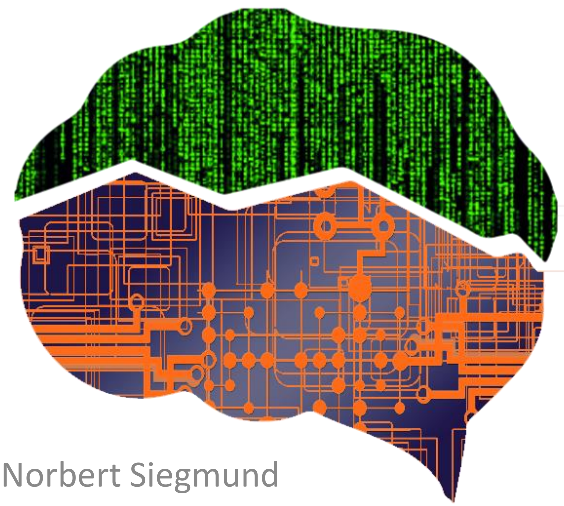
Search-Based Software Engineering

Introduction and Motivation



Prof. Dr.-Ing. Norbert Siegmund
Intelligent Software Systems

Bauhaus-Universität Weimar

Organizational Stuff

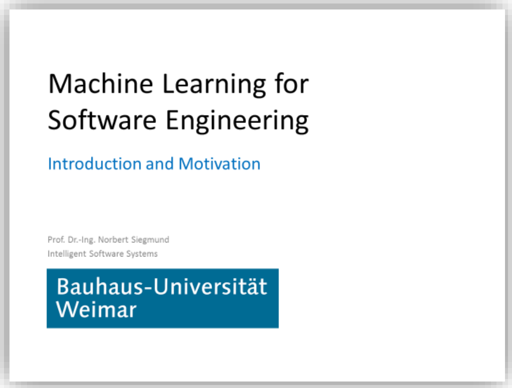
- Lecture: Monday 09:15-10:45 in room SR015
  - Cover broad idea of topic
  - Presents theoretical foundations of algorithms
  - Give examples
- Exercise: Tuesday 11:15-12:45 in room SR014
  - Evaluate and implement algorithms
  - Starting at 16th of April
- Slides on the Website (+ some code examples)

Tasks and Exam

- Tasks:
  - Several small tasks over the semester
  - Requirement for exam (CS4DM 4.5ECTS; DE 6ECTS -> more tasks)
  - Python required!
  - Optional: Datacamp Python courses: https://www.datacamp.com/groups/shared_links/24b6408183201a4e5c49dc853f7ff21210d876b4

- Exam: Oral or written depending on the number of students

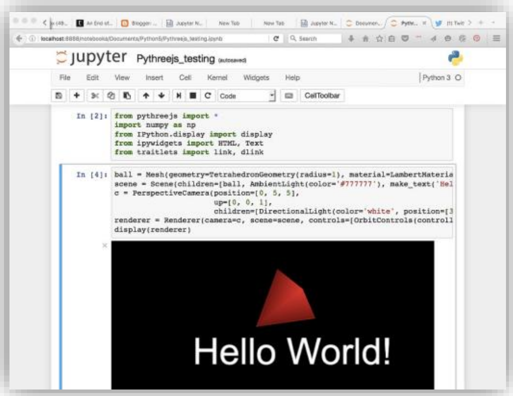
How to Attend the Lecture



Slides: Algorithms, theory, summary, visualizations, important points

Whiteboard: Derivation of algorithms, examples, visualizations

Discuss!

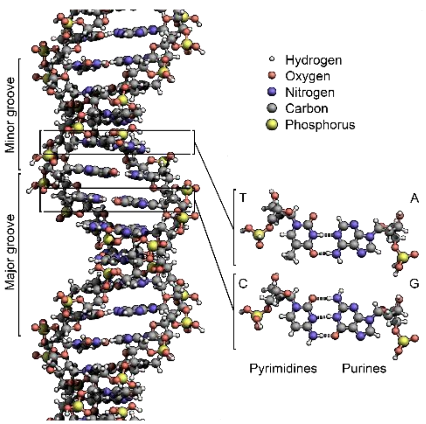


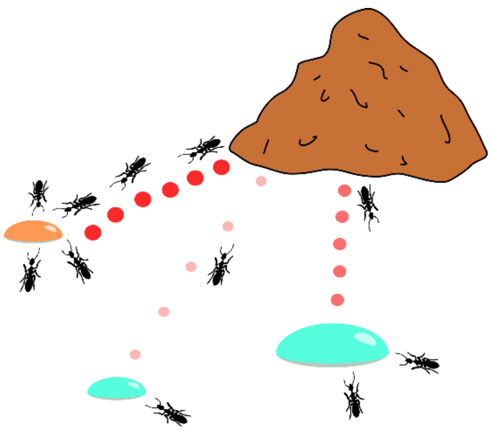
Programming: Live demonstration, visualization, and testing

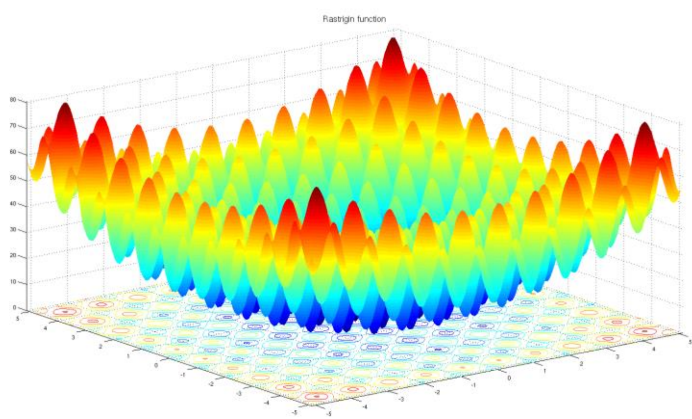
Overview

Topics of the Course

- Meta-Heuristics for optimization
  - Simulated annealing
  - Genetic and evolutionary algorithms
  - Particle swarm optimization
  - Ant colonization

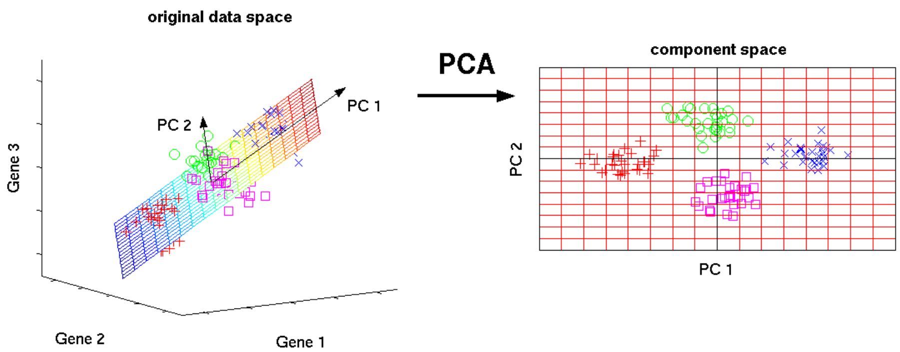




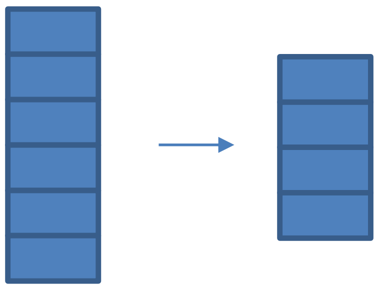


Topics of the Course

- Dimensionality reduction

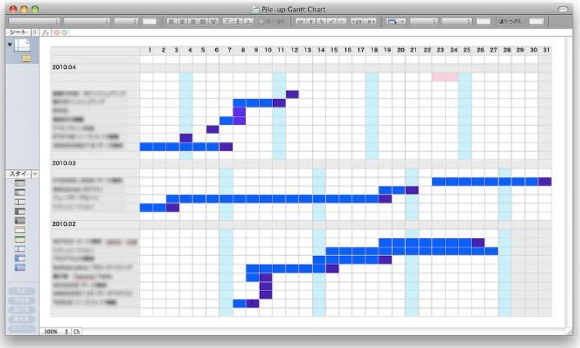


Principle component analysis (PCA)



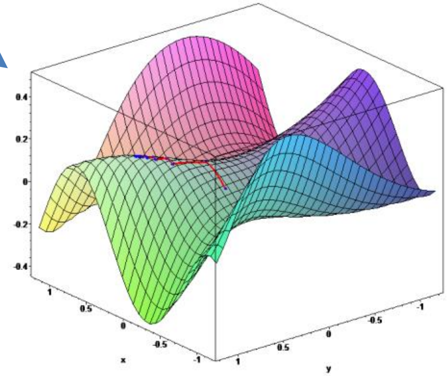
Feature selection

What’s the Relation of Optimization to Software Engineering?



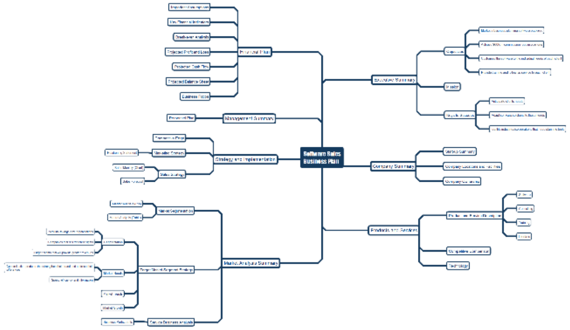
Software development cost estimation

Software testing



Optimization tasks

Software quality assurance

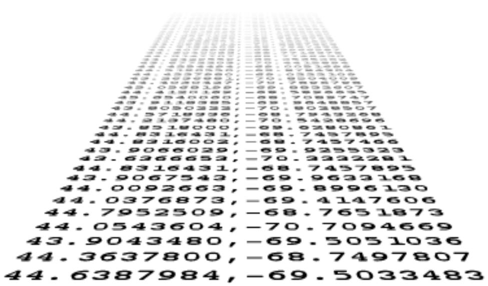


Configuration management

What's the Relation of Dimensionality Reduction to Software Engineering?

Which code change decreased my performance?

What is the most performance-critical feature?



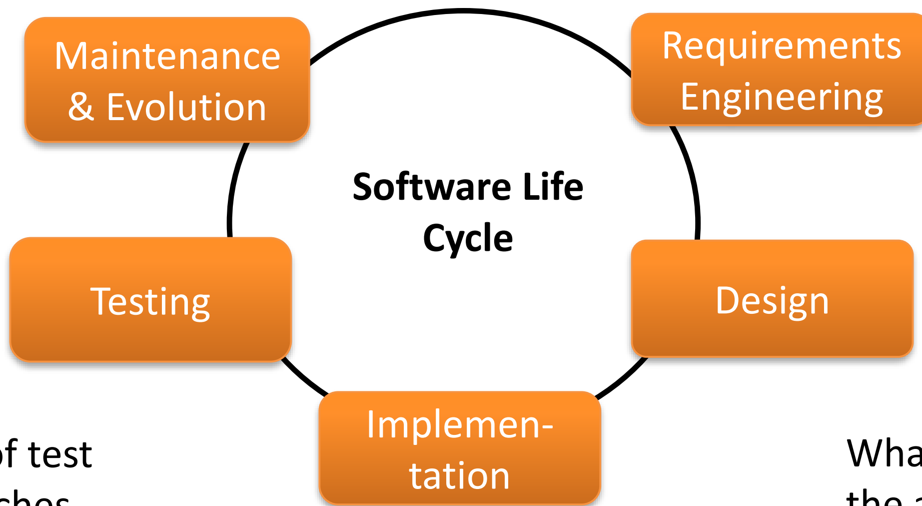
	A	B	C	D	E
1	721111	838195	492655	721111	838195
2	391781	338317	626697	391781	338317
3	733567	713921	764558	733567	713921
4	414992	419754	134267	414992	419754
5	721534	175785	389954	721534	175785
6	344821	359643	548463	344821	359643
7	992494	965532	149188	992494	965532
8	459663	733145	324676	459663	733145
9	712153	826921	778719	712153	826921
10	747953	527973	151976	747953	527973

Reduce the amount of data for any learning, optimization, prediction, and analysis technique

Search-Based Software Engineering

What is the best sequence of refactoring steps to apply to this system?

What is the set of requirements that balances software development cost and customer satisfaction?



What is the smallest set of test cases that covers all branches in this program?

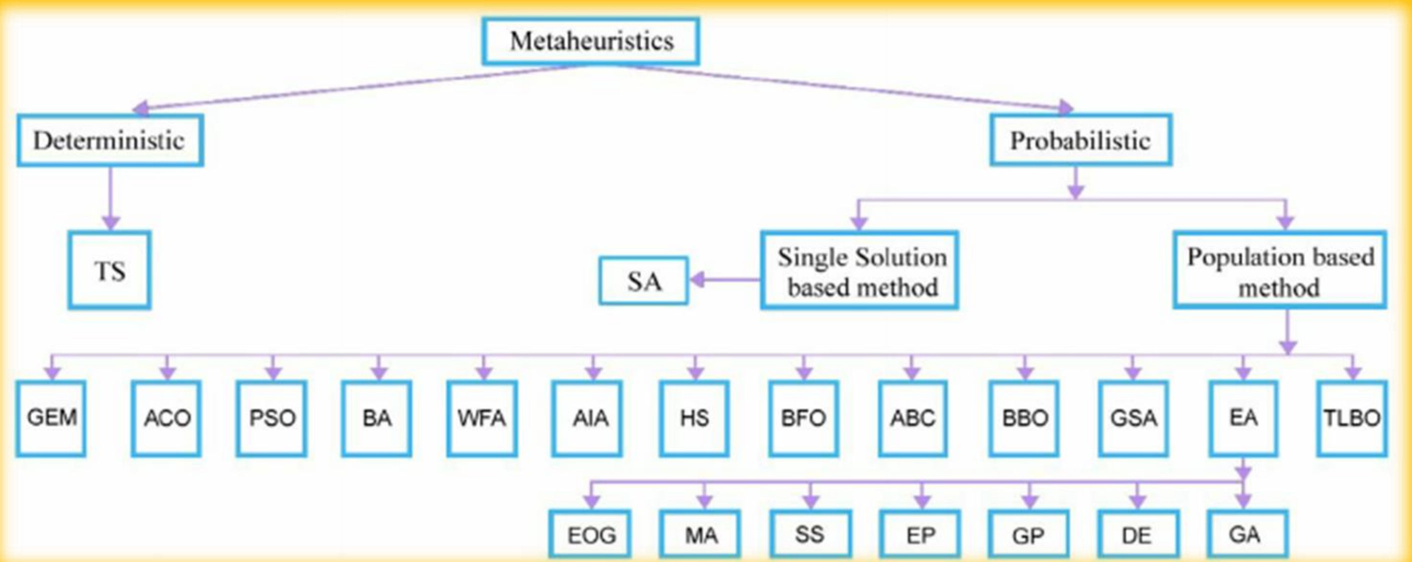
What is the best way to structure the architecture of this system to enhance its maintainability?

What is the best allocation of resources to this software development project?

Search-Based Software Engineering: Trends, Techniques and Applications

MARK HARMAN, University College London
S. AFSHIN MANSOURI, Brunel University
YUANYUAN ZHANG, University College London

Meta-Heuristics for Optimization



What You Should Learn

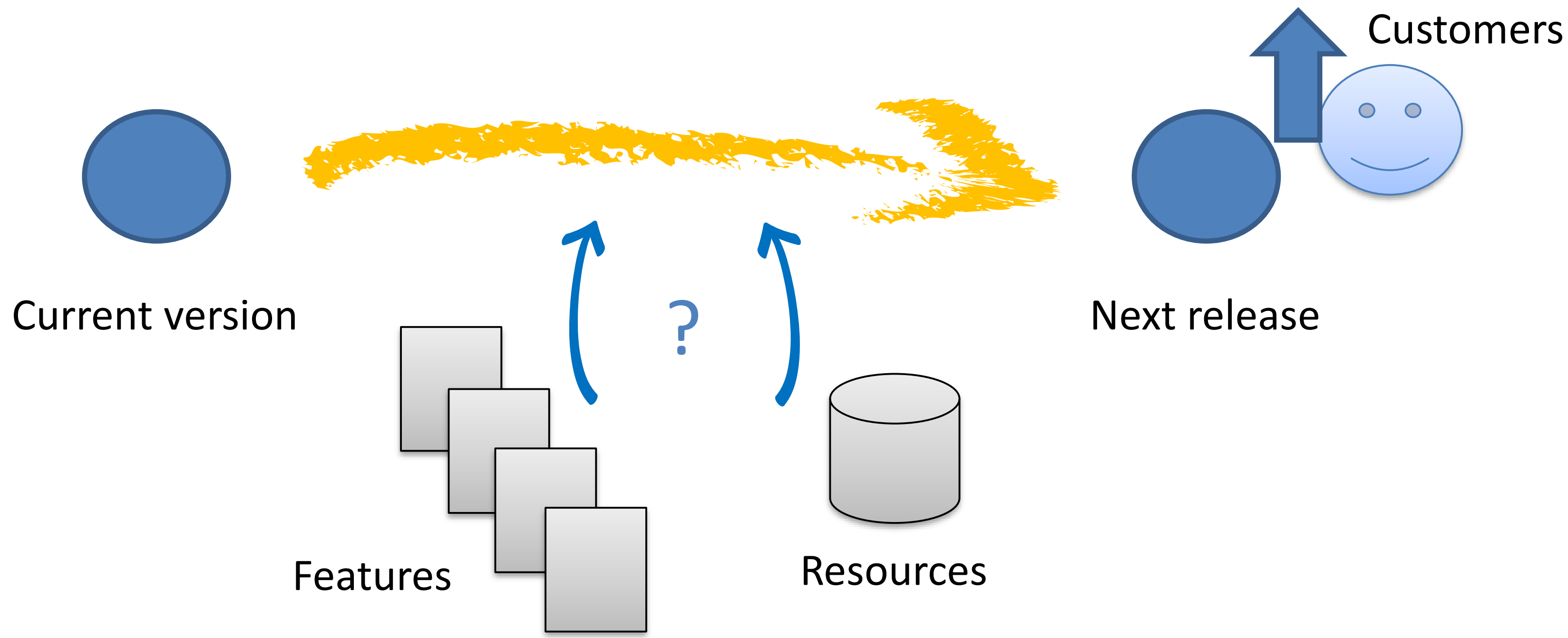
- What are optimization problems?
- When to use approximative vs. exact optimization techniques?

- What are the different strategies to find good solutions?
  - Single state techniques
  - Multiple state techniques
  - Combinatorial techniques

- How to tune exploitation vs. exploration?

Introduction I

- Many topics in SE and other fields aim at finding the best *setting* to achieve a goal
  - Example: The next release problem (NP-hard)



Introduction II

- Finding good designs can also be represented as an optimization problem:
  - Find an architecture (HW architecture or SW architecture),
  - Find a placement of services, components, modules on HW devices
  - Find a system design for communication
  - Find a schedule of tasks and process

- Whereas
  - A cost function (e.g., performance, errors, communication effort, etc.) is minimized, and
  - A set of constraints is simultaneously satisfied

Formalizing Optimization

Minimize $f(x)$

Subject to $g_i(x) \geq b_i;\ i = 1,2, \dots, n$

Where

x is a vector of decision variables;
f is the cost (objective) function;
g_i's are a set of constraints.

- Find an input value to function f such that the function takes its minimum value and the constraints over the input values are satisfied

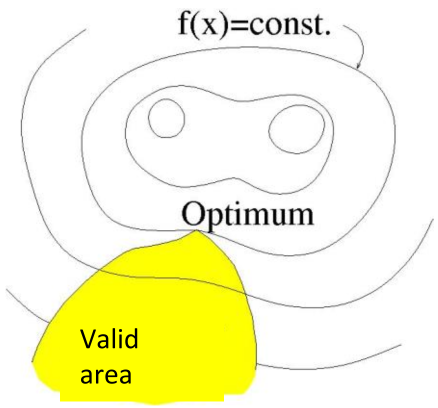
Types of Optimization Problems

- Linear Programming (LP)
  - Linear cost (objective) function
  - Linear constraints
  - Algorithms: Simplex, Interior Point
  - Finds exact solution(s)

Minimize $f(x) = c^T x$

Subject to
$$Ax \geq b$$
$$x \geq 0$$
$$x \text{ in } R^n$$

- Nonlinear Programming (NLP)
  - Constraints or cost function are nonlinear
  - Examples: quadratic programming
  - Does not find exact solution(s) efficiently



Types of Optimization Problems

- Integer Programming (IP)
  - Similar to LP and NLP with $x \in Z^n$ with 0/1 as special case
  - Also linear integer programming ILP
  - NP-hard
- Mixed Integer Programming (MIP)
  - Mix of real and natural numbers

Harder than LP problems, because no derivate information is available and the surface of the solution space is not smooth

- So far: Continuous optimization problems with an infinite number of feasible solutions
- Now: Combinatorial problems with a finite number of valid solutions

Combinatorial Optimization (CO)

- Decisions variables are discrete such that a solution is a set or sequence of integers or other discrete objects
- Formalization of a combinatorial optimization problem:
  - Input is a set D_P of instances
  - Output is a finite set $S_P(I)$ of solutions for each instance of $I \in D_P$ and
  - A function m_P which maps for each solution $x \in S_P(I)$ in every instance $I \in D_P$ a positive, real number as the solution value: $\mathrm{y} = m_P(x, I)$
- Optimal minimal solution:
  - $x^* \in S_P(I)$ with $m_P(x^*, I) \leq m_P(x, I) \forall x \in S_P(I)$

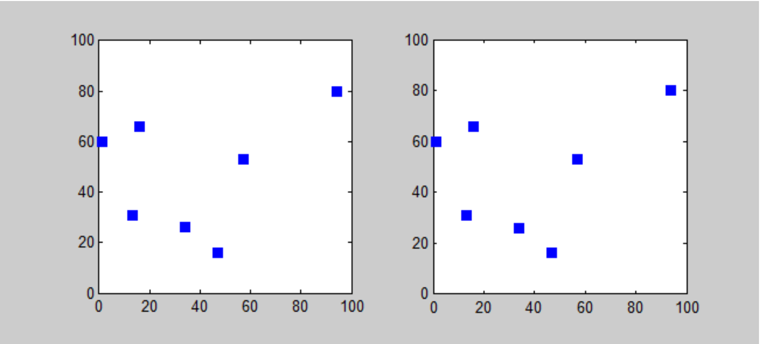
Combinatorial Optimization Properties

- Most CO problems are NP-complete, which results in an exponential increase of computation time with respect to the problem size n
- Exact approaches:
  - Reformulated as an ILP problem
  - However, only small problems can be easily solved
- Approximate approaches are needed
  - Using heuristics to search through the space of feasible (valid) solutions to find the optimal one

Traveling Salesman Problem (TSP)

- Goal: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?
- NP-hard problem (formulated in 1930)

- Applications:
  - Planning, logistics, manufacturing
  - DNA sequencing, astronomy



TSP Formalization

- Goal: Minimize the round trip path
- Solution: Order of the cities from 1 to n (permutate the set from 1 to n)
- Encoding:
  - Distances are stored in a matrix $d_{i,j}$
  - Going from city i to j is expressed by $x_{i,j} = 1$ and 0 otherwise
- Formulation:

$$minimize \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} x_{i,j}$$

subject to

$$\sum_{i=1, i \neq j}^{n} x_{i,j} = 1 \; \forall \, j \in \{1, \dots, n\}$$

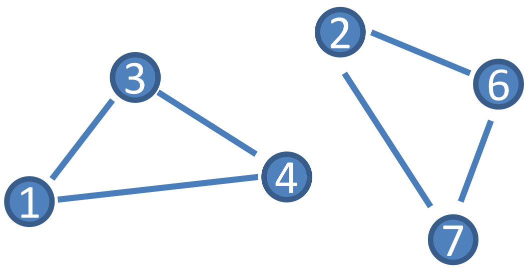
$$\sum_{i=1, i \neq j}^{n} x_{j,i} = 1 \; \forall \, j \in \{1, \dots, n\}$$

$$x_{i,j} \geq 0 \; (i = 1,2, \dots, n; j = 1,2, \dots, n)$$

Avoiding Disjoint Tours

- 2n-1 additional constraints must be added
  - Eg.:

$$x_{2,1} + x_{2,3} + x_{2,5} + x_{6,1} + x_{6,3} + x_{6,4} + x_{7,1} + x_{7,3} + x_{7,4} \geq 1$$



- Number of possible solutions: (n-1)!/2
- $n! > 2^n > n^3 > n^2 > n$

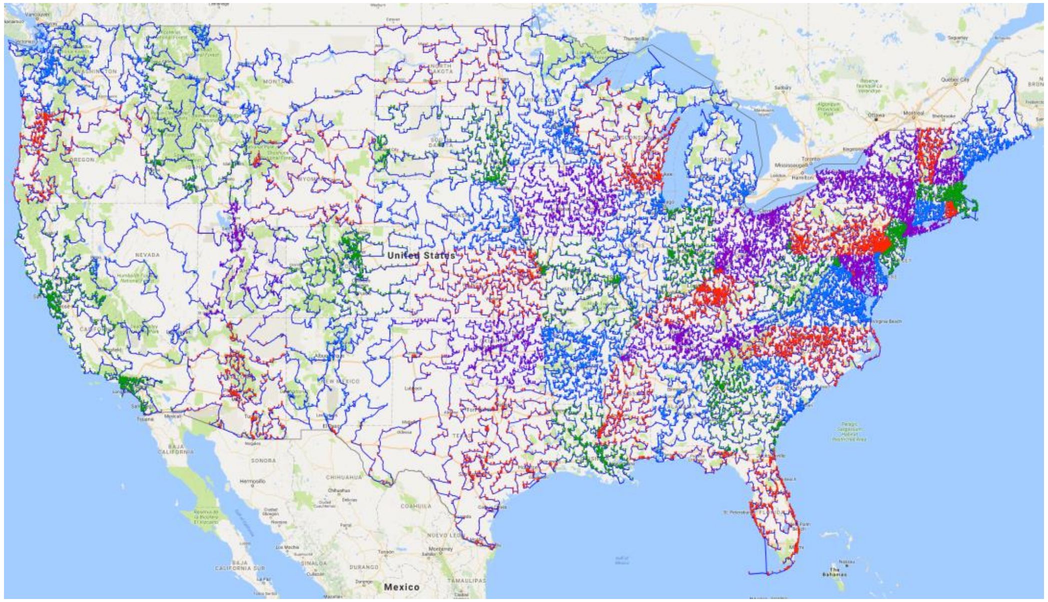
Branch & Bound

- Implicitly enumerates all solutions of a combinatorial problem
- Gains its efficiency by eliminating (cutting) subtrees
- Branching
  - Splits the solution into disjunctive sub problems
- Bound
  - Use upper and lower bounds of the values of the cost function
  - Upper bound = best found solution for minimization
  - Lower bound = If lower bound > upper bound => eliminate subtree; if not cheapest partial solution

Branch & Cut

- Similar to B&B, but relaxes constraints for lower bounds to ease the problem (ILP->LP)
- Iteratively applies Simplex on cut solutions
- But, how to find valid cuts?

Idea: Use heuristics to find near-optimal solutions.



Searching for Optimal Solutions

- Challenges
  - Search space is too big
  - Too many solutions to compute
  - Even good heuristics for a *systematic* search are too costly in terms of performance and memory consumption

- Observation:
  - A sub-/near-optimal solution is usually sufficient
  - It is more important to get a solution in a given time interval

Take Home Message:

- IP problems are inherently harder to solve than LP problems
- Combinatorial problems are optimization problems similar to the IP class with a finite set of solutions

- Exact approaches do not scale for NP-hard problems, so we need heuristics

Next Lecture

- Single-State meta-heuristics for global optimization
  - Hill climbing
  - Random search
  - Simulated annealing
  - Tabu search
  - Iterated local search