 1958, that builds itself from an already established tour. In our implementation we used the greedy algorithm as a starting point. 2-Opt iterates through the cities in this tour and compares different potential routes from the current city to other cities in the path. If the cost of switching the path is cheaper, 2-Opt swaps the path and continues to build on the path.



2-Opt: Pros and Cons

Pros

- ➢ Fast run time
- ➢ Space efficient
- ➢ Simple to implement

Cons

- ➢ Circuits aren't always clean
- ➢ Optimal Paths are not guaranteed
- ➢ Small Reductions on Greedy Approach
- ➢ 2-Opt doesn't work if Greedy doesn't give a path

Challenges Faced

- ➢ 2-Opt is typically used with symmetric traveling salesman problems, meaning we needed to consider how infinite edges would affect the algorithm with the section of the path that has to reverse. We faced this challenge by implementing two things:
  - A function to validate reverse paths in the Linked List
  - A k-value to limit how far we travel from the current city to save time
- ➢ To avoid having to iterate through the linked list to calculate the new cost when a swap is made, when we evaluated a reversed section we calculated both the forward and backward cost of the section and choose the cheaper one. This meant evaluated cost only took k iterations instead of n. In large samples, this can save a lot of time.

Theoretical Complexity of 2-Opt

Get Initial Solution from Greedy	$O(n^3)$ time	$O(n^2)$ space	
Generate the cost matrix	$O(n^2)$ time	$O(n^2)$ space	
Generate the linked-list	$O(n)$ time	$O(n)$ space	
Iterate through at most n times:	$O(n)$ time	$O(1)$ space	$O(n^2k^2)$ time $O(k)$ space
Iterate over every city:	$O(n)$ time	$O(1)$ space	
Iterate over the next k cities:	$O(k)$ time	$O(k)$ space	
Check for 2-Opt swaps	$O(k)$ time	$O(1)$ space	
Perform the best swap	$O(k)$ time	$O(1)$ space	

TOTAL: $O(n^3)$ time $O(n^2)$ space

2-Opt vs Greedy vs Branch and Bound

	Random		Greedy		Branch and Bound			2-Opt (k = 50)				
# Cities	Time(sec)	Path Length	Path Length	% of Random	Time (sec)	Path Length	% of Greedy	Time (sec)	Path Length	% of Greedy	Swaps Made	Swaps Found
15	0.002	21401.2	13189	61.63	1.141	10359	78.55	0.01	10772.2	81.68	3.8	5.8
30	0.055	42669.8	21336.4	50.01	TB	TB	N/A	0.021	20860.6	97.77	3	3.2
60	28.65	81345.8	27842	34.23	TB	TB	N/A	0.089	26312.6	94.51	9.6	11
100	TB	TB	37309	N/A	TB	TB	N/A	0.123	35926.8	96.3	9.2	10.8
200	TB	TB	58241.2	N/A	TB	TB	N/A	0.248	56483.2	96.98	18	20.6
1,000	TB	TB	161956.8	N/A	TB	TB	N/A	3.19	158265.4	97.72	64.2	73.8
10,000	TB	TB	704815.6	N/A	TB	TB	N/A	245.394	693006.4	98.32	482.8	519.2

The table above shows that 2-Opt combined with the greedy approach is a very fast solution to the traveling salesman problem. We were able to run samples of up to 10,000 cities without using an excessive amount of computation time. This approach does come at a hit to optimality. 2-Opt typically reduces the greedy approach by 2-10%, less than the Branch and Bound algorithm. However, in large samples 2-Opt is a much more efficient algorithm.

Additional Improvements

- Calculate optimal k value
- Test if a changing k value improves time
- Modify swap calculations to include prior calculations
- Make additional iterations through the cities skip previously calculated invalid swaps
- Implement the 3-Opt as well as 2-Opt