Artificial Intelligence: History and Methods

Course: CECS545

Artificial Intelligence

University of Louisville

Dr. Roman V. Yampolskiy

AI History (old)

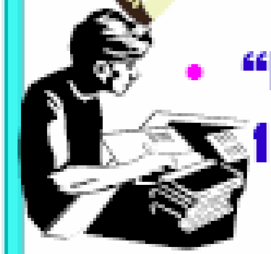
- Philosophy Foundations (400 B.C. – present)
  - Mind: dualism (Descartes), materialism (Leibniz), empiricism (Bacon, Locke)
  - Thought: syllogism (Aristotle), induction (Hume), logical positivism (Russell)
  - Rational agentry (Mill)
- Mathematical Foundations (c. 800 – present)
  - Early: algorithms (al-Khowarazmi, 9th century Arab mathematician), Boolean logic
  - Computability (20th century – present)
    - Cantor diagonalization, Gödel's incompleteness theorem
    - Formal computuational models: Hilbert's Entscheidungsproblem, Turing
    - Intractability and NP-completeness

AI History (not too old)

- Computer Engineering (1940 – present)
- Linguistics (1957 – present)
- Stages of AI

  - Gestation (1943 – c. 1956), infancy (c. 1952 – 1969)

  - Disillusioned early (c. 1966 – 1974), later childhood (1969 – 1979)



  - “Early” (1980 – 1988), “middle” adolescence (c. 1985 – present)

Potted history of AI

1943	McCulloch & Pitts: Boolean circuit model of brain
1950	Turing's "Computing Machinery and Intelligence"
1952–69	Look, Ma, no hands!
1950s	Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
1956	Dartmouth meeting: "Artificial Intelligence" adopted
1965	Robinson's complete algorithm for logical reasoning
1966–74	AI discovers computational complexity Neural network research almost disappears
1969–79	Early development of knowledge-based systems
1980–88	Expert systems industry booms
1988–93	Expert systems industry busts: "AI Winter"
1985–95	Neural networks return to popularity
1988–	Resurgence of probability; general increase in technical depth "Nouvelle AI": ALife, GAs, soft computing
1995–	Agents, agents, everywhere . . .
2003–	Human-level AI back on the agenda

The history of artificial intelligence (birth)

The birth of artificial intelligence (1943 – 1956)

The first work recognised in the field of AI was presented by **Warren McCulloch** and **Walter Pitts** in 1943. They proposed a model of an artificial neural network and demonstrated that simple network structures could learn.

McCulloch, the second "founding father" of AI after Alan Turing, had created the corner stone of neural computing and artificial neural networks (ANN).

- The third founder of AI was **John von Neumann**, the brilliant Hungarian-born mathematician. In 1930, he joined the Princeton University, lecturing in mathematical physics. He was an adviser for the Electronic Numerical Integrator and Calculator project at the University of Pennsylvania and helped to design the **Electronic Discrete Variable Calculator**. He was influenced by McCulloch and Pitts's neural network model. When **Marvin Minsky** and **Dean Edmonds**, two graduate students in the Princeton mathematics department, built the first neural network computer in 1951, von Neumann encouraged and supported them.

- Another of the first generation researchers was **Claude Shannon**. He graduated from MIT and joined Bell Telephone Laboratories in 1941. Shannon shared Alan Turing's ideas on the possibility of machine intelligence. In 1950, he published a paper on chess-playing machines, which pointed out that a typical chess game involved about 10^{120} possible moves (Shannon, 1950). Even if the new von Neumann-type computer could examine one move per microsecond, it would take 3×10^{106} years to make its first move. Thus Shannon demonstrated the need to use heuristics in the search for the solution.

- In 1956, **John McCarthy**, **Marvin Minsky** and **Claude Shannon** organised a summer workshop at Dartmouth College. They brought together researchers interested in the study of machine intelligence, artificial neural nets and automata theory. Although there were just ten researchers, this workshop gave birth to a new science called ***artificial intelligence***.

The rise of artificial intelligence, or the era of great expectations (1956 – late 1960s)

- The early works on neural computing and artificial neural networks started by McCulloch and Pitts was continued. Learning methods were improved and **Frank Rosenblatt** proved the ***perceptron convergence theorem***, demonstrating that his learning algorithm could adjust the connection strengths of a perceptron.

- One of the most ambitious projects of the era of great expectations was the **General Problem Solver (GPS)**. **Allen Newell** and **Herbert Simon** from the Carnegie Mellon University developed a general-purpose program to simulate human-solving methods.
- Newell and Simon postulated that a problem to be solved could be defined in terms of ***states***. They used the mean-end analysis to determine a difference between the current and desirable or *goal state* of the problem, and to choose and apply *operators* to reach the goal state. The set of operators determined the solution plan.

- However, GPS failed to solve complex problems. The program was based on formal logic and could generate an infinite number of possible operators. The amount of computer time and memory that GPS required to solve real-world problems led to the project being abandoned.
- In the sixties, AI researchers attempted to simulate the thinking process by inventing *general methods* for solving *broad classes of problems*. They used the general-purpose search mechanism to find a solution to the problem. Such approaches, now referred to as ***weak methods***, applied weak information about the problem domain.

- By 1970, the euphoria about AI was gone, and most government funding for AI projects was cancelled. AI was still a relatively new field, academic in nature, with few practical applications apart from playing games. So, to the outsider, the achieved results would be seen as toys, as no AI system at that time could manage real-world problems.

Unfulfilled promises, or the impact of reality (late 1960s – early 1970s)

The main difficulties for AI in the late 1960s were:

- Because AI researchers were developing general methods for broad classes of problems, early programs contained little or even no knowledge about a problem domain. To solve problems, programs applied a search strategy by trying out different combinations of small steps, until the right one was found. This approach was quite feasible for simple **toy problems**, so it seemed reasonable that, if the programs could be "scaled up" to solve large problems, they would finally succeed.

Many of the problems that AI attempted to solve were **too broad and too difficult**. A typical task for early AI was machine translation. For example, the National Research Council, USA, funded the translation of Russian scientific papers after the launch of the first artificial satellite (Sputnik) in 1957.

- **The spirit is willing but the flesh is weak**
- **The vodka is good but the meat is rotten**

- In 1971, the British government also suspended support for AI research. Sir **James Lighthill** had been commissioned by the Science Research Council of Great Britain to review the current state of AI. He did not find any major or even significant results from AI research, and therefore saw no need to have a separate science called “artificial intelligence”.

Why Study Artificial Intelligence?

- New Computational Capabilities
  - Advances in uncertain reasoning, knowledge representations
  - Learning to act: robot planning, control optimization, decision support
  - Database mining: converting (technical) records into knowledge
  - Self-customizing programs: learning news filters, adaptive monitors
  - Applications that are hard to program: automated driving, speech recognition

Why Study Artificial Intelligence?

- **Better Understanding of Human Cognition**
  - Cognitive science: theories of knowledge acquisition (e.g., through practice)
  - Performance elements: reasoning (inference) and *recommender* systems
- **Time is Right**
  - Recent progress in algorithms and theory
  - Rapidly growing volume of online data from various sources
  - Available computational power
  - Growth and interest of AI-based industries (e.g., data mining/KDD, planning)

Some Hard Questions…

- Who is liable if a robot driver has an accident?
- Will machines surpass human intelligence?
- What will we do with superintelligent machines?
- Would such machines have conscious existence? Rights?
- Can human minds exist indefinitely within machines (in principle)?

Questions?

Artificial Intelligence: Traveling Salesperson Problem, Project 1

Course: CECS545

Artificial Intelligence

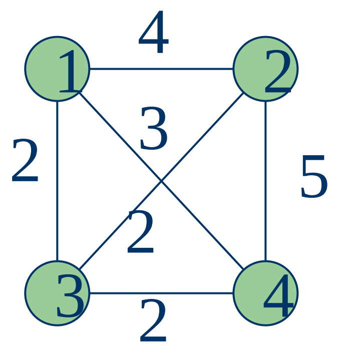
University of Louisville

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)

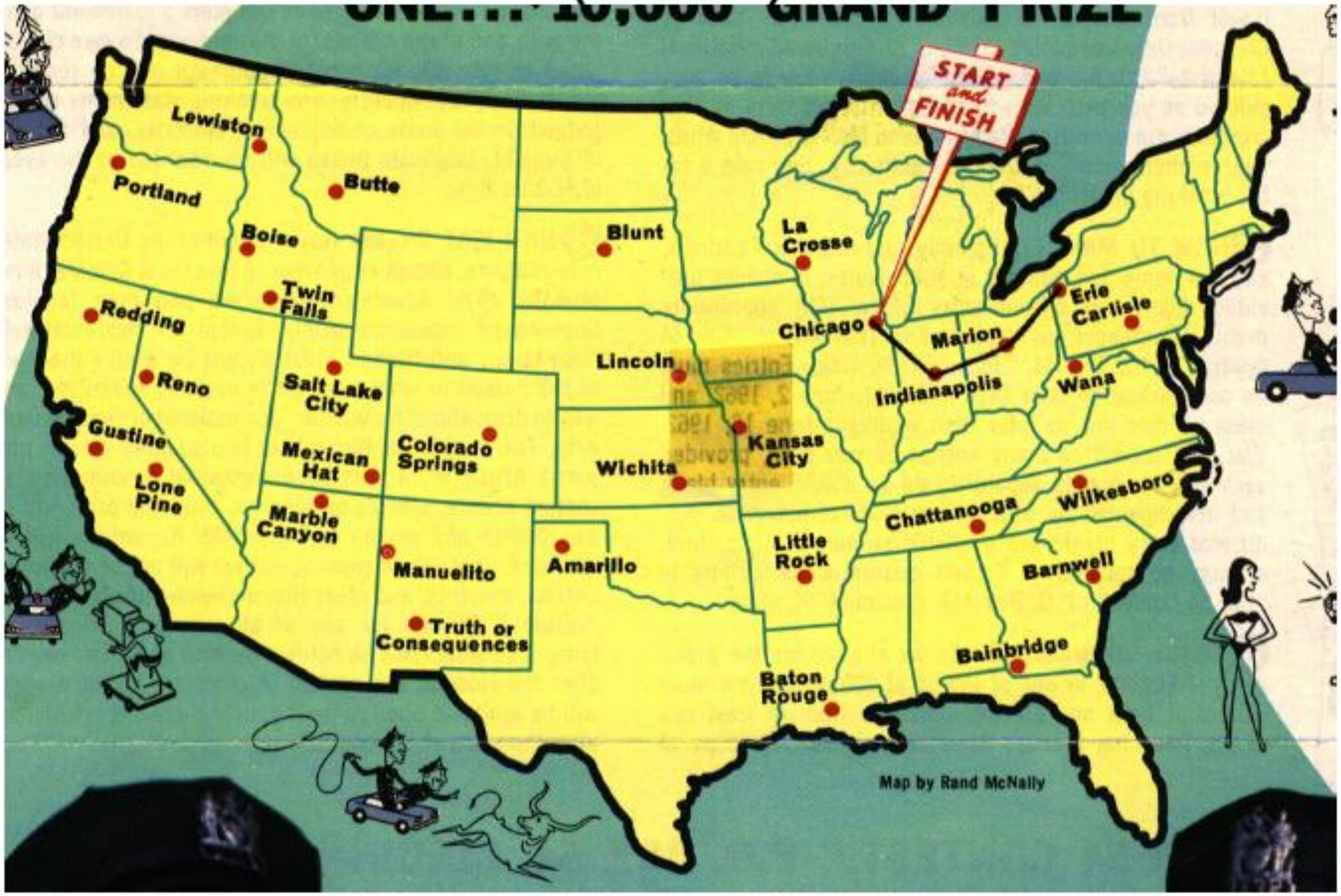
- *Question*: Find shortest (simple) cycle that visits every city

Learning Goals for Today

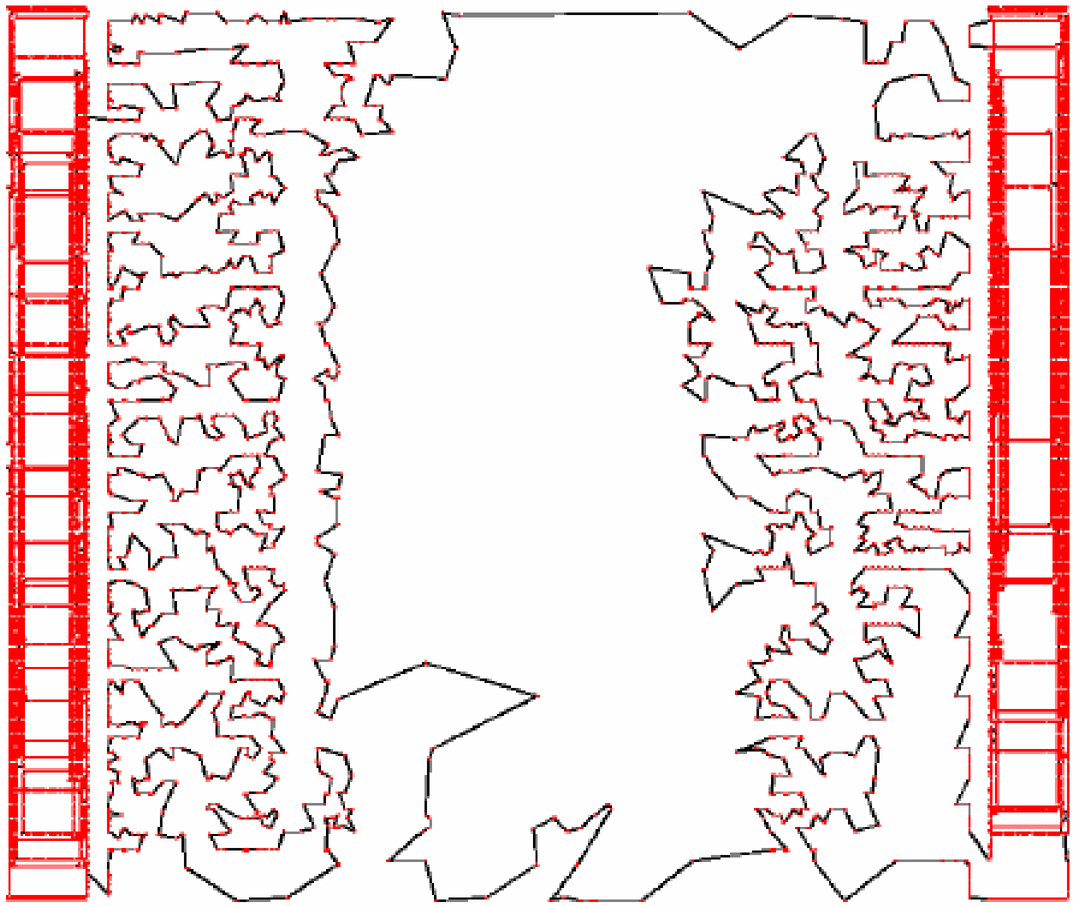
After today's unit, you will be able to:

- 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.

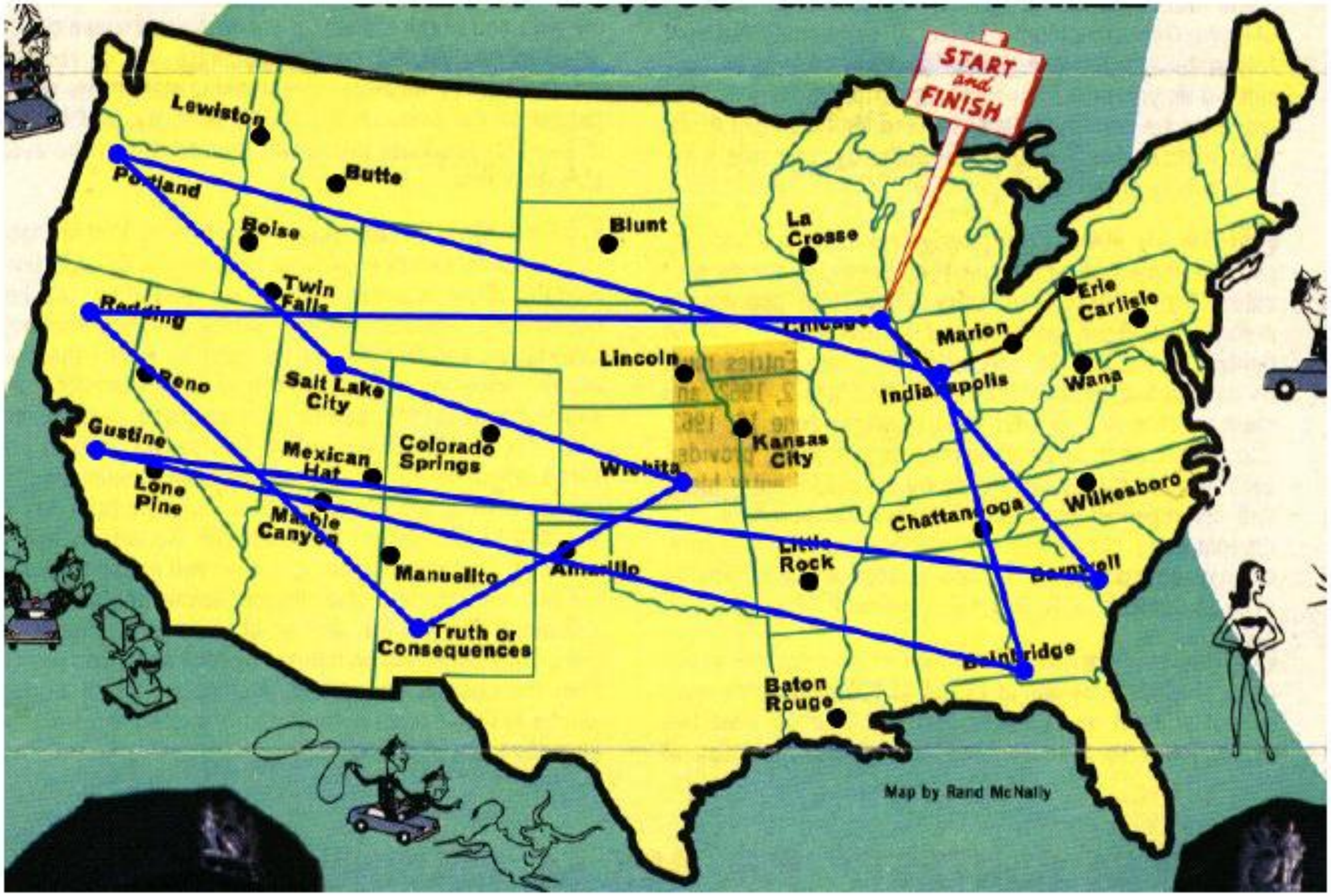
This works even when the costs aren’t just distance, as with airline ticket prices, which may be cheaper in one direction than another or cheaper for a longer flight than a shorter one.

The “Random” Approach

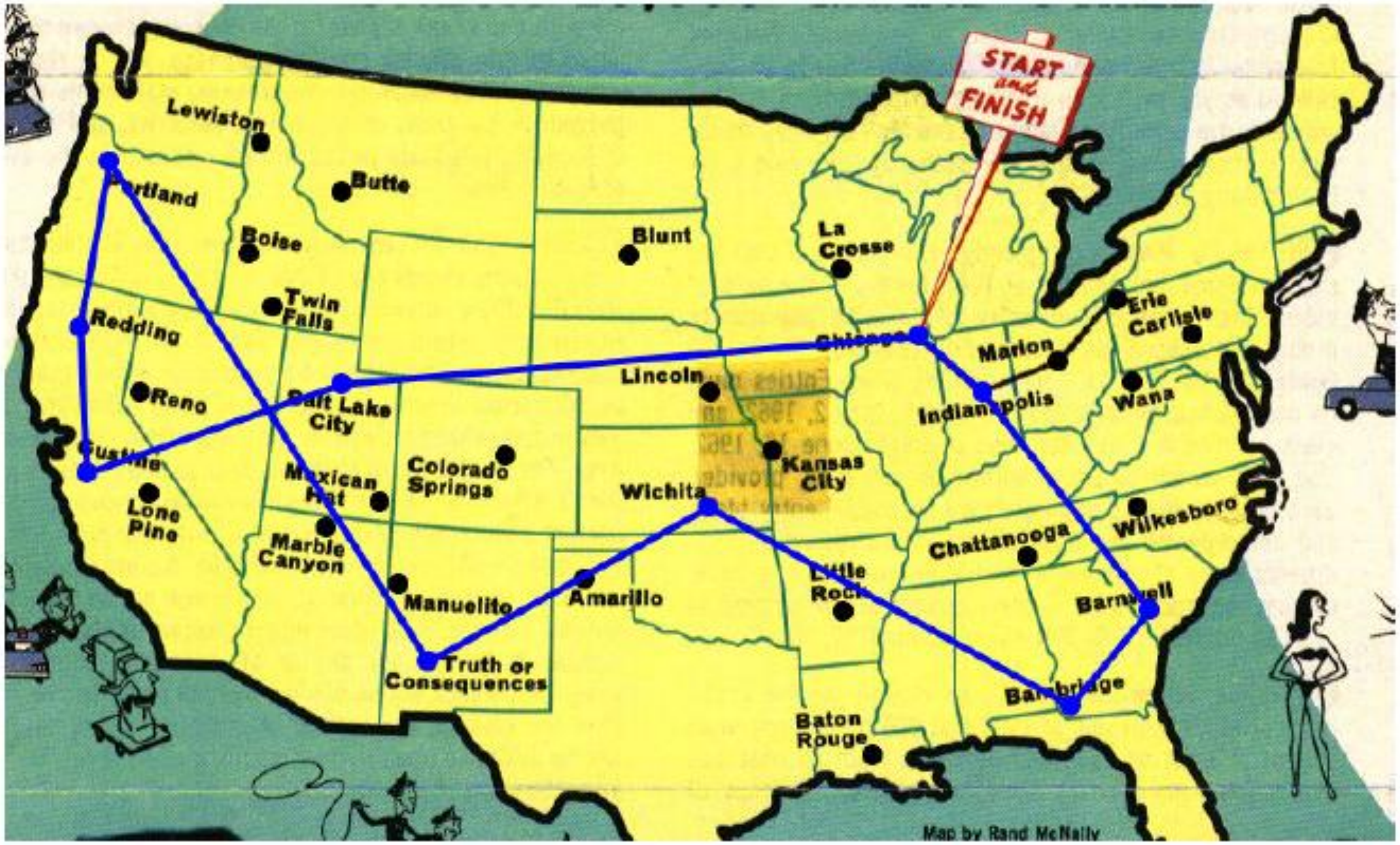
Algorithm:

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?

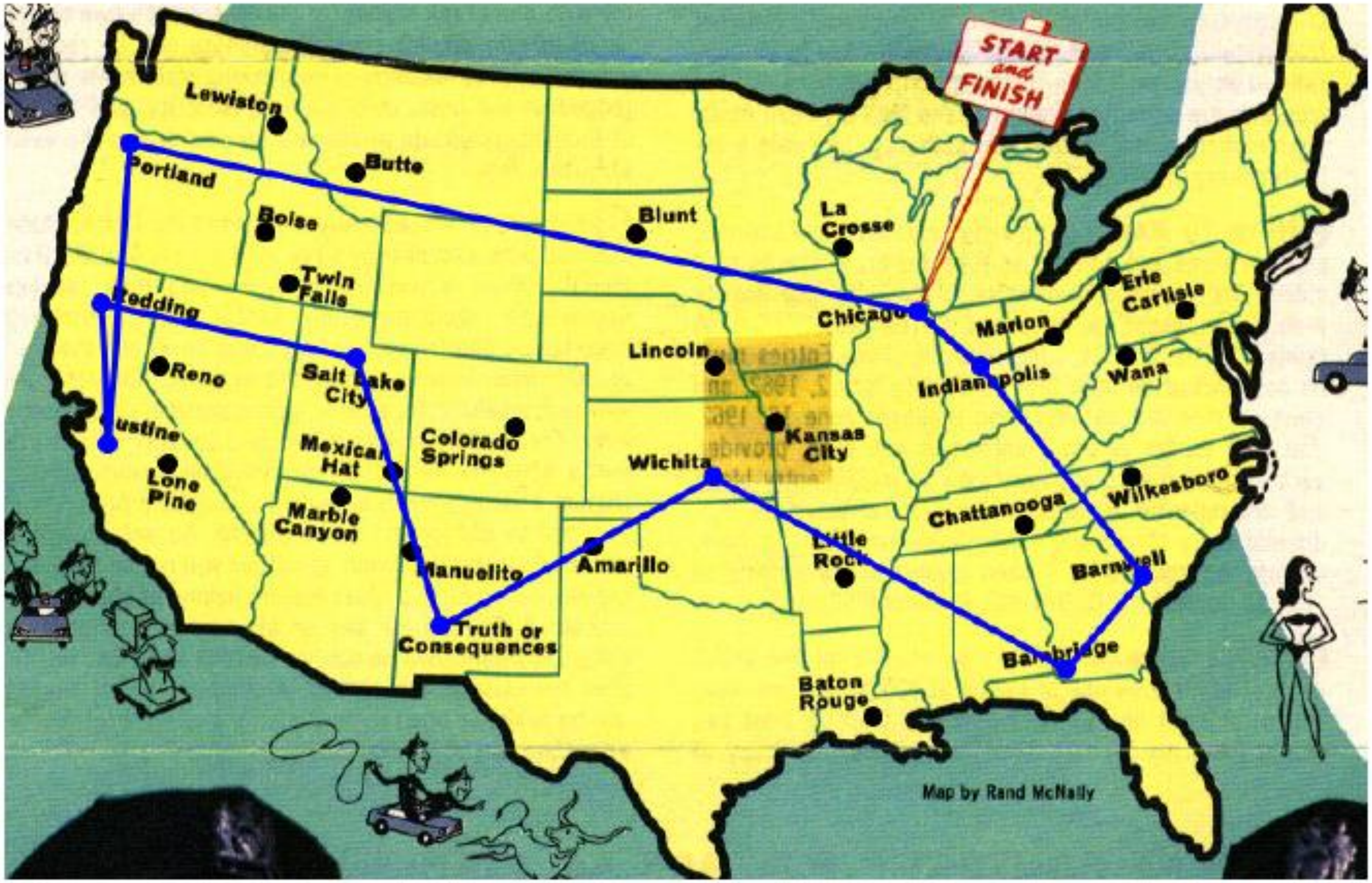
The “Greedy” Approach

Algorithm:

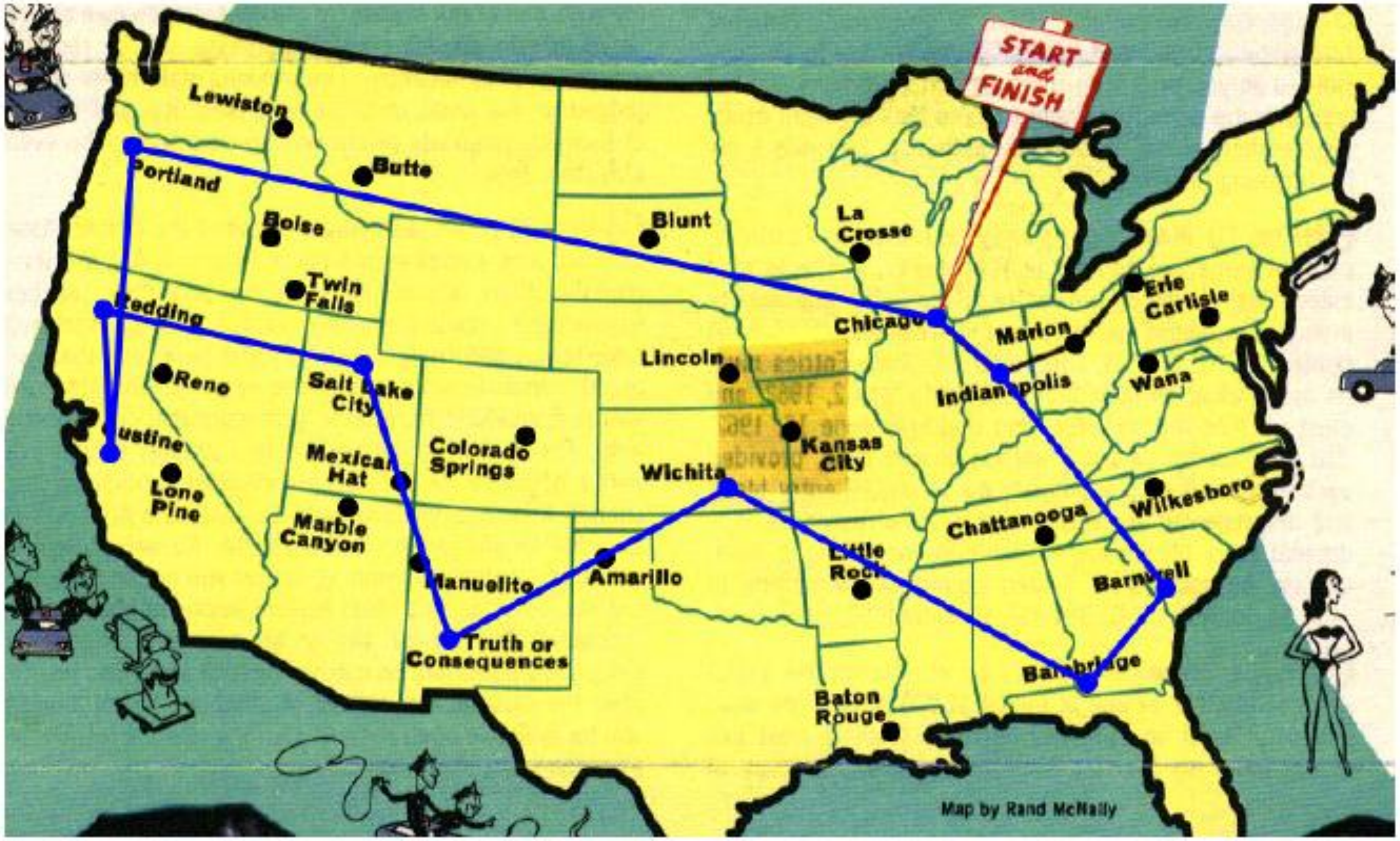
1. Visit (start at) the start city.
2. As long as some city has not been visited:
   a) Look through all unvisited cities and pick the closest.
   b) Visit that closest city.
3. Return to the start city.

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?

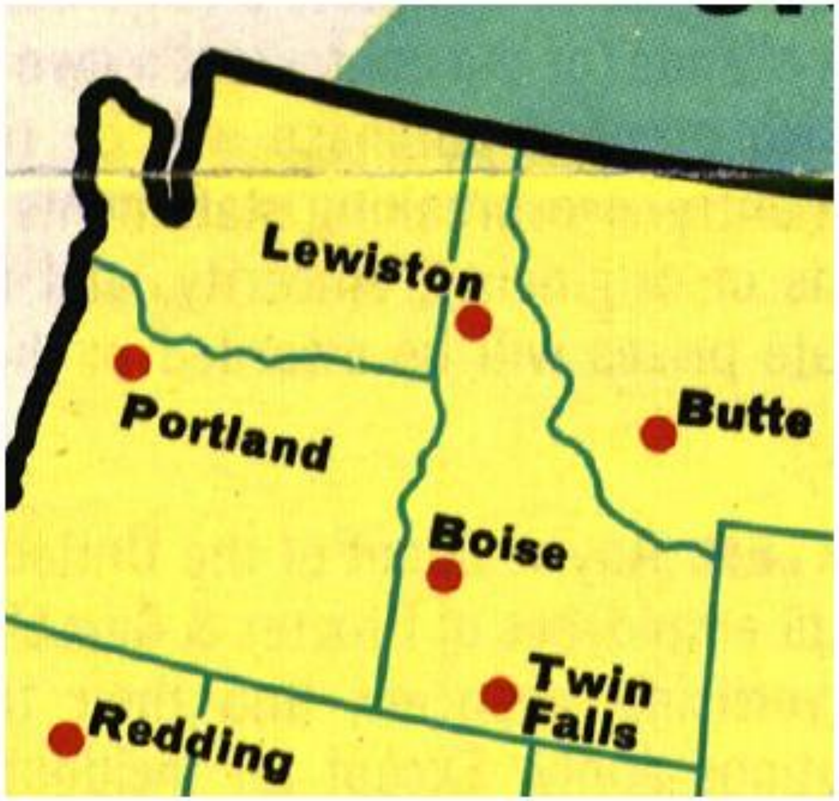
Look just at the Northwest.

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 goes 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://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95

- Data files for development and testing of your code will be generated using Concorde: http://www.tsp.gatech.edu/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!