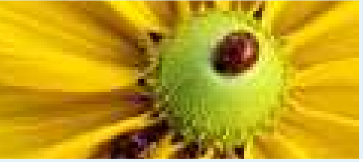
Introduction to Heuristic Search

Jordan Thayer

jthayer@draper.com

Logistics



Course Schedule

- Terminology, TSP, Depth First Search — October 7th
- Heuristics, Tree, Local, Multicore Search — October 14th
- 15 Puzzle, Heuristic Construction, Best First Search — October 21st
- Tile Puzzles, Bounded Suboptimality, Anytime Search October 28th
- Grid Navigation, Inadmissible Heuristics Learning in Search — November 4th
- Search on Disk for Big Problems — November 11th

Course Schedule — Revised

■ TSP, Heuristics, Basic Algorithms Tree Search, Local Search	October 7th
■ 15 Puzzle, Heuristic Construction, Best First Search	October 14th
■ Multiple Cores, Multiple Heuristics Advanced Search Techniques	October 21st
■ Traveling for STAC Program	October 28th
■ Traveling for STAC Program	November 4th
■ Home for Birth of Niece	November 11th

If there's interest in making up the other material, we can see about working something out.

Course Materials

- TSP Ingestion Code, Visualizer & Reference Solutions
  - Tiles Ingestion Code, Visualizer

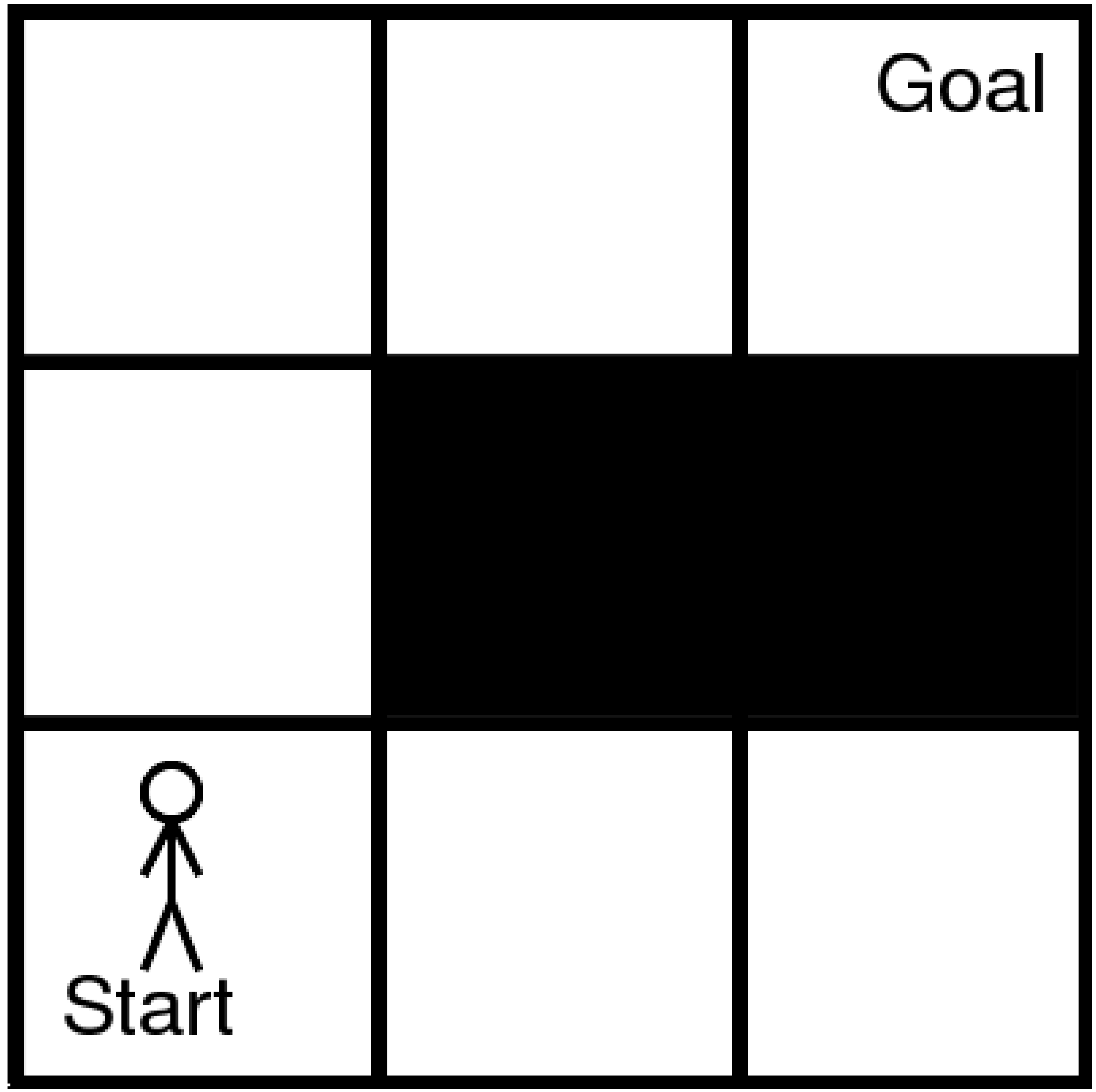
  https://github.com/jordanthayer/tsp-demo
- C++ Search Library https://github.com/jordanthayer/search
  - C++ translation of Ocaml, done by Ethan Burns
- Search Visualizations
  - My Youtube Channel
  - Still Images from my home page
- These Slides will be somewhere soon as well.

Useful Texts

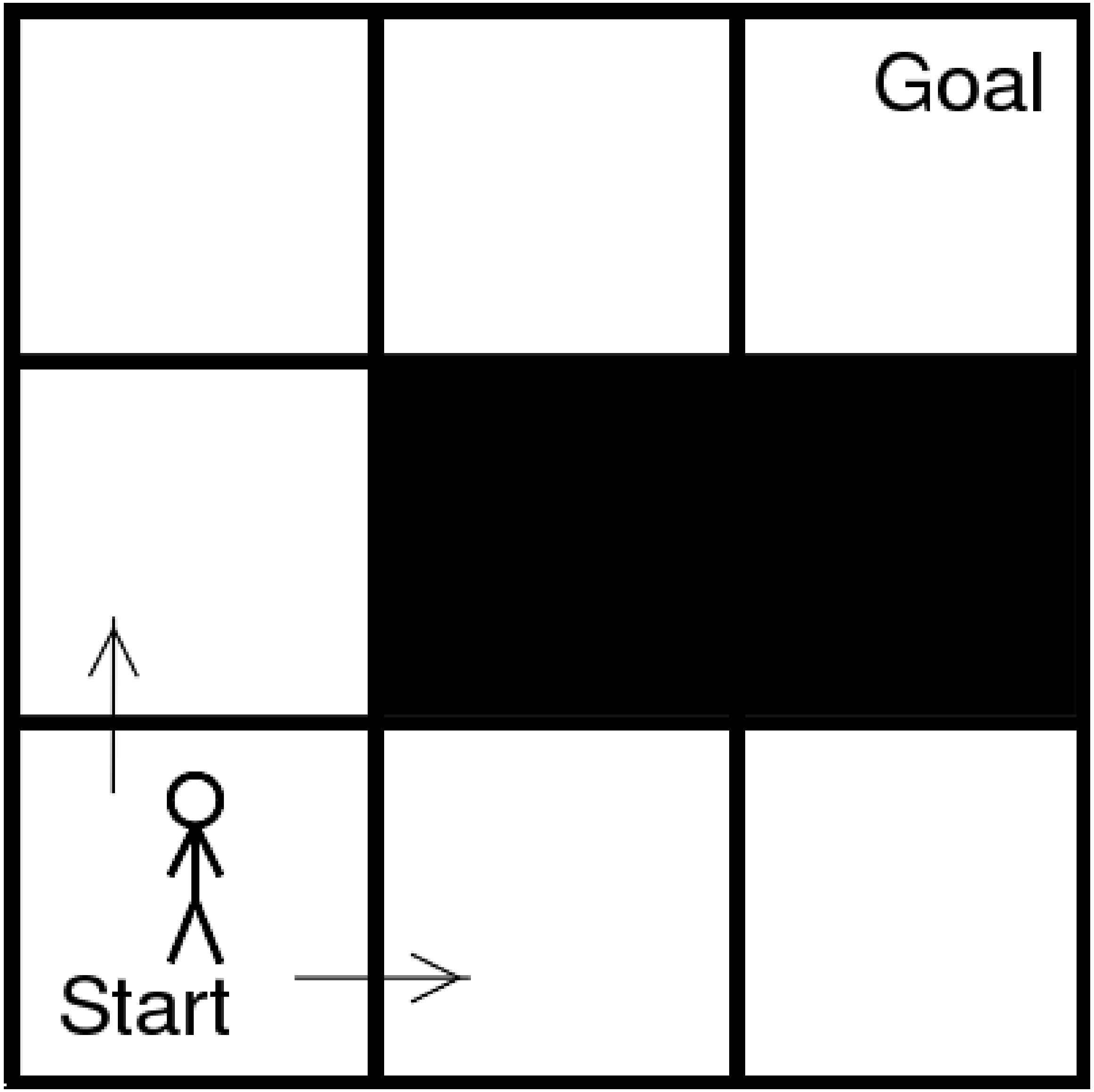
- Artificial Intelligence: A Modern Approach
  Especially Chapters TODO
- Heuristics: Intelligent Search Strategies for Computer Problem Solving
- Do the Right Thing

Heuristic Search

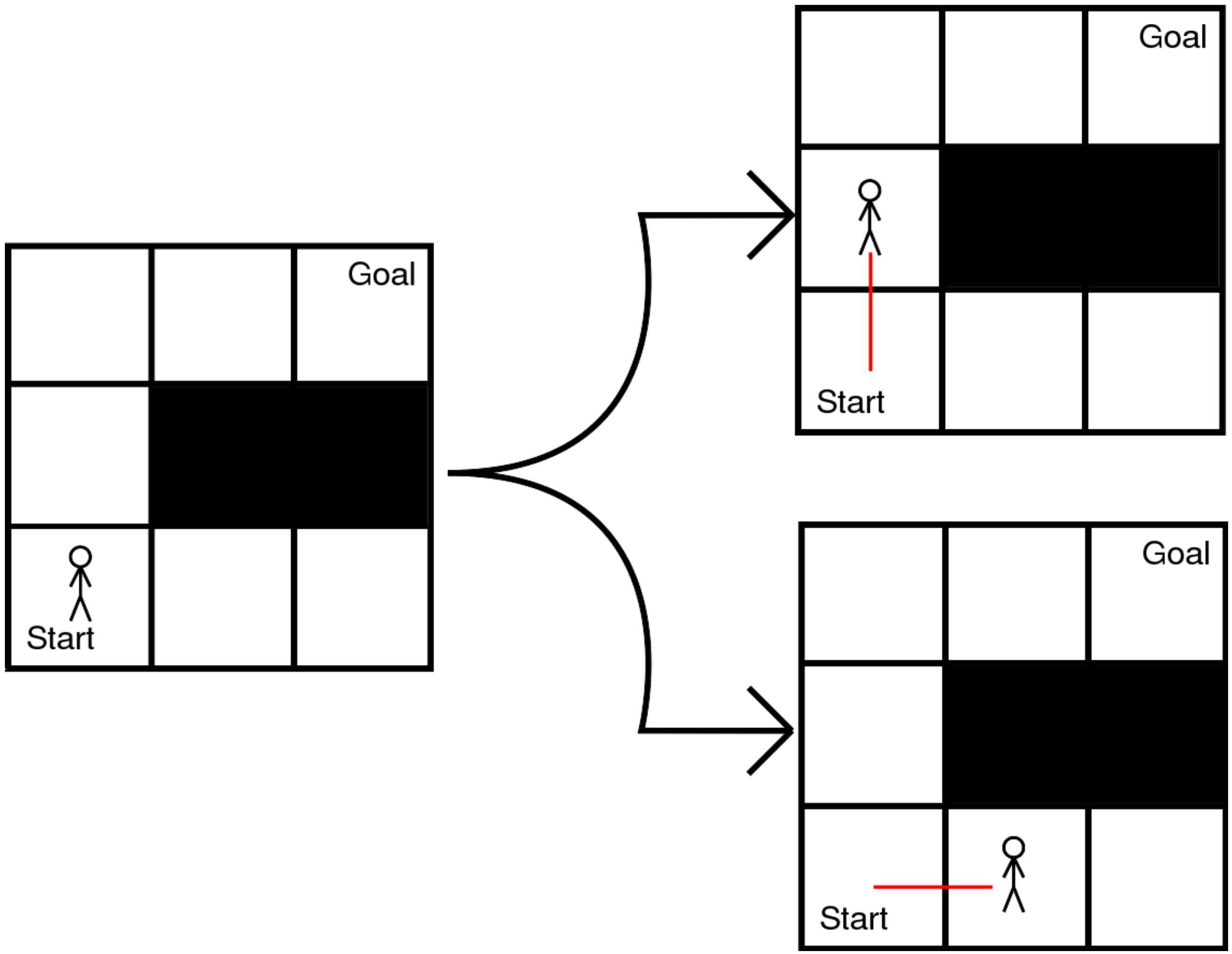
A Simple Problem



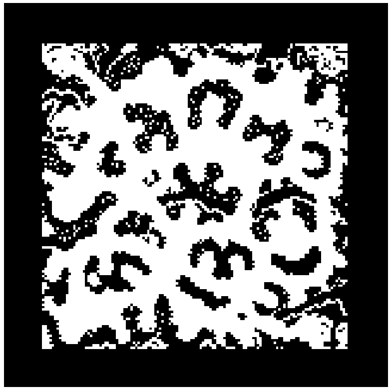
Where can I get to from here?

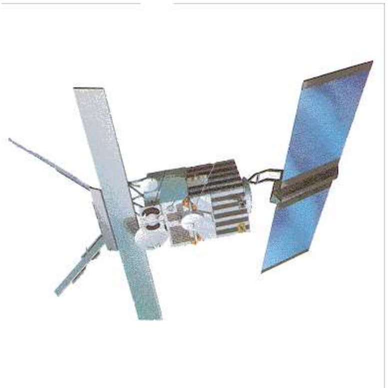


Where can I get to from here?



Other Heuristic Search Problems





- A starting configuration
- Primitive operations to move between configurations
- A goal test

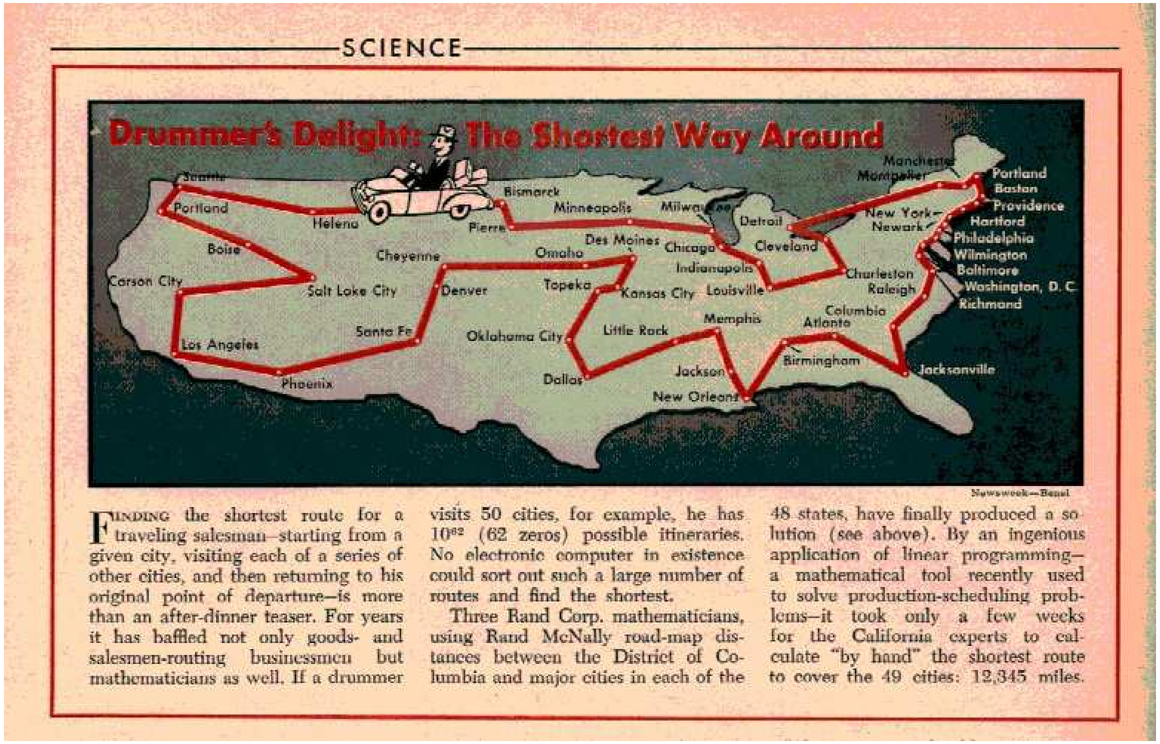
TSP

Problem Definition

A Salesman wants to find the shortest tour of a fixed set of cities, starting with the city they live in, and returning there once again at the end of the trip.

Problem Definition

A Salesman wants to find the shortest tour of a fixed set of cities, starting with the city they live in, and returning there once again at the end of the trip.



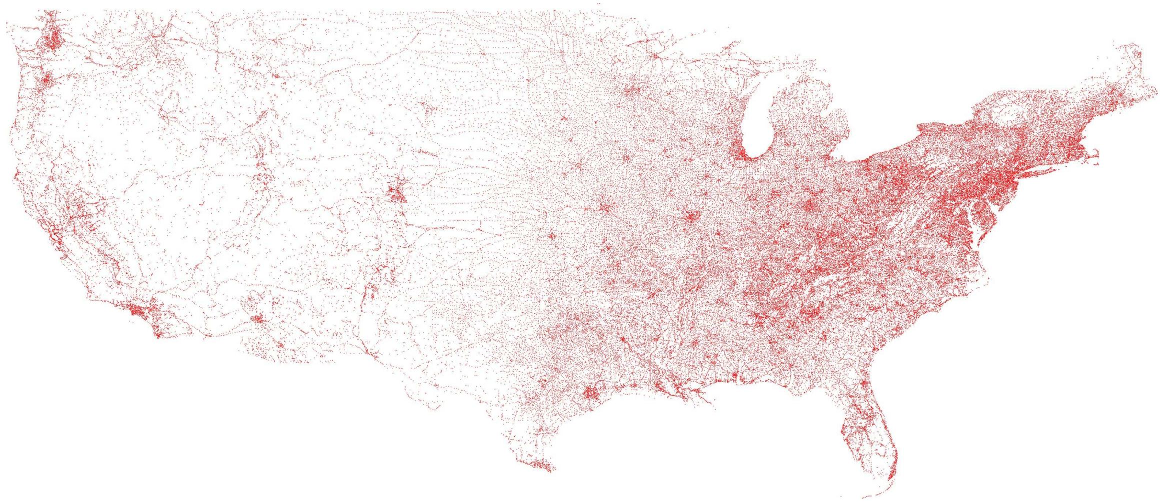
SCIENCE

FINDING the shortest route for a traveling salesman—starting from a given city, visiting each of a series of other cities, and then returning to his original point of departure—is more than an after-dinner teaser. For years it has baffled not only goods- and salesmen-routing businessmen but mathematicians as well. If a drummer visits 50 cities, for example, he has 10^{62} (62 zeros) possible itineraries. No electronic computer in existence could sort out such a large number of routes and find the shortest.

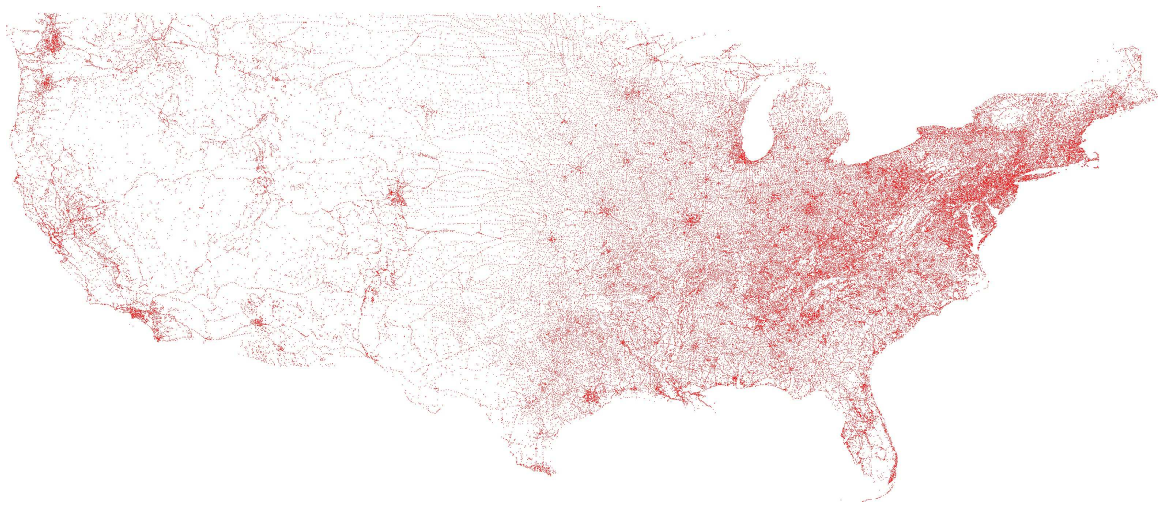
Three Rand Corp. mathematicians, using Rand McNally road-map distances between the District of Columbia and major cities in each of the 48 states, have finally produced a solution (see above). By an ingenious application of linear programming—a mathematical tool recently used to solve production-scheduling problems—it took only a few weeks for the California experts to calculate "by hand" the shortest route to cover the 49 cities: 12,345 miles.

Problem Definition

A Salesman wants to find the shortest tour of a fixed set of cities, starting with the city they live in, and returning there once again at the end of the trip.

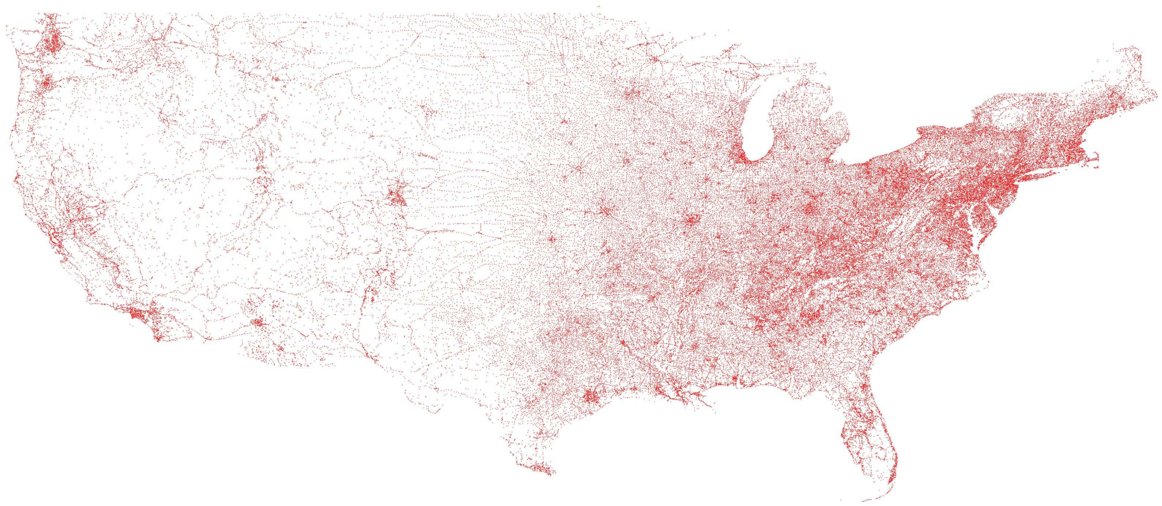


Size of the TSP Problem



This instance has 115475 cities.

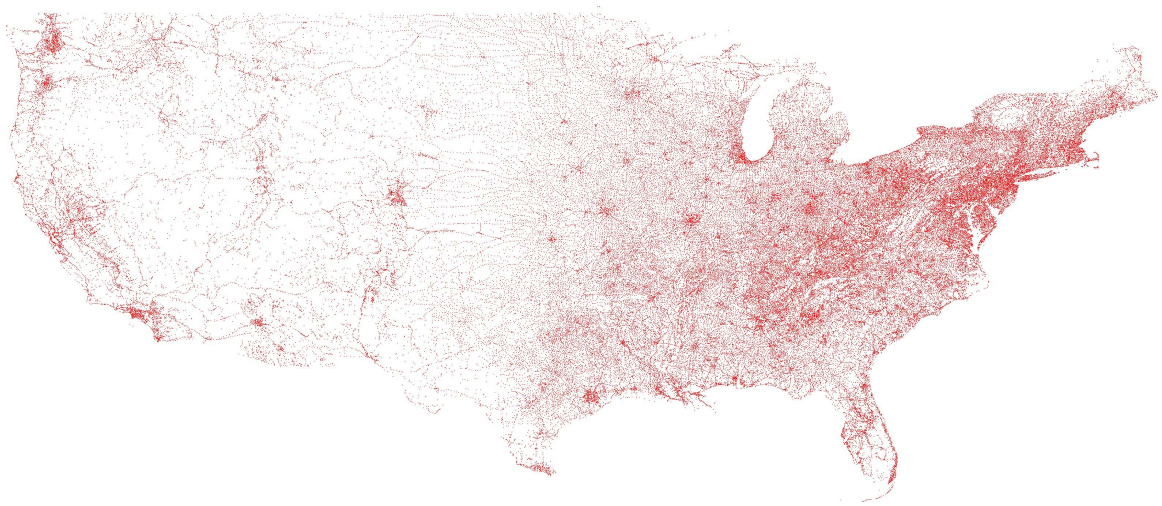
Size of the TSP Problem



This instance has 115475 cities.

There are, roughly, $\frac{115475!}{2}$ unique tours.

Size of the TSP Problem

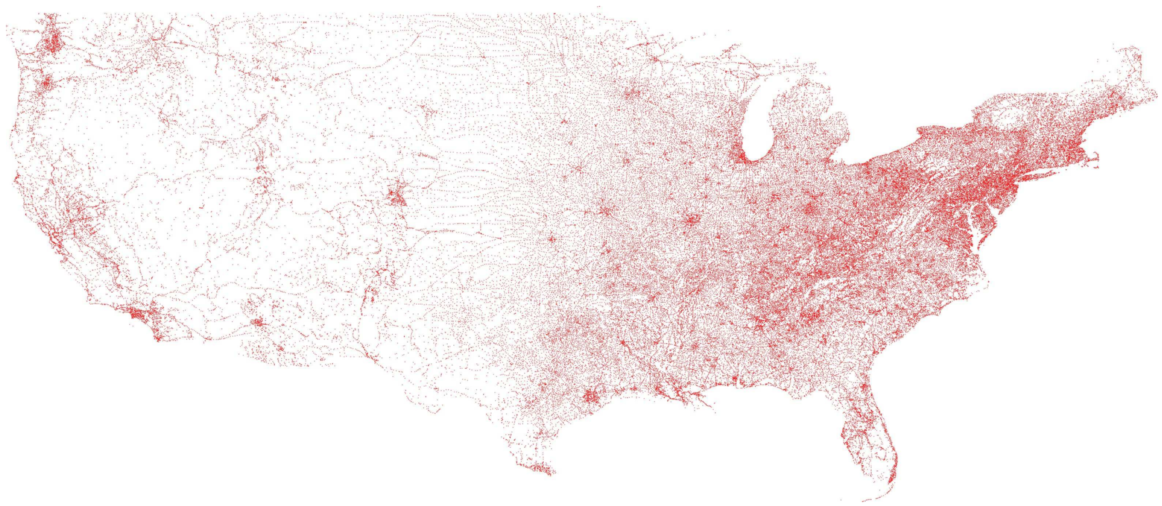


This instance has 115475 cities.

There are, roughly, $\frac{115475!}{2}$ unique tours.

That's about $1.9 \cdot 10^{534443}$ tours.
For reference, Eddignton's number is about $1.6 \cdot 10^{80}$.

Size of the TSP Problem



This instance has 115475 cities.

There are, roughly, $\frac{115475!}{2}$ unique tours.

That's about $1.9 \cdot 10^{534443}$ tours.
For reference, Eddignton's number is about $1.6 \cdot 10^{80}$.
So how do we start solving this?

Depth First

1. For a given city at the head of a partial tour
2. If this city completes the tour
   Record the new solution
3. Otherwise, for each neighboring city
   consider it as the head of a new partial tour
   recurse

```
dfs(Node n):
  if goal(n):
    updateIncumbent(n)
  else:
    for s in successors(n):
      dfs(s)
```

```
incumbent = None
updateIncumbent(Node n):
  if incumbent == None or n.cost < incumbent.cost:
    incumbent = n
```

```
successors(Node n):
  remaining = n.remaining
  city = n.city
  cost = node.cost
  accum = []
  for ind from 0 to length(remaining) - 1:
    remainingPrime = copy(remaining)
    cityPrime = remainingPrime[ind]
    costPrime = cost + distance(city, cityPrime)
    del remainingPrime[ind]
    accum.extend(Node(cityPrime,
                      remainingPrime,
                      costPrime))
  return accum
```

Go to the java program, show a couple instances.

```
dfs(Node n):
  if goal(n):
    updateIncumbent(n)
  else if better(n, incumbent):
    for s in successors(n):
      dfs(s)
```

```
better(Node n, Node inc):
  inc == None or n.cost < inc.cost
```

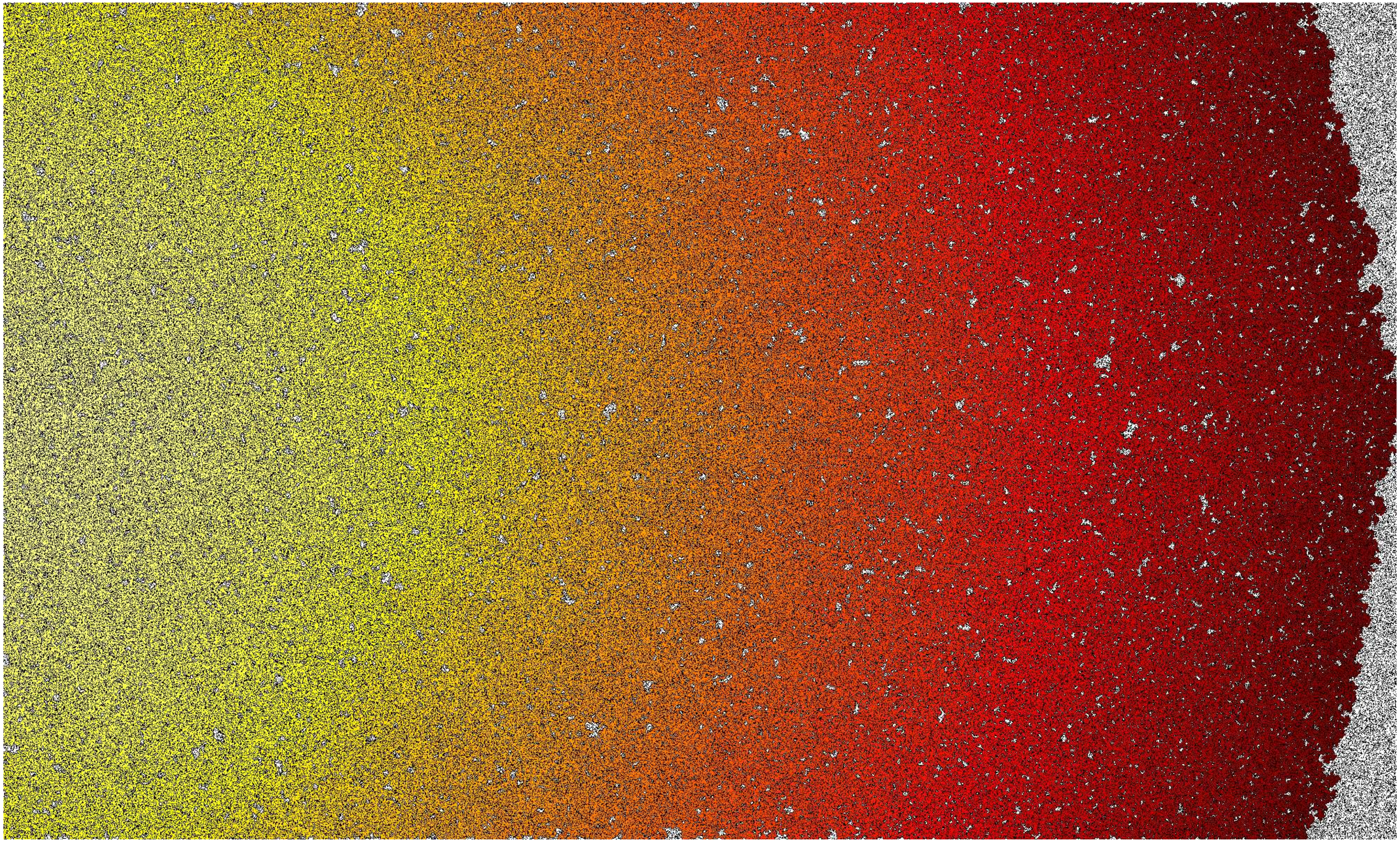
This is still complete and converges on optimal!

```
dfs(Node n):
  if goal(n):
    updateIncumbent(n)
  else:
    for delta in successorDeltas(n):
      applyDelta(delta,n)
      dfs(n)
      undoDelta(n)
```

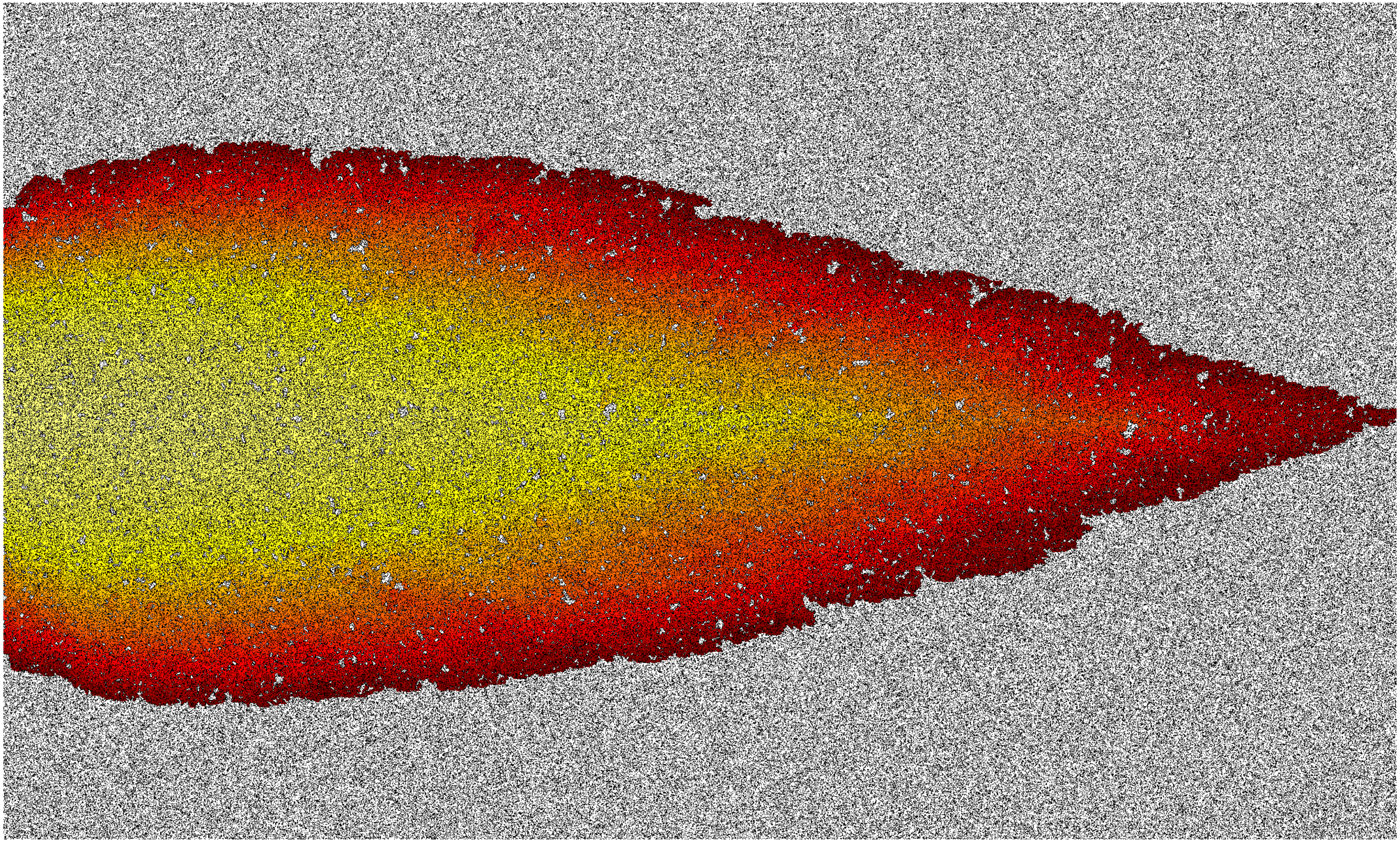
```
successors(Node n):
  remaining = n.remaining
  city = n.city
  cost = node.cost
  accum = []
  for ind from 0 to length(remaining) - 1:
    remainingPrime = copy(remaining) ## Expensive!
    cityPrime = remainingPrime[ind]
    costPrime = cost + distance(city, cityPrime)
    del remainingPrime[ind]
    accum.extend(Node(cityPrime,
                      remainingPrime,
                      costPrime))
  return accum
```

By working with a single logical state, and making and undoing changes, we can touch less memory and generally go faster.

Heuristics



Why Heuristics?



Where Do Heuristics Come From?

- Solutions to a Relaxed Problem
- Solutions to an Abstract Problem
- General Rules of Thumb from Domain Experts
- Machine Learning

Estimates of Cost

A very useful thing to estimate is the cost of completing a solution.

Estimates of Cost

A very useful thing to estimate is the cost of completing a solution. In general, $f(n) = g(n) + h(n)$ where
$f(n)$ is the total estimated cost
$g(n)$ is the cost of the partial solution
$h(n)$ is some estimate of the cost of completing the solution.

Estimates of Cost

A very useful thing to estimate is the cost of completing a solution. In general, $f(n) = g(n) + h(n)$ where
$f(n)$ is the total estimated cost
$g(n)$ is the cost of the partial solution
$h(n)$ is some estimate of the cost of completing the solution.
What is $g(n)$ for the TSP?
What are some potential $h(n)$s?

Estimates of Cost

A very useful thing to estimate is the cost of completing a solution. In general, $f(n) = g(n) + h(n)$ where
$f(n)$ is the total estimated cost
$g(n)$ is the cost of the partial solution
$h(n)$ is some estimate of the cost of completing the solution.
What is $g(n)$ for the TSP?
What are some potential $h(n)$s?

- The distance to the nearest city.
- Of all remaining cities, the furthest distance in latitude and the furthest distance in longitude from either the tour head or the starting city.
- The minimum spanning tree of the remaining cities

Minimum Spanning Tree as an admissible estimate of cost to go.

Preference Heuristics

A preference heuristic simply tells us, among two solutions or partial solutions, or in general, two search alternatives, which should the algorithm consider first.

Preference Heuristics

A preference heuristic simply tells us, among two solutions or partial solutions, or in general, two search alternatives, which should the algorithm consider first.

- Given two partial tours, which one do I prefer?
- Given a current partial tour, how should I extend it?

Preference Heuristics

A preference heuristic simply tells us, among two solutions or partial solutions, or in general, two search alternatives, which should the algorithm consider first.

- Given two partial tours, which one do I prefer?
- Given a current partial tour, how should I extend it?

The nearest neighbor is a very natural intuition, and it works pretty well in practice.

Using Heuristics in DFS


```
dfs(Node n):
  if goal(n):
    updateIncumbent(n)
  else if feasible(n, incumbent):
    next = successors(n)
    sort(next, nearest(n.city)) # Visit Children in
    for s in next:              # heuristic order.
      dfs(s)


feasible(Node n, Node inc):
 if inc == None:
   return True
 else:
   return inc.cost > n.cost + mst(n) # g(inc) > f(n)
```

Another Demo!

Show the child ordering DFS implementation against the non-child ordering implementation.

Local Search

Local Search in General

```
localSearch(currentSolution):
  updateIncumbent(currentSolution)
  if outOfTime:
    return currentSolution
  else:
    possible = neigbors(currentSolution)
    next = select(possible, currentSolution)
    localSearch(next)
```

Random Walk

```
randomWalk(currentSolution):
  updateIncumbent(currentSolution)
  if outOfTime:
    return currentSolution
  else:
    possible = neigbors(currentSolution)
    next = randomElt(possible)
    randomWalk(next)
```

Hill Climbing


```
hillClimb(currentSolution):
  updateIncumbent(currentSolution)
  if outOfTime:
    return currentSolution
  else:
    possible = neigbors(currentSolution)
    next = None
    for p in possible:
      if p.cost < currentSolution.cost:
        next = p
        current = p
    if next:
      hillClimb(next)
    else:
      return hillClimb(randomSolution())
```

Completeness Proofs

Wrap-Up

Relationship Between Techniques

Properties of Problem Beneficial to These Techniques

- Solutions Exist at a Fixed Depth
- Feasible Solutions Are Easy To Find
- Moving Between Solutions is Easy
- Good Selection of Heuristics

Similar Relevant Things I Won't Talk About

Next Week

- Graphs Vs. Trees
- Best First Search
- Heuristic Construction
- Dealing With Non-Assignment Problems