Algorithms: Why and How

Dr. Ted Ralphs

LEHIGH
UNIVERSITY

What is an Algorithm?

What is an Algorithm?

- According to Webster's Collegiate Dictionary:

  "A procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation; broadly: a step-by-step procedure for solving a problem or accomplishing some end especially by a computer"

- More concisely, an *algorithm* is a procedure for converting *input* to *output*.
- In this way, algorithm is similar to words such as recipe, procedure, technique, method, process, routine, etc.
- However, algorithms usually have the following special properties:
  - Each step is precisely defined.
  - They are used to solve mathematical *problems*.
  - They are *finite*.
- In modern use, an algorithm is usually a procedure implemented on a computer.

An Interesting Quote

"Here is your book, the one your thousands of letter have asked us to publish. It has taken us years to do, checking and rechecking countless recipes to bring you only the best, only the interesting, only the perfect. Now we can say, without a shadow of a doubt, that every single one of them, if you follow the directions to the letter, will work for you exactly as it did for us, even if you have never cooked before."

–McCall's Cookbook (1963)

A Brief History of Algorithms

- According to the Oxford English Dictionary, the word algorithm is a combination of the Middle English word *algorism* with *arithmetic*.
- This word probably did not enter common usage in the English language until sometime last century.
- The word *algorism* derives from the name of an Arabic mathematician circa A.D. 825, whose surname was Al-Khwarizmi.
- Al-Khwarizmi wrote a book on solving equations from whose title the word *algebra* derives.
- It is commonly believed that the first algorithm was Euclid's Algorithm for finding the greatest common divisor of two integers, m and n ($m \geq n$).
  - Divide m by n and let r be the remainder.
  - If $r = 0$, then $\gcd(m, n) = n$.
  - Otherwise, $\gcd(m, n) = \gcd(n, r)$.

Google Etymology for Algorithm

al·go·rithm
/ˈalgəˌriT͟Həm/

noun
noun: **algorithm**; plural noun: **algorithms**

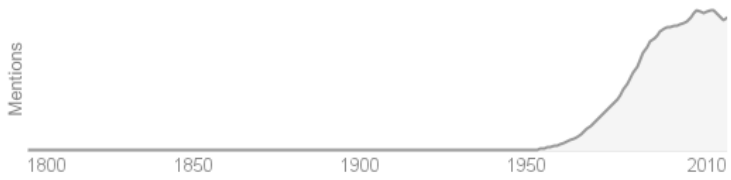
a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
"a basic **algorithm** for division"

Origin

late 17th century: variant (influenced by Greek *arithmos* 'number') of Middle English *algorism*, via Old French from medieval Latin *algorismus* . The Arabic source, *al-Ḵwārizmī* 'the man of Ḵwārizm' (now Khiva), was a name given to the 9th-century mathematician Abū Ja'far Muhammad ibn Mūsa.

Translate algorithm to Choose language

Use over time for: algorithm



Show Me the Algorithms

- Algorithms are literally everywhere you look.
- What are some common applications of algorithms?

- Why is it important that algorithms execute quickly?

Show Me the Algorithms

- Algorithms are literally everywhere you look.
- What are some common applications of algorithms?
  - Your car is run by algorithms.
  - Computers and the Internet are run by algorithms.
  - All your phone calls are routed by algorithms.
  - Spotify, Pandora, and Netflix use algorithms to recommend music/movies to you.
  - Amazon uses algorithms to determine how to get you your stuff quickly.
  - Google uses algorithms to show you the fastest route from place to place or to target advertising.
  - Money managers use algorithms to find good investments.
  - Biologists use algorithms to discover drugs.
  - *You use algorithms to make decisions, such as how to spend your money.*
- Why is it important that algorithms execute quickly?

What is a Problem?

What is a Problem?

- Roughly, a *problem* specifies what set of outputs is desired for each given set of inputs.
- A *problem instance* is just a specific set of inputs.
- Example: The Sorting Problem

  Input: A sequence of n numbers $a_1, a_2, \ldots, a_n$.
  Output: A reordering $a'_1, a'_2, \ldots, a'_n$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.
- *Solving* a problem instance consists of specifying a procedure for converting the inputs to an output of the desired form (called a *solution*).
- An algorithm that is guaranteed to result in a solution for every instance is said to be *correct*.
- Note that a given instance may have either no solutions or more than one solution.

Another Example: Fibonacci Numbers

Thanks to David Eppstein

- Another simple example of the importance of efficient algorithms arises in the calculation of *Fibonacci numbers*.
- Fibonacci numbers arise in population genetics, as well as a host of other applications.
- The n^{th} Fibonacci number is defined recursively as follows:

$$F(1) = 1, \tag{1}$$
$$F(2) = 1, \tag{2}$$
$$F(n) = F(n-1) + F(n-2)\ \forall n \in \mathbb{N}, n > 2. \tag{3}$$

- How do we calculate $F(n)$ for a given $n \in \mathbb{N}$?

Calculating Fibonacci Numbers

- Obvious Solution: Recursive implementation.

```
def fibonacci1(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fibonacci1(n-1) + fibonacci1(n-2)
```

- How efficient is this?
- Is there a more efficient algorithm?

Calculating Fibonacci Numbers: Second Implementation

- Second Try: Store and reuse intermediate results.

```
def fibonacci2(n):
    f = [0, 1, 1]
    for i in range(3, n+1):
        f.append(f[i-1] + f[i-2])
    return f[n]
```

- Are there any downsides of this implementation?

Calculating Fibonacci Numbers: Third Implementation

- Third Try: Only store the intermediate results that are needed.

```
def fibonacci3(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a+b
    return a
```

- Can we do any better?

Calculating Fibonacci Numbers: Fourth Implementation

- Fourth Try: Fast doubling

```
def fibonacci5(n):
    return _fibonacci5(n)[0]

# Returns the tuple (F(n), F(n+1)).
def _fibonacci5(n):
    if n == 0:
        return (0, 1)
    else:
        a, b = _fibonacci5(n // 2)
        c = a * (b * 2 - a)
        d = a * a + b * b
        if n % 2 == 0:
            return (c, d)
        else:
            return (d, c + d)
```

Calculating Fibonacci Numbers: Fifth Implementation

- Fifth Try: Calculate using direct formula.

```
def fibonacci(n):
    x = (1 + sqrt(5))/2
    return x**n - (1-x)**n)/(x - (1-x))
```

- This produces the answer in one step.
- What is a possible problem with this?

Importance of Studying Algorithms

- We have just seen several different algorithms for solving the same problem.
- The improved algorithms were much more *efficient* than the first one.
- Would you rather have a faster computer or a better algorithm?

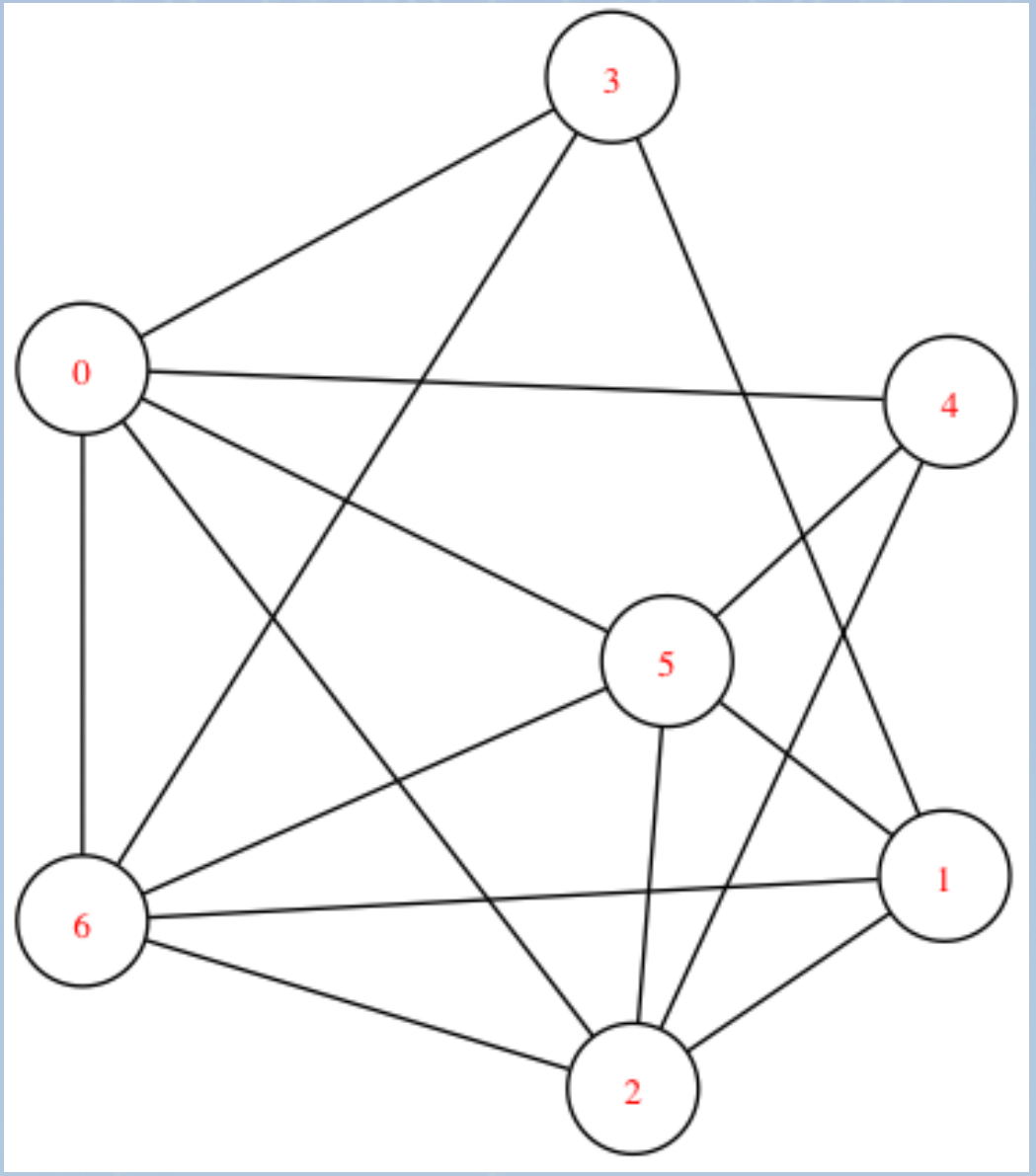
Importance of Studying Algorithms

- We have just seen several different algorithms for solving the same problem.
- The improved algorithms were much more *efficient* than the first one.
- Would you rather have a faster computer or a better algorithm?
  - For large values of n, a 20 year old computer running any of the later algorithms will beat a state-of-the-art computer running the first algorithm.
  - It is generally easier to make performance gains through faster software than through faster hardware.

Graphs

- A *graph* is an abstraction used to model such connectivity relations.
- A consists of a list of "items," along with a list of pairs of items that are connected.
- The study of such graphs and their properties, called *graph theory*, is hundreds of years old.
- Graphs can be visualized easily by creating a physical manifestation.
- There are several variations on this theme.
  - The connections in the graph may or may not have a *direction*.
  - We may not allow more than one connection between a pair of items.
  - We may not allow an item to be connected to itself.
  - The connections may have *weight*, *length*, *cost*, etc.

Example of an Abstract Graph

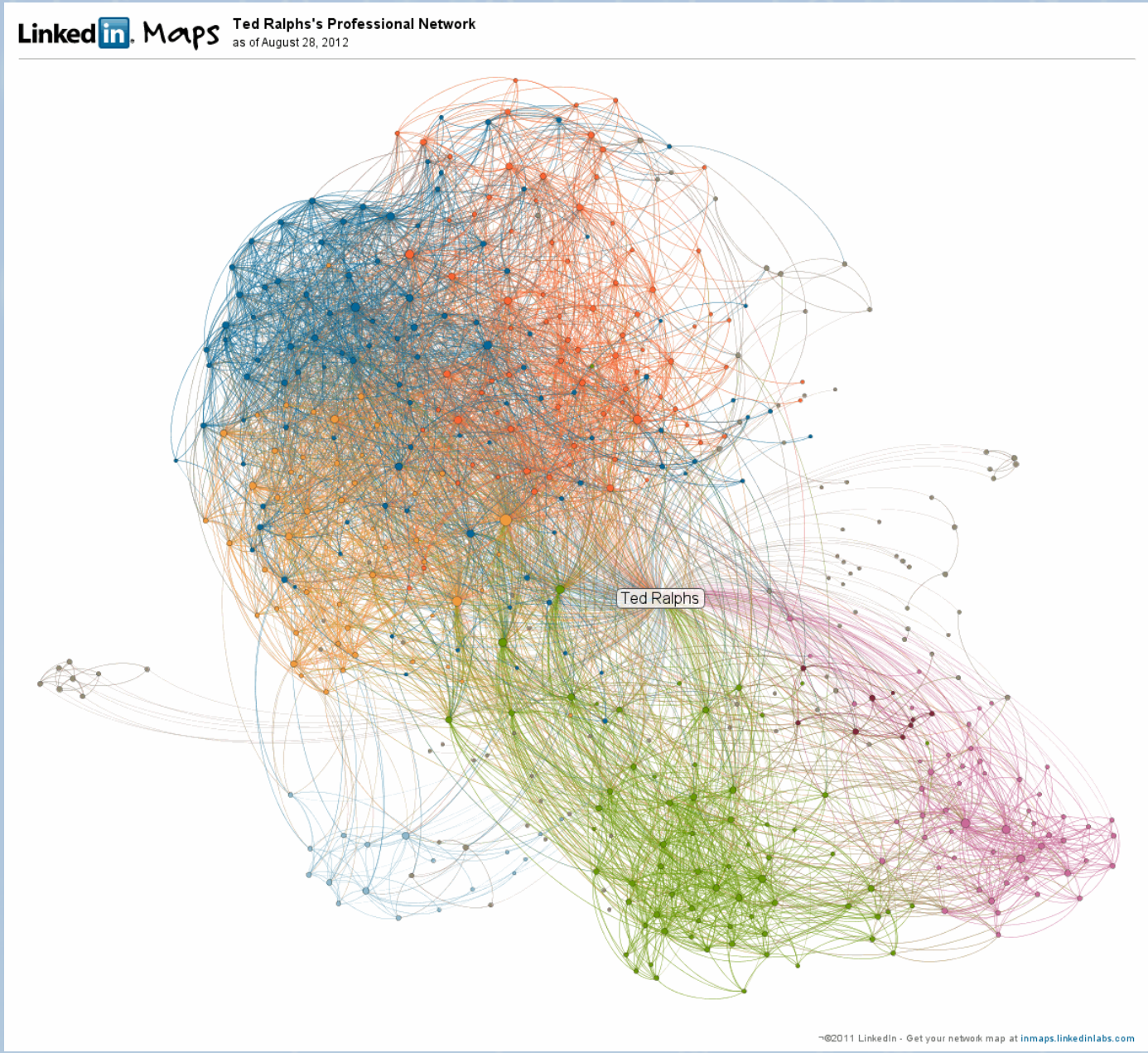


Applications of Graphs

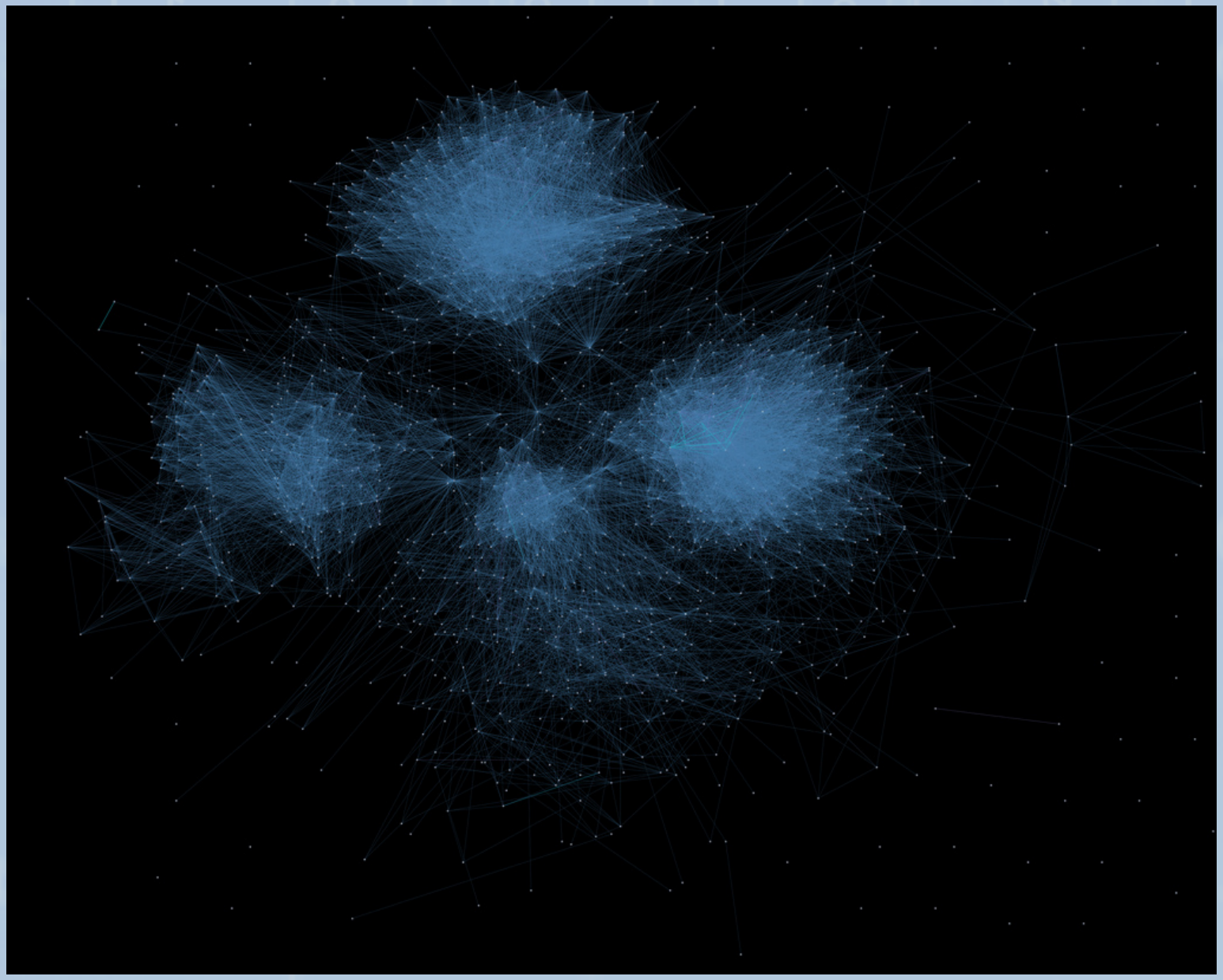
Applications of Graphs

- Maps
- Social Networks
- World Wide Web
- Circuits
- Scheduling
- Communication Networks
- Production and Logistics
- ...

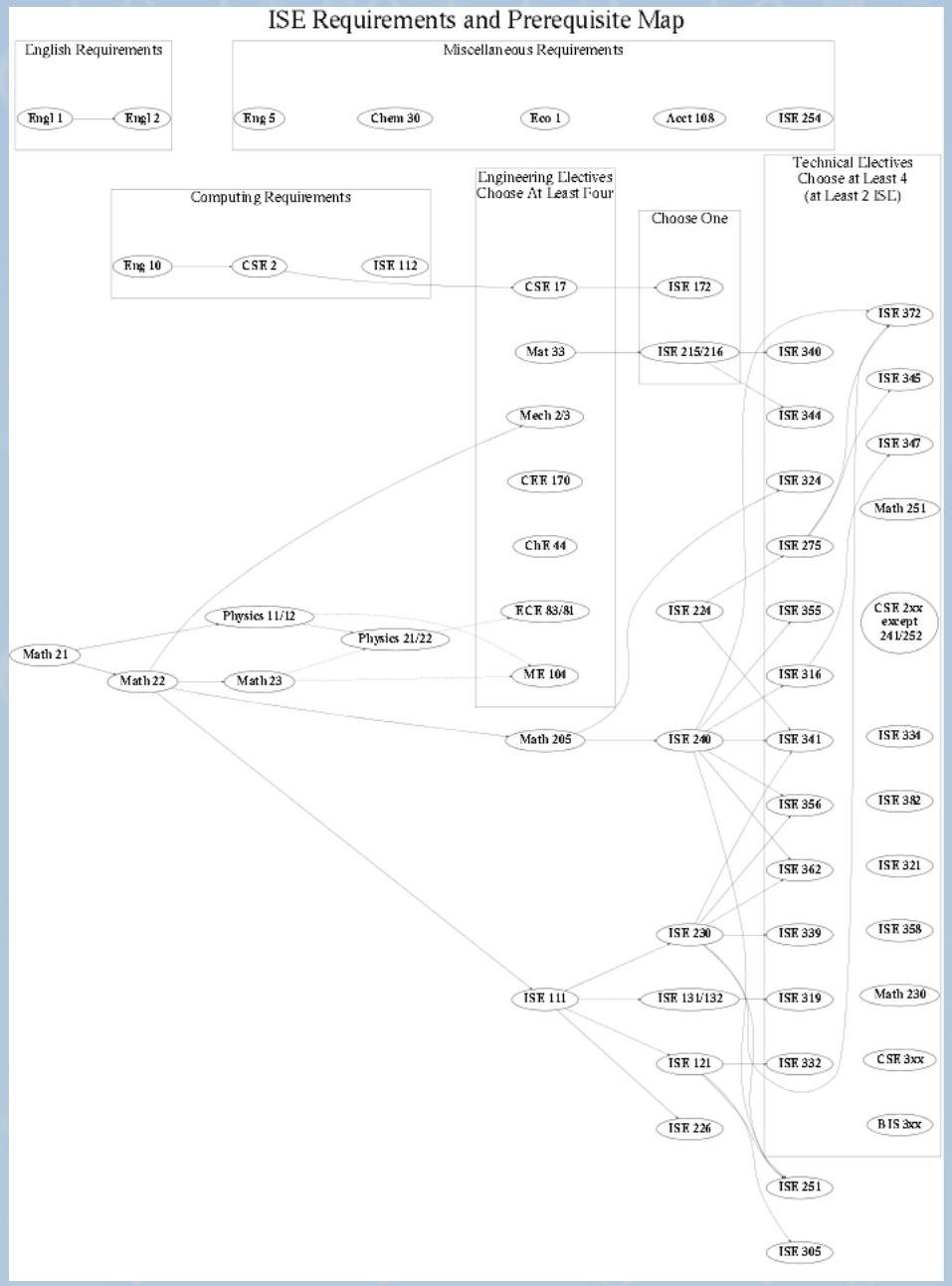
Graphs from Social Networks



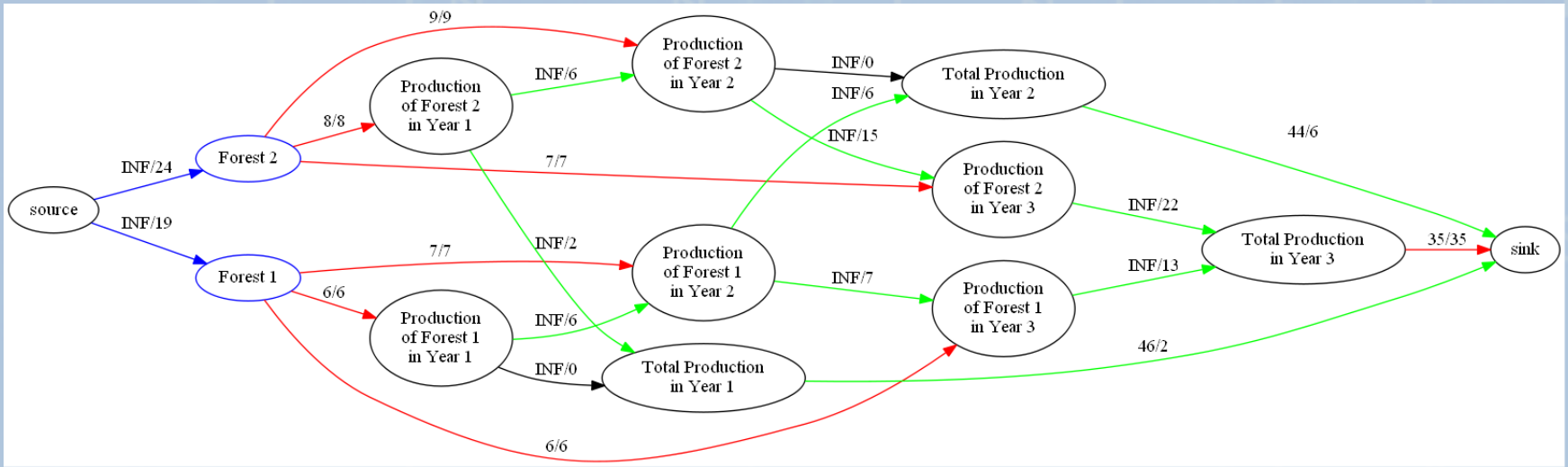
A Facebook Graph



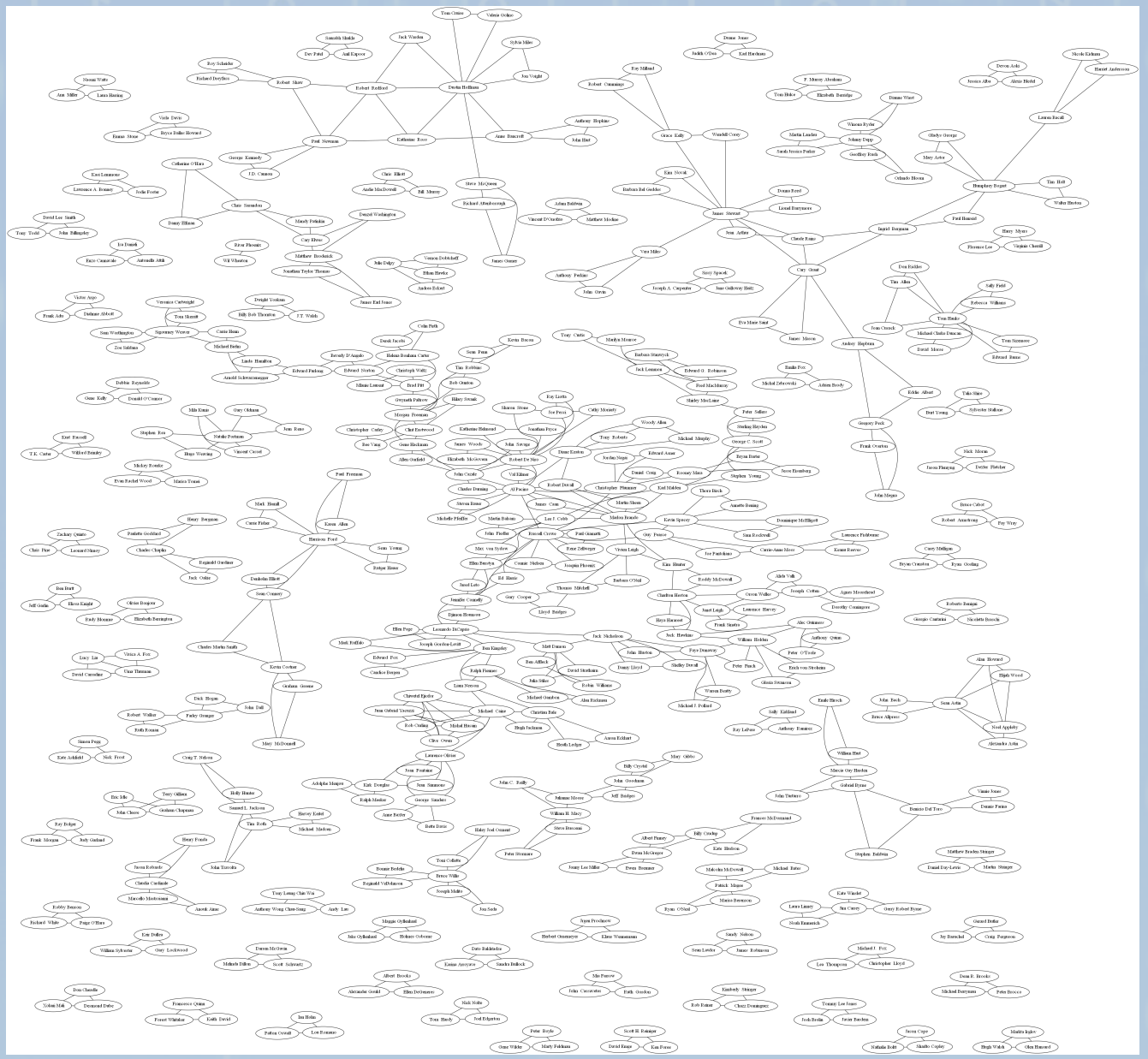
Scheduling


ISE Requirements and Prerequisite Map
English Requirements
Engl 1
Engl 2
Miscellaneous Requirements
Eng 5
Chem 30
Eco 1
Acct 108
ISE 254
Computing Requirements
Eng 10
CSE 2
ISE 112
Engineering Electives
Choose At Least Four
CSE 17
Mat 33
Mech 2/3
CEE 170
ChE 44
ECE 83/81
ME 104
Math 205
Choose One
ISE 172
ISE 215/216
ISE 224
ISE 240
ISE 230
ISE 131/132
ISE 121
ISE 226
Technical Electives
Choose at Least 4
(at Least 2 ISE)
ISE 340
ISE 344
ISE 324
ISE 275
ISE 355
ISE 316
ISE 341
ISE 356
ISE 362
ISE 339
ISE 319
ISE 332
ISE 372
ISE 345
ISE 347
Math 251
CSE 2xx except 241/252
ISE 334
ISE 382
ISE 321
ISE 358
Math 230
CSE 3xx
BIS 3xx
ISE 251
ISE 305
Math 21
Math 22
Math 23
Physics 11/12
Physics 21/22
ISE 111

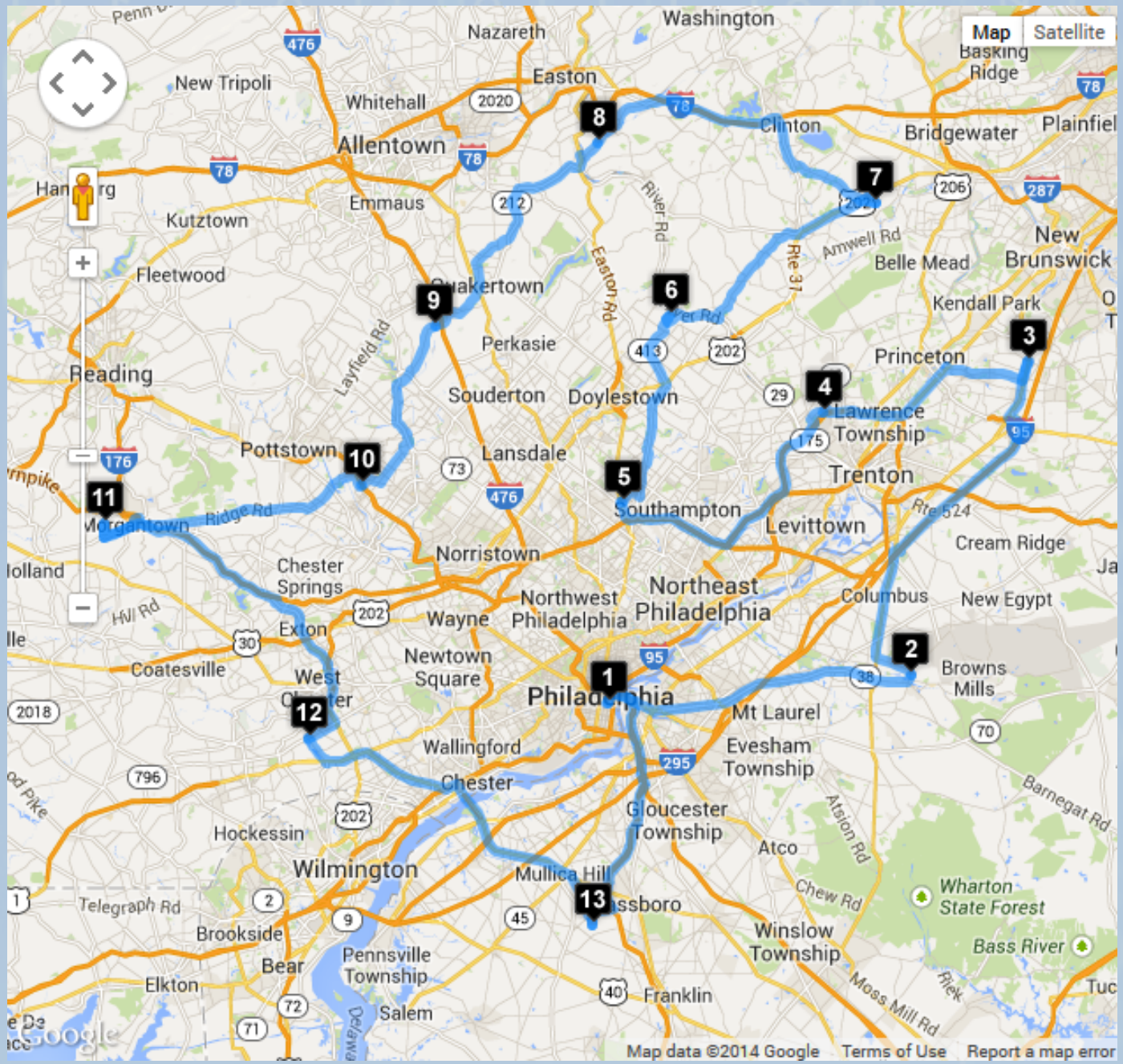
Flow Problems



Graphs for Fun



Graphs from Map Data

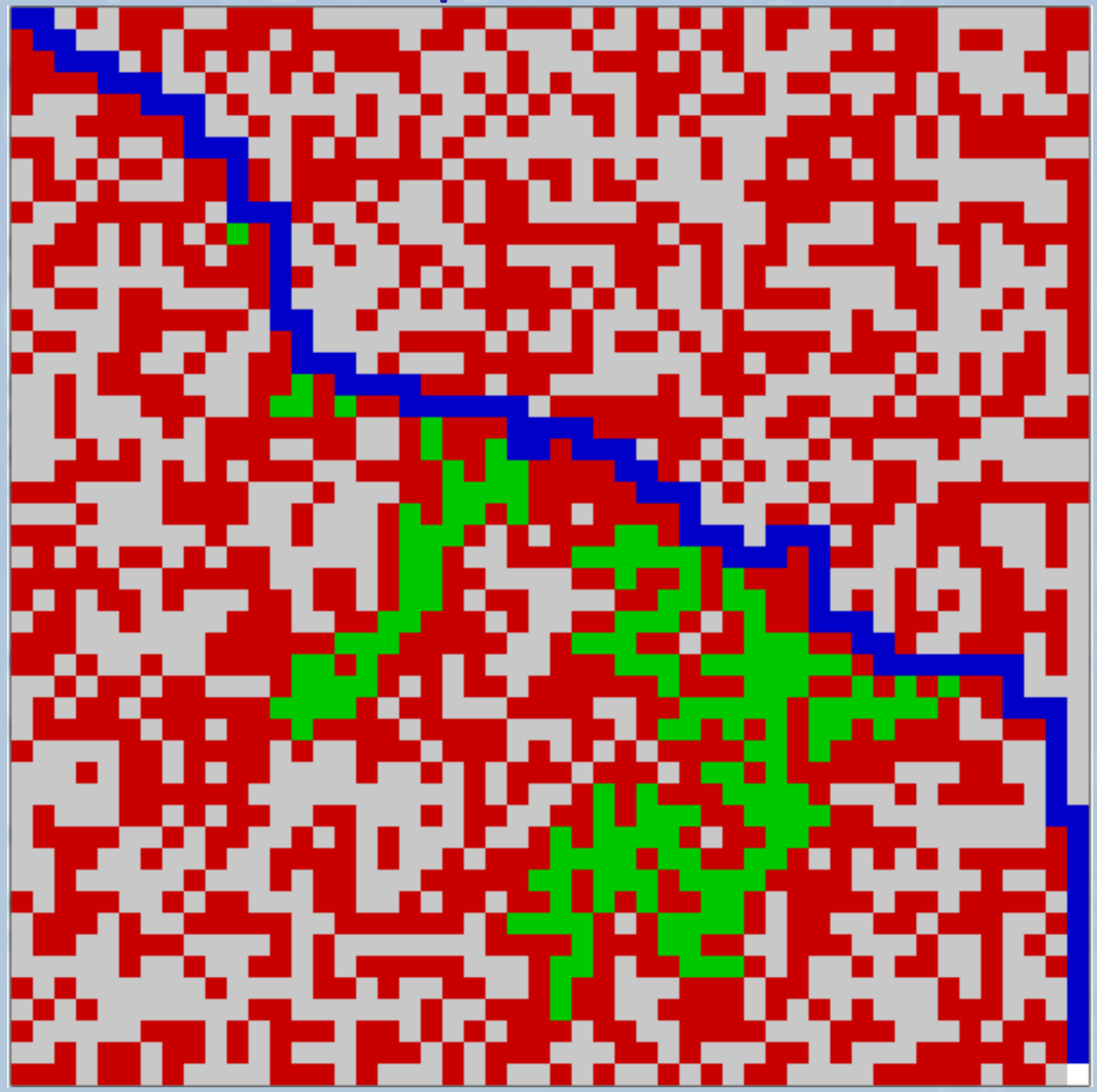


Finding a Simple Path in a Graph

- We now revisit the question of whether there is a path connecting a given pair of vertices in a graph.
- Using the operations in the `Graph` class, we can answer this question directly using a recursive algorithm.
- We must pass in a vector of bools to track which nodes have been visited.

```
def SPath(G, v, w, visited = {})
    if v == w:
        return True
    visited[v] = True
    for n in v.get_neighbors():
      if not visited[n]:
          if SPath(G, n, w, visited):
              return True
    return False
```

A Graph from a Maze



Finding a Hamiltonian Path

- Now let's consider finding a path connecting a given pair of vertices that also visits every other vertex in between (called a *Hamiltonian path*).
- We can easily modify our previous algorithm to do this by passing an additional parameter `d` to track the path length.
- What is the change in running time?

Finding a Hamiltonian Path (code)

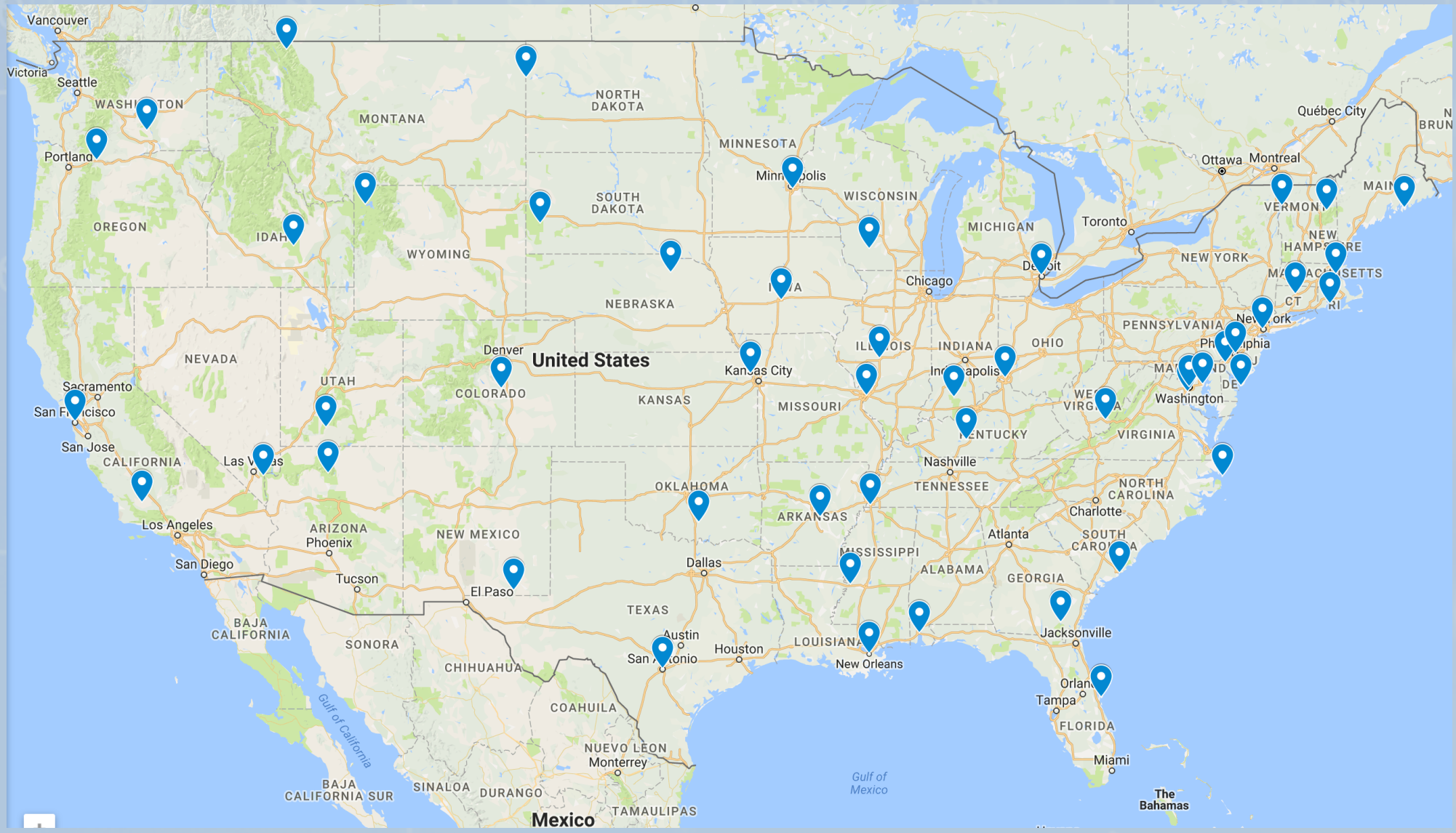
```
def HPath(G, v, w = None, d = None, visited = {})
    if d == None:
        d = G.get_node_num()
    if v == w:
        return d == 0
    if w == None:
        w = v
    visited[v] = True
    for n in v.get_neighbors():
      if not visited[n]:
          if SPath(G, n, w, d-1, visited):
              return True
    visited[v] = False
    return False
```

What is the difference? Why is it more difficult?

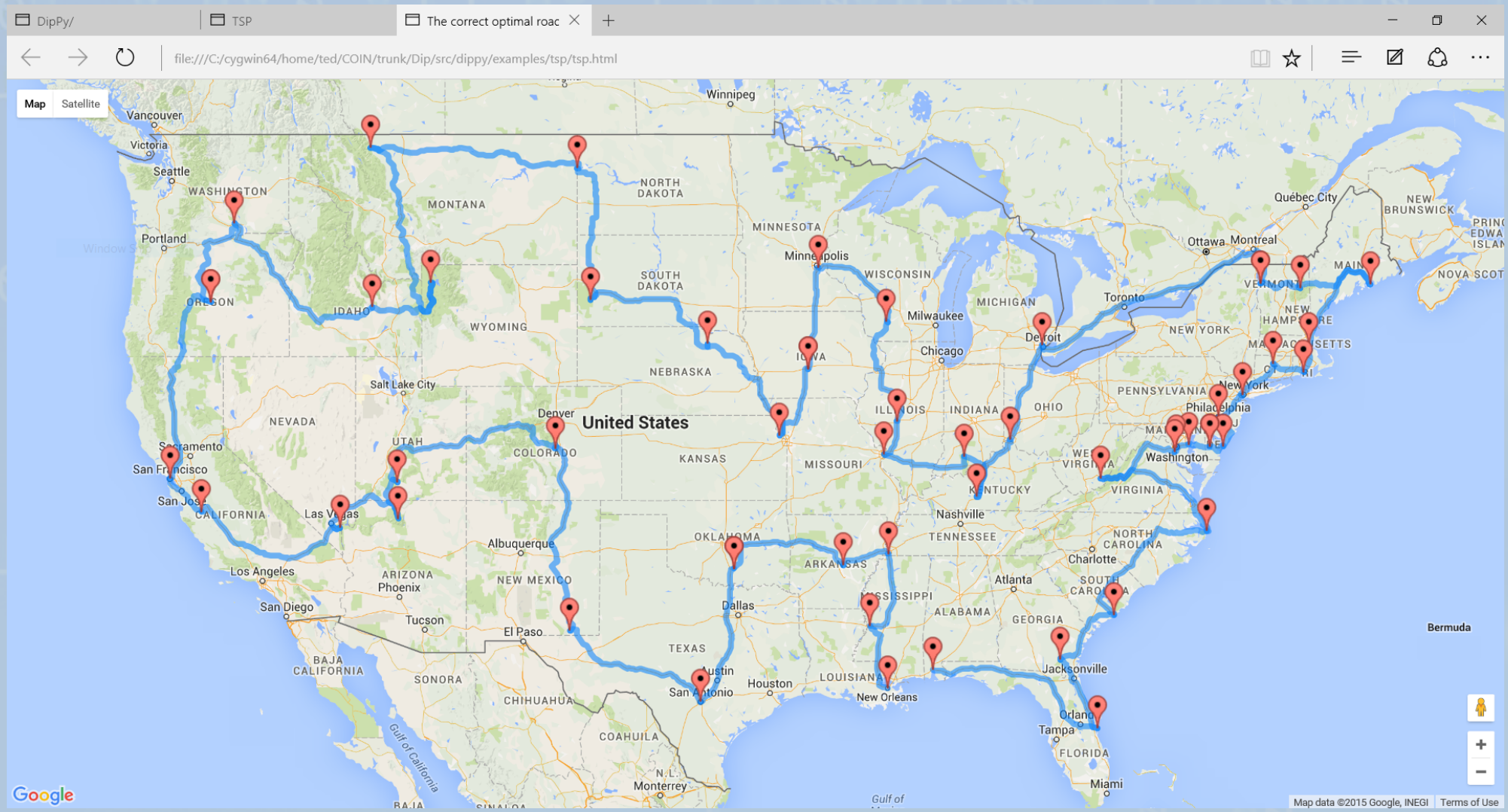
The Traveling Salesman Problem

- In the *Traveling Salesman Problem*, we are given a list of locations that a salesman needs to visit.
- The salesman starts at home and then must visit all cities in some order, returning back to home.
- The goal is to make the overall route as short as possible.
- Consider an instance with 50 location.
  - Number of possible solutions: $\approx 10^{64}$.
  - Number of atoms in the universe: $\approx 10^{79}$.
  - Enumeration of all solutions is not an option.
  - Instead, we do intelligent, partial enumeration.

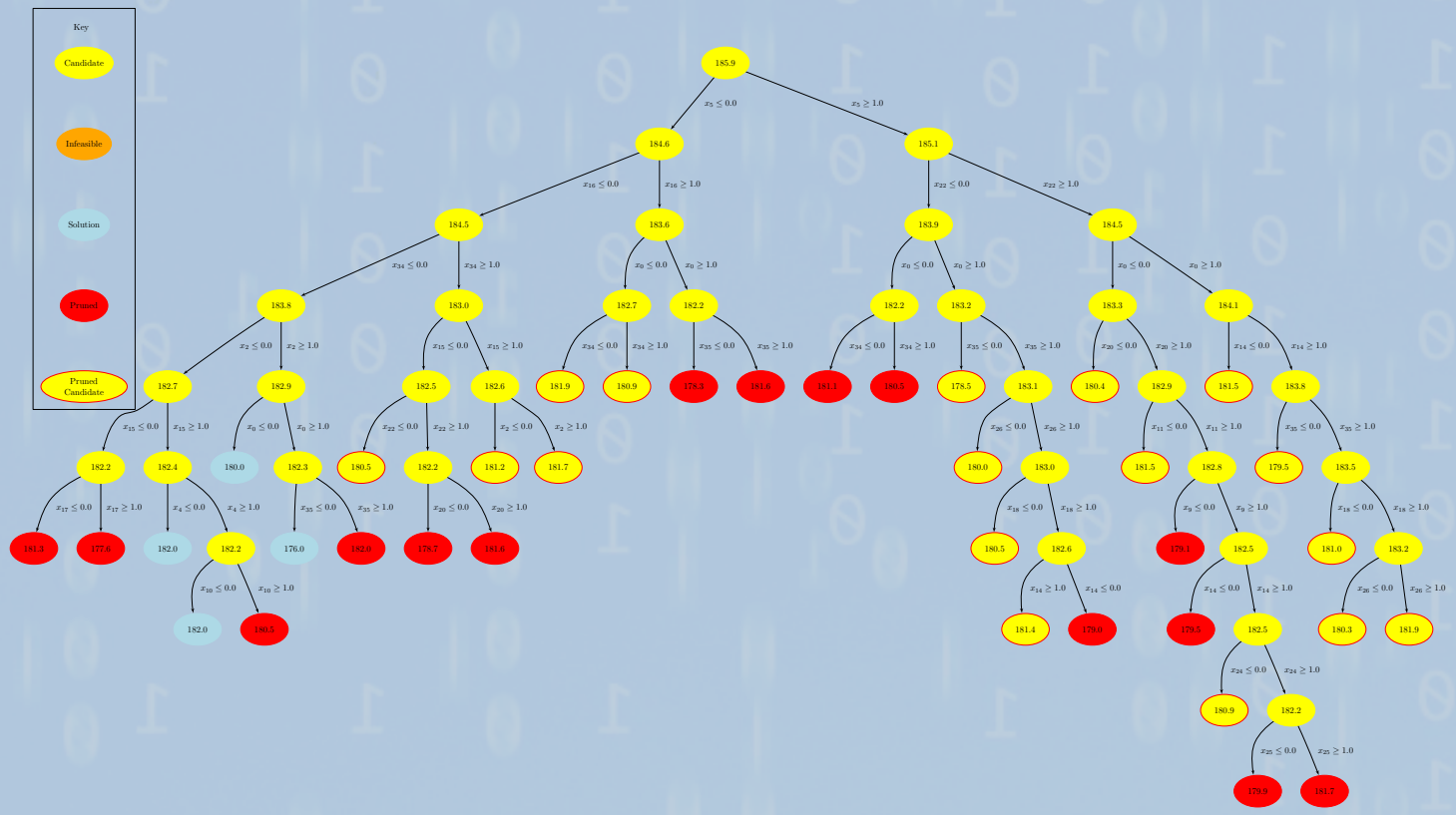
Example TSP


Vancouver
Victoria
Seattle
Portland
MONTANA
NORTH DAKOTA
MINNESOTA
SOUTH DAKOTA
WISCONSIN
MICHIGAN
OREGON
IDAHO
WYOMING
NEBRASKA
Chicago
Toronto
Ottawa
Montreal
Québec City
NEW YORK
PENNSYLVANIA
NEVADA
UTAH
COLORADO
Denver
United States
Kansas City
KANSAS
MISSOURI
INDIANA
OHIO
Washington
Sacramento
San Jose
CALIFORNIA
Los Angeles
San Diego
ARIZONA
Phoenix
Tucson
NEW MEXICO
El Paso
OKLAHOMA
Dallas
TEXAS
ARKANSAS
TENNESSEE
Nashville
VIRGINIA
NORTH CAROLINA
Charlotte
Atlanta
GEORGIA
ALABAMA
MISSISSIPPI
LOUISIANA
New Orleans
Houston
Austin
Jacksonville
Tampa
FLORIDA
Miami
The Bahamas
Gulf of Mexico
BAJA CALIFORNIA
BAJA CALIFORNIA SUR
SONORA
CHIHUAHUA
COAHUILA
NUEVO LEON
Monterrey
SINALOA
DURANGO
TAMAULIPAS
Mexico
Gulf of California

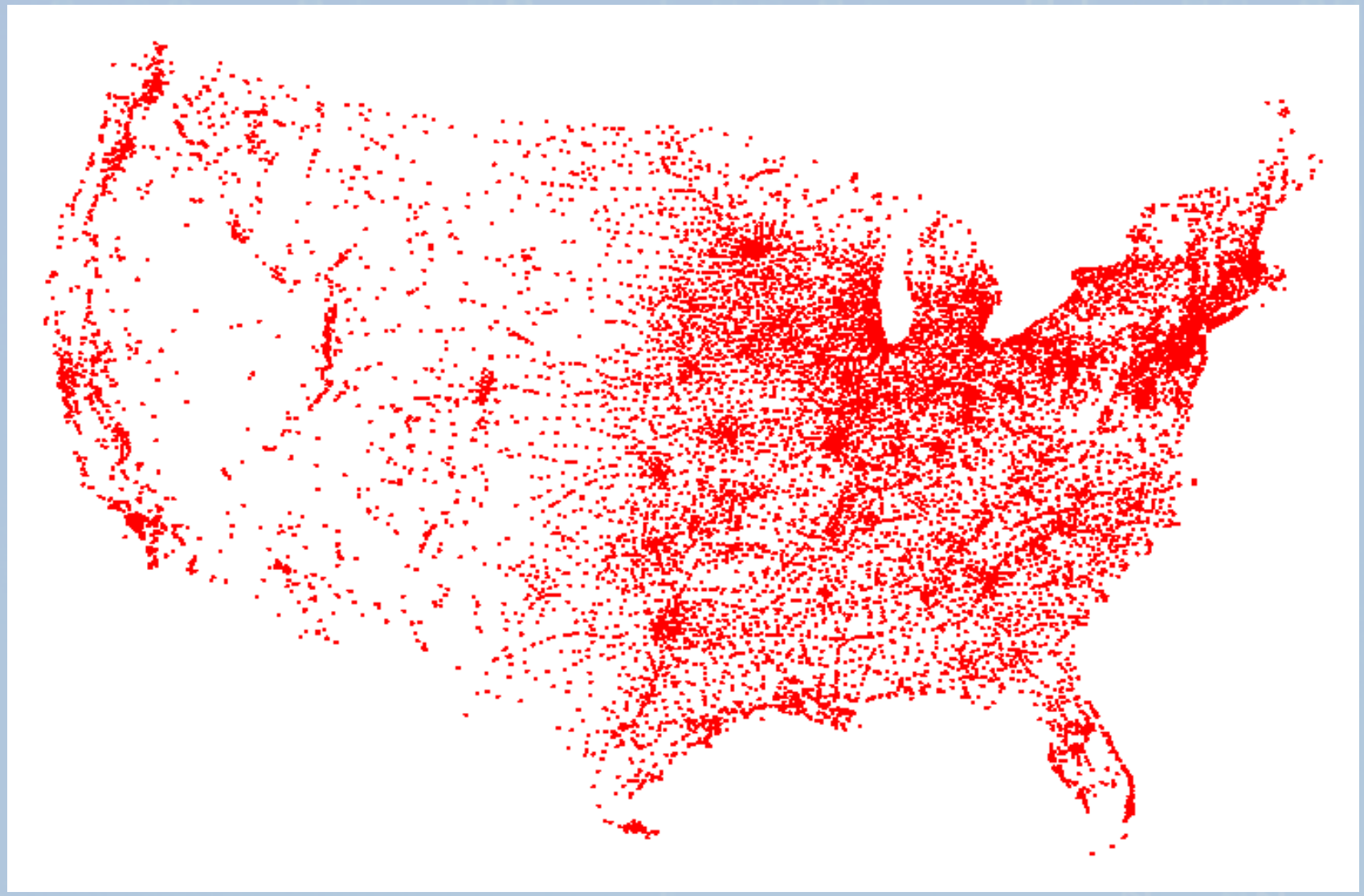
Example TSP


DipPy/
TSP
The correct optimal roac
file:///C:/cygwin64/home/ted/COIN/trunk/Dip/src/dippy/examples/tsp/tsp.html
Map
Satellite
United States
Winnipeg
Vancouver
Victoria
Seattle
Portland
WASHINGTON
OREGON
MONTANA
IDAHO
WYOMING
NORTH DAKOTA
SOUTH DAKOTA
NEBRASKA
MINNESOTA
Minneapolis
WISCONSIN
MICHIGAN
Milwaukee
Chicago
IOWA
ILLINOIS
INDIANA
OHIO
Detroit
Toronto
Québec City
Ottawa
Montreal
NEW BRUNSWICK
NOVA SCOTIA
NEW YORK
PENNSYLVANIA
New York
Philadelphia
Washington
VIRGINIA
Salt Lake City
NEVADA
UTAH
Denver
COLORADO
KANSAS
MISSOURI
KENTUCKY
TENNESSEE
Nashville
Charlotte
NORTH CAROLINA
SOUTH CAROLINA
Atlanta
GEORGIA
ALABAMA
MISSISSIPPI
LOUISIANA
New Orleans
Jacksonville
Tampa
FLORIDA
Miami
Bermuda
Sacramento
San Francisco
San Jose
CALIFORNIA
Las Vegas
Los Angeles
San Diego
ARIZONA
Phoenix
Tucson
Albuquerque
NEW MEXICO
El Paso
TEXAS
Dallas
Austin
San Antonio
Houston
OKLAHOMA
ARKANSAS
BAJA CALIFORNIA
SONORA
CHIHUAHUA
COAHUILA
Monterrey
Gulf of California
Google
Map data ©2015 Google, INEGI
Terms of Use

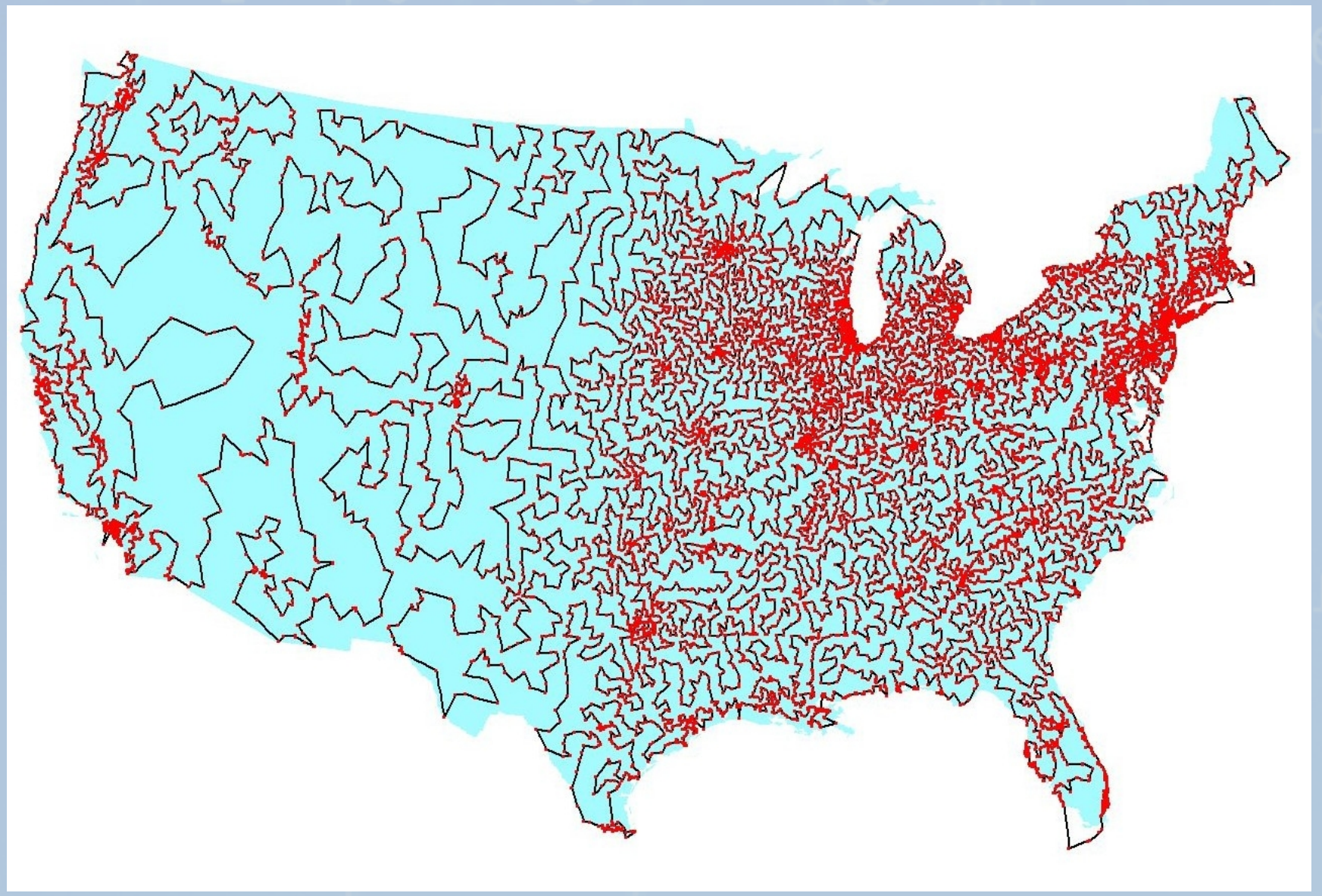
Needle in the Haystack



A Large-scale Instance: USA13509



The Solution



Summing Up

- Algorithms that are both efficient and correct are a technology that must be developed.
- Data structures allow us to represent relationships between various data and allow us to manipulate them effectively during an algorithm.
- Efficient algorithms enable us to solve important problems more quickly, which is critical for many applications.
- Development of algorithms requires an understanding of theory (mathematics) and computation.
- There are many interesting challenges to be addressed in solving optimization problems and this is a good way to have an impact on the world!
- Good luck and above all, have fun!

Thank You!

Questions

- What nerdy books do you enjoy most? What books do you think are critical to read for an industrial engineer?
- What are some common difficulties in implementing algorithms efficiently at scale?
- How does OR change from country to country? Is it difficult to communicate with people who are not in the US and did not receive the same education as you?
- Is there a method you use to narrow down the algorithms before deciding on one?
- Regarding optimization techniques, are there algorithms/methods to solve current real-world issues that work well in theory but with computing power requirements that are too high for current technology (silicon transistor sizes, etc)?
- What steps would you take if given an inefficient algorithm in code and then given the task to optimize it? Also, could/do you apply your optimization problem skills to CPU architecture as well?

- This class is just our first introduction into operations research. If we want to continue in this direction, what are a few different routes we can take?
- What kind of process is required when transferring an OR problem from the theory on a whiteboard using figures to a coding program and/or make it a more realistic problem? Do you have to ignore certain aspects of the theory to make it realistic, and if so, how do you decide what gets left out?
- I've studied short path algorithms before in computer science, I was wondering how often programming is used in the Industrial Engineering field. Is there a special place for industrial engineering in computer science, such as for optimizing problems besides path algorithms?
- What got you interested in the Traveling Salesman problem? Is there a connection in your life that spurred the drive to find a solution to the complex problem?
- After reading through the linear programming formulation for The Traveling Salesman along with a few online articles, I still am unclear on how the algorithm finds the solution in a short amount of time. Could you general overview of how this formulation operates?
- Do you find theory or computation harder? Is it difficult to transfer

theory over to computation or does a solid understanding of theory make computation much easier?

- What affect, if any, does Quantum Computing have on Modern Operations Research. I have read articles that state the TSP problem is used to measure the speed of Supercomputers.

- While working with computationally complicated problems, such as the traveling salesman problem, do you use more advanced operations research specific computing programs to assist in solving these problems? If so, which programs are used?

- What is something that you emphasize to your students about being successful in the world of Industrial and Systems Engineering?

- Hello! As a college student, I'd like to know your opinion about studying abroad as an Industrial Engineering major. I'm currently looking for different options abroad and there are so many to choose from! Based on your past experiences, do you have any advice, programs in mind, or ways in general to go abroad in the Industrial Engineering major? For example, should I be taking classes or working or just traveling...etc.

- Do you really have five tattoos of the same thing in different languages on your body, if so what languages?

- Why does Dijkstra's algorithm (the algorithm used for google maps/traveling salesmen problems) become so much more complicated than shortest path models?
- What are some of the most common and "unpredictable" fallacies that occur when a simple optimization problem is applied at a macro scale and how are these problems then addressed in the constraints of the problems or analysis of the results?
- How long did it take you to set up the "perfect college tour" problem, with all of the research that went into it?
- Other than finding the best route between cities for a travelling salesman, what is the best application for solutions to the travelling salesman problem?
- What inspired you to focus your career on operations research? What are you specifically researching at the Zuse Institute? What made you want to focus on discrete optimization? Have you ever encountered a situation where the implementation of algorithms ended up producing worse results than before?
- What previously computationally difficult problems have we now resolved? And why and how? Do you foresee future problems being resolved? And in what manner?