

Artificial Intelligence: Traveling Salesperson Problem, Project 1

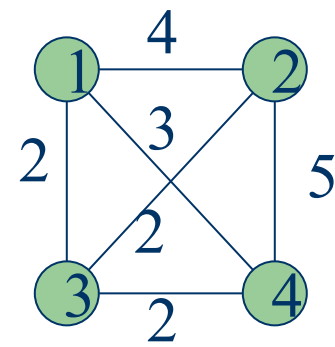
Course: CSE545
Artificial Intelligence
University of Louisville

Slides by Steve Wolfman

Dr. Roman V. Yampolskiy

TSP

- NP-complete problem in combinatorial optimization studied in computer science.
- Given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city (except the last one) exactly once.



Formal Problem Definition

- *Instance*: n vertices (cities), distance between every pair of vertices
 - (Alternatively locations of cities)
- *Question*: Find shortest (simple) cycle that visits every city

Learning Goals for Today

After today's unit, you will be able to:

- define the Traveling Salesperson Problem (TSP) clearly as an algorithmic problem
- outline the code for the “random”, “greedy”, and “random restart” approaches to solving TSP

Traveling Salesperson Problem (TSP)



Also TSP



Laser-carving a custom computer chip.



A More General Definition of TSP

Input:

- a list of “cities” by name; each name is unique
- a list of costs: one cost for each possible trip from one city in the list to another

Output: a tour (list of “cities” by name) that includes each input city exactly once, except that the first city also appears at the end.

Ideally, the output tour should be the cheapest of all possible tours, given the list of costs.


This works even when the costs aren't just distance, as with airline ticket prices, which may be cheaper in one direction than another or cheaper for a longer flight than a shorter one.



The “Random” Approach



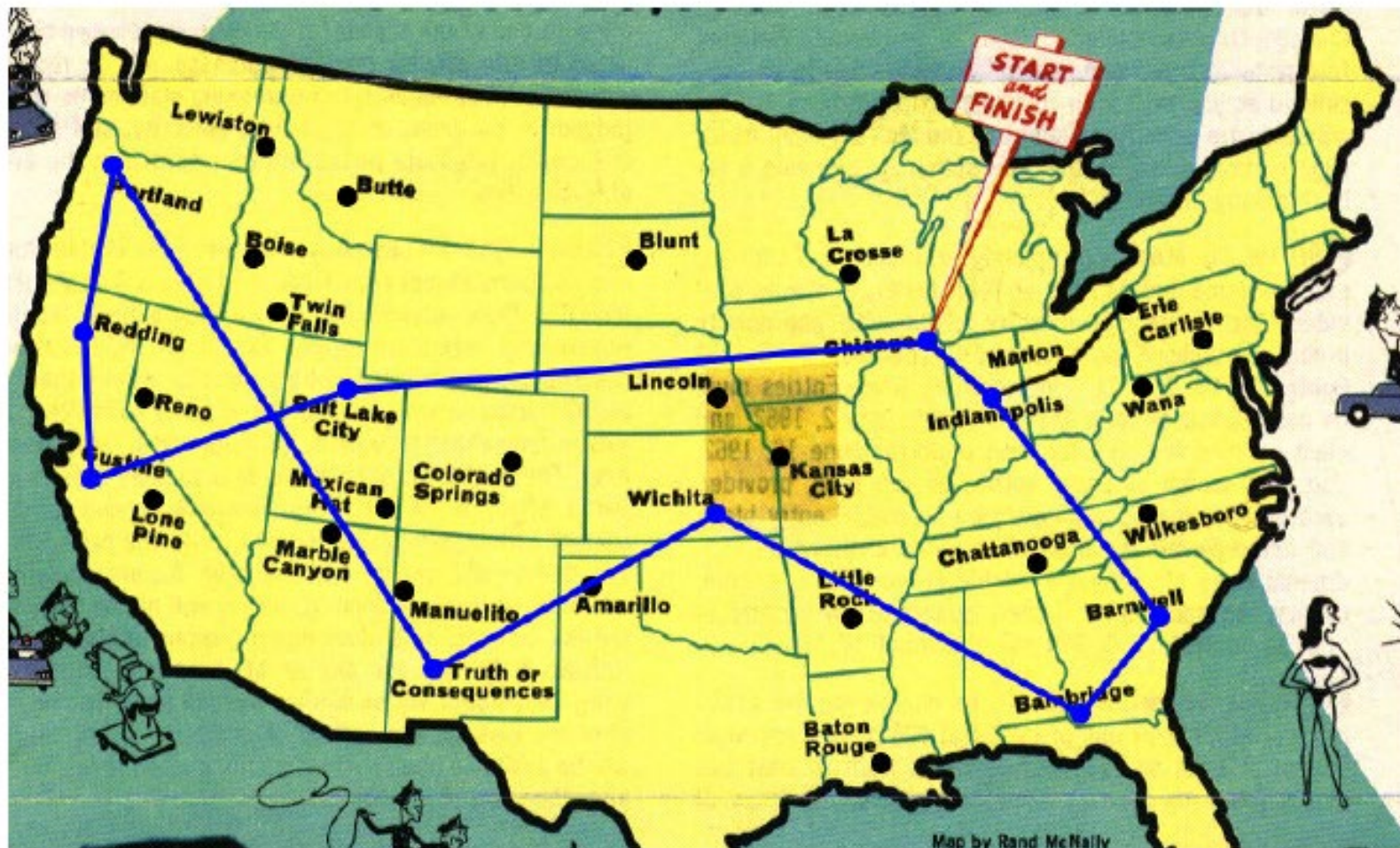
Algorithm:

1. Visit (start at) the start city.
 2. As long as some city has not been visited:
 - a) Pick one of the unvisited cities at random.
 - b) Visit that city.
 3. Return to the start city.
- 

Sample Random Solution...



A coincidentally very GOOD
random solution...



How could you improve this solution?
How could you make an algorithm to find and execute the improvement?



Problems with Random?

Does random always give us an answer?

Does it always give us a good answer?

Does it always give us the best answer?

If not, when does it go wrong?



The “Greedy” Approach

Algorithm:

1. Visit (start at) the start city.
2. As long as some city has not been visited:
 - a) Look through all unvisited cities and pick the closest.
 - b) Visit that closest city.
3. Return to the start city.

There is a large class of “greedy” algorithms. They are greedy because they make what looks like the best choice *right now*.

A Sample Greedy Solution...



Greedy again on the same problem...



Why is it the same?

Problems with Greedy?

Does greedy always give us an answer?

Does it always give us a good answer?

Does it always give us the best answer?

If not, when does it go wrong?



The “Random Restarts” Approach


Algorithm:

1. Solve the problem using the random approach.
2. Record the result as the “best solution so far”.
3. Do the following ***some number*** of times:
 - a) Solve the problem using the random approach.
 - b) If the result is better than the best solution so far, record this better result as the best so far.
4. Report the best solution so far.

How many times you do step 3 is an input to the algorithm.
But computers are fast; so, think big: 1000? 10000? 100000?



Problems with Random Restart?



Does it always give us an answer?

Does it always give us a good answer?

Does it always give us the best answer?

If not, when does it go wrong?



How Many Tours Are There?

Look just at the
Northwest.

How many tours go
through these 6 cities?

Hint: pick a starting city.

Now, how many next
cities are there?

How many after that?



Be careful... there's still one more trick here!

Hint: does the best tour starting at Lewiston go to Butte or Portland first?

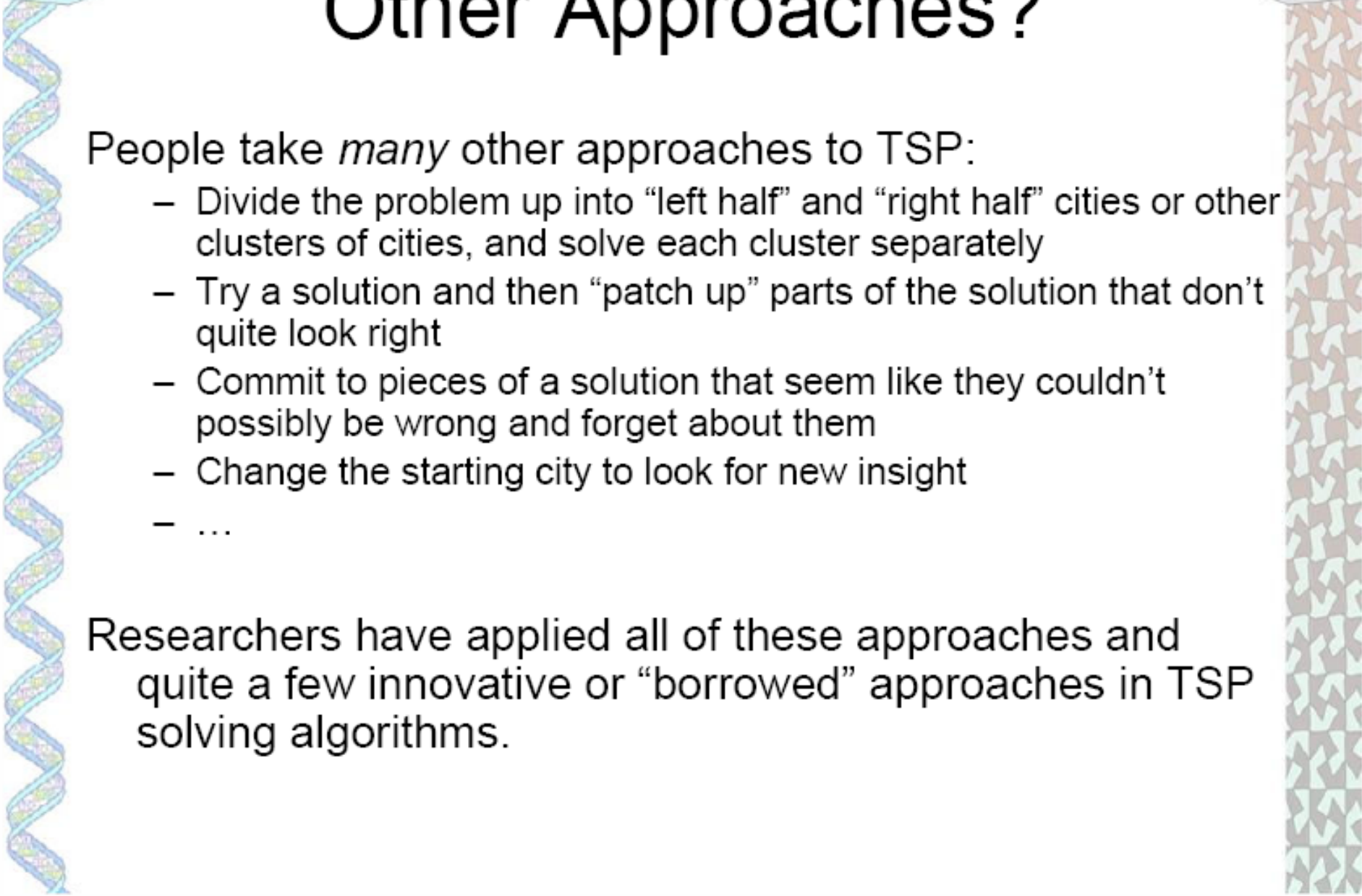


Other Approaches?

People take *many* other approaches to TSP:

- Divide the problem up into “left half” and “right half” cities or other clusters of cities, and solve each cluster separately
- Try a solution and then “patch up” parts of the solution that don’t quite look right
- Commit to pieces of a solution that seem like they couldn’t possibly be wrong and forget about them
- Change the starting city to look for new insight
- ...

Researchers have applied all of these approaches and quite a few innovative or “borrowed” approaches in TSP solving algorithms.



Project 1: Travelling Salesperson Problem - Brute Force

- Learning objectives. At the completion of this project, you should be able to:
 - Implement a method to generate all permutations of a set of numbers.
 - Trace a Hamiltonian path through an undirected graph.
 - Use brute force to find the minimum cost solution to a Traveling Salesperson Problem.

Problem Description

- You will be expected to use brute force to calculate the minimum cost paths for a series of problems.
- Data for each problem will be supplied in a .tsp file (a plain text file).

Hints

- Be careful of exponential explosion. 10 cities will have over 3 million possible paths.
- The largest data set will be 12 cities.
- I strongly urge you to create reusable code. It will be needed for the future projects.
- In the future projects, we will be dealing with MUCH larger datasets. So large that brute force approach would not be possible.

Deliverables

- Project report (2-3 pages). Describe how you generated the permutations representing the tours. Show the route and the cost of the optimal tour for each provided dataset (see template).
- Well-commented source code for your project (You can use any language you like, but I reserve the right to ask you to demo performance of your algorithm on a new dataset).
- You don't have to include a GUI with visual representation of the solutions for this project, but it might be useful for your future TSP related projects in this course.

Data Format

- **Data Format:** You can read about standard TSP problem files here:
<http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>
- Data files for development and testing of your code will be generated using Concorde:
<http://www.math.uwaterloo.ca/tsp/concorde/index.html>

Sample data file with 7 cities:

NAME: concorde7
TYPE: TSP
COMMENT: Generated by CCutil_writetsplib
COMMENT: Write called for by Concorde GUI
DIMENSION: 7
EDGE_WEIGHT_TYPE: EUC_2D
NODE_COORD_SECTION
1 87.951292 2.658162
2 33.466597 66.682943
3 91.778314 53.807184
4 20.526749 47.633290
5 9.006012 81.185339
6 20.032350 2.761925
7 77.181310 31.922361

Distance Formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

The End!

