Heuristics Part 3: Local Search

Video (16 mins): https://youtu.be/XUNGtxoBbPQ

Previous Strategies for Solving TSP

- Eye-balling
- Exhaustive search
  - consider all possible tours, resulting in true optimal value
  - complexity is O(`n!`)
- Random search
  - consider some number `k` of randomly chosen tours
  - complexity is O(`kn`)
  - depending on chance, the results may get close or far from the optimal value
- Greedy search
  - construct solution step by step by optimizing local decisions
  - complexity is $O(n^2)$

Local Search Algorithm

- In a greedy algorithm, we construct a solution step by step
  - In the intermediate steps, we do not have a complete solution.
  - Advantages: simple to implement, fast running time.
- A complementary strategy is local search algorithm
  - Start with an arbitrary complete solution (e.g., random or greedy solution).
  - Iteratively make small modifications to the solution to improve the value.

Advantages and Disadvantages

- Advantages:
  - We can run as many iterations as we can afford to (recall random search). With more iterations, we are more likely to get closer to the optimal value.
  - At any point, we can stop the iterations, and we always have a complete solution (not necessarily optimal).
- Disadvantage:
  - If the starting solution turns out to be an especially bad choice, it may need a large number of iterations to get to a good solution.

Local Search Algorithm for TSP

- Define a neighborhood.

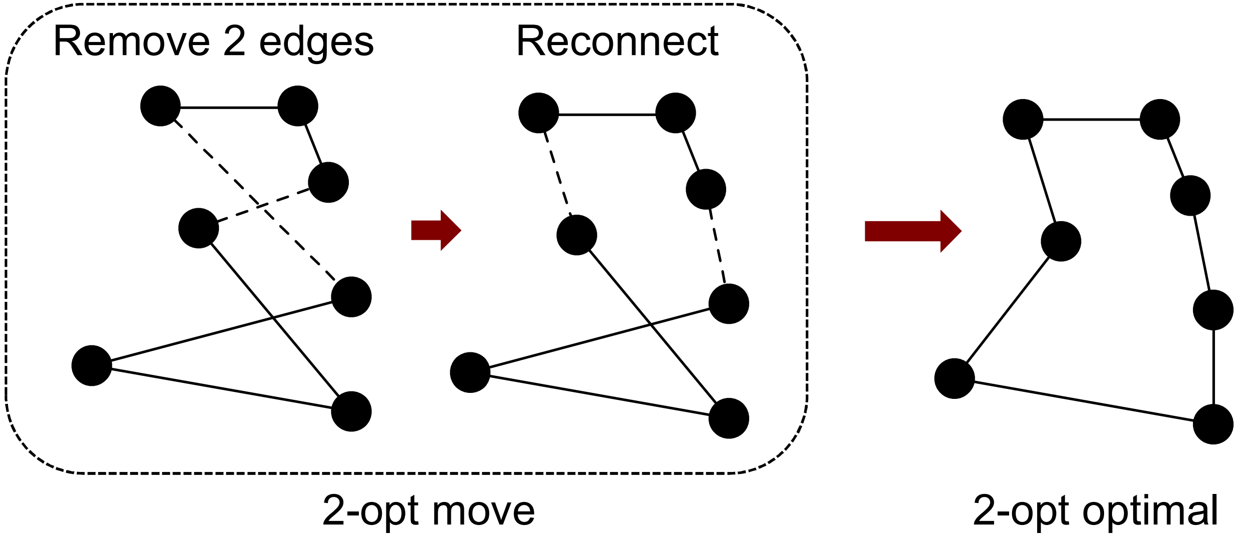
- Example: "**Edge Exchange**" neighborhood:

  Delete *k* edges in the current tour and then add *k* edges which form a new feasible tour.

- This neighborhood is called *k*-opt.

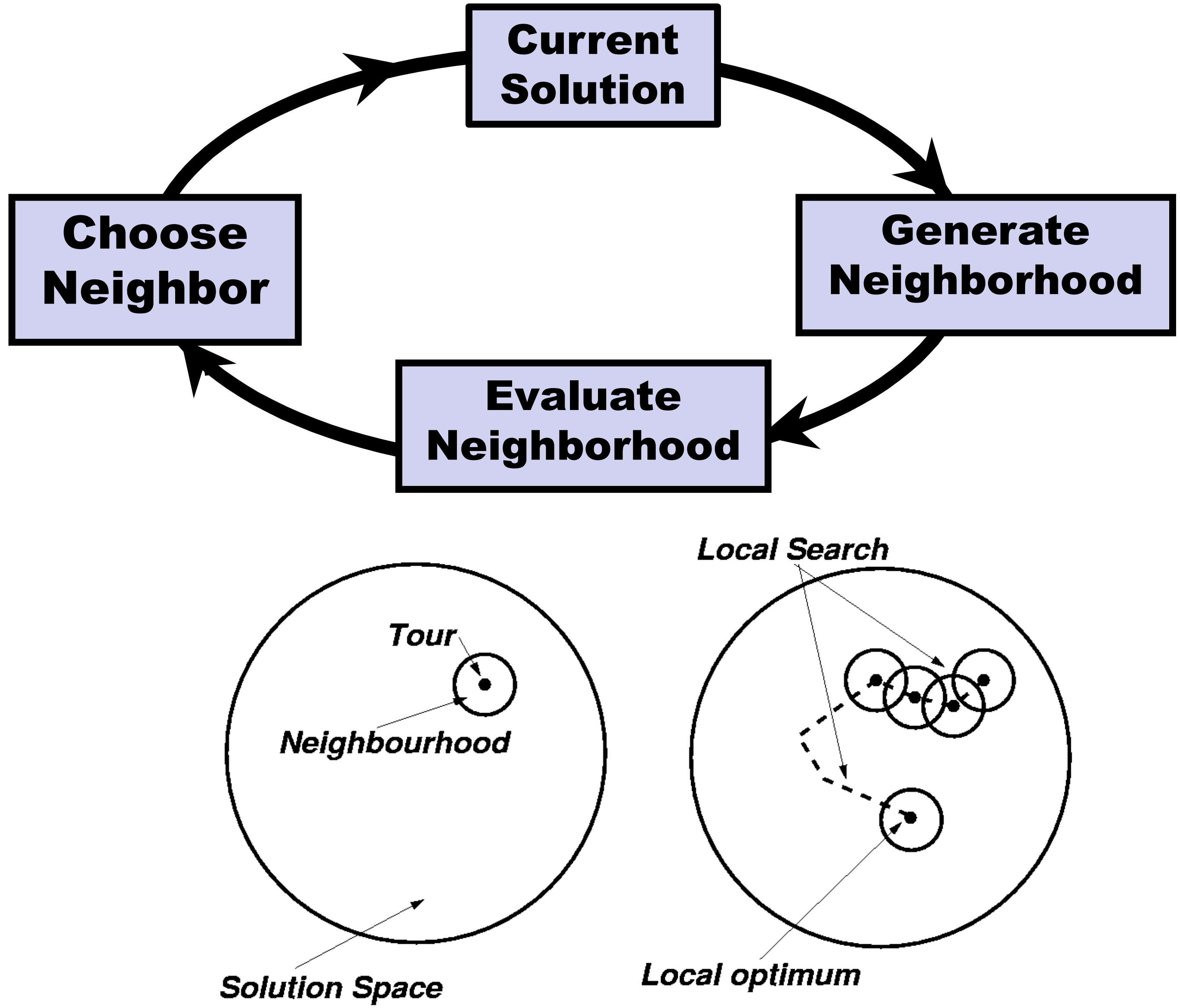
2-opt Neighborhood

- Delete 2 edges, add 2 edges to restore the tour



- 2-opt removes the "crossings" of edges in a tour
- Animation: http://pedrohfsd.com/2017/08/11/2opt-part2.html

Local Search Algorithm for TSP



2-opt Analysis

- Size of the 2-opt neighborhood:
  - Number of possible edge pairs = $n(n-3)/2 = O(n^2)$
  - Why?

- At each iteration, generate and evaluate $O(n^2)$ new tours and choose the best available.

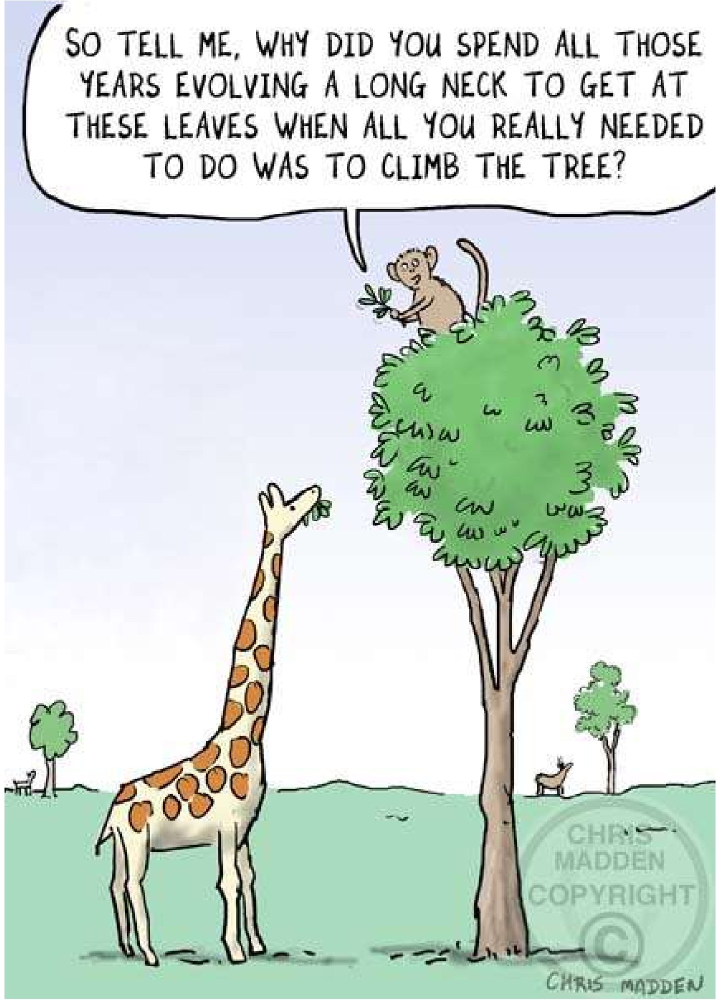
⇨ Worst-case run time: $O(n^2)$

TSP Support in GraphLab.py

- `two_opt(sequence, graph)`
  - Returns the new sequence of vertices

- GraphLab.py can be found in eLearn/Contents/wk9

Improving Solutions with Evolution



- Survival of the fittest.
- Solutions mate and exchange their "genes".
- Occasionally genes mutate.
- Through "natural selection", desirable features are passed on to the next generation.
- This is called Genetic Algorithm.

Reference: JSC Chapter 12

Optional: More About GA

- Read the following from Chapter 12 (TSP) of JSC
  - Point Mutations (pp. 407-410 of the PDF)
  - The Genetic Algorithm (pp. 410-417)
  - Crossovers (pp.417-419)
  - There is no need to try out the “Tutorial Projects”, but it may help to glance through the steps to get a rough idea of what the “Tutorial Projects” are trying to illustrate.

Summary

- The Traveling Salesman problem (TSP) is too complex to solve with a straightforward test of all possible paths.
  - a map with 25 cities has over 3×10^{23} tours.
- Instead of examining all possible tours, we try random samples.
  - make n random tours, return the one with the lowest cost.
- We can get good enough solutions by heuristics.
  - *Greedy algorithm*: construct the solution step by step by optimizing local decisions.
  - *Local search algorithm*: start with any solution, and improve it iteratively
    - 2-OPT is a powerful local improvement operator.
  - *Genetic algorithm*: use ideas inspired by natural selection
    - By looking at only 25,000 out of the 3×10^{23} possible tours, `esearch` might find the optimal tour.