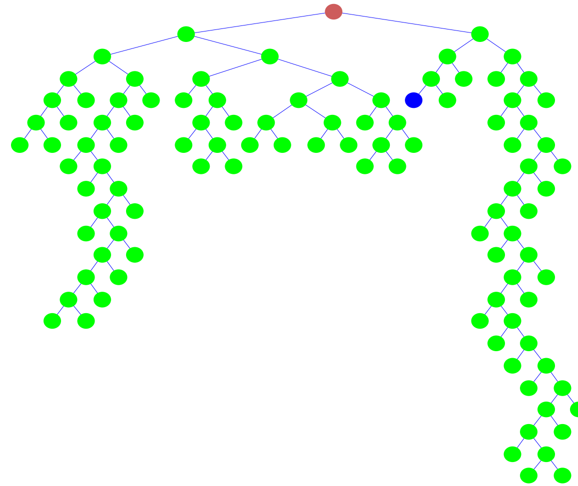
Solving Hard Combinatorial Problems: A Research Overview



Ted Ralphs

Department of Industrial and Systems Engineering
Lehigh University, Bethlehem, PA

http://www.lehigh.edu/~tkr2

Outline of Talk

- Introduction to mathematical models
- Overview of discrete and combinatorial optimization
- Methods for solving discrete optimization problems
- Current research

Mathematical Programming Models

- What does *mathematical programming* mean?
- Programming here means "planning."
- Literally, these are "mathematical models for planning."
- Also called *optimization models*.
- Essential elements
  - Decision variables
  - Constraints
  - Objective Function
  - Parameters and Data

Forming a Mathematical Programming Model

The general form of a *mathematical programming model* is:

$$
\begin{aligned}
&min\ f(x) \\
&s.t.\ \ g_i(x) \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i \\
&\qquad\qquad x \quad \in \quad X
\end{aligned}
$$

$X \subseteq \mathbb{R}^n$ is an (implicitly defined) set that may be discrete.

Types of Mathematical Programs

- The type of a math program is determined primarily by
  - The form of the objective and the constraints.
  - The discrete set X.
- In this talk, we will consider linear models.
  - The objective function is linear.
  - The constraints are linear.
  - Linear programs are specified by a cost vector $c \in \mathbb{R}^n$ a constraint matrix $A \in \mathbb{R}^{m \times n}$ and a right-hand side vector $b \in \mathbb{R}^m$ and have the form

$$\begin{aligned} & min\, c^T x \\ & s.t. \quad Ax \geq b \end{aligned}$$

Solving Linear Models

- We can solve linear models efficiently if $X = \mathbb{R}^n$ (these are called *linear programs*).
- In many situations, X is discrete or semi-discrete—this makes the model much harder to solve.
- These models are called *integer linear programs* (ILPs) or *mixed integer linear programs* (MILPs).
- ILPs can be extremely difficult to solve in practice.
- The simplest form of ILP is a *combinatorial optimization problem* (COP).
- In a COP, the decisions are yes/no.

Combinatorial Optimization

- A *combinatorial optimization problem* $CP = (E, \mathcal{F})$ consists of
  - A *ground set* E,
  - A set $\mathcal{F} \subseteq 2^E$ of *feasible solutions*, and
  - A *cost function* $c \in \mathbb{Z}^E$ (optional).
- The *cost* of $S \in \mathcal{F}$ is $c(S) = \sum_{e \in S} c_e$.
- A *subproblem* is defined by $\mathcal{S} \subseteq \mathcal{F}$.
- Problem: Find a least cost member of $\mathcal{F}$.

Example: Perfect Matching Problem

- We are given a set of n people that need to paired in teams of two.
- Let c_{ij} represent the "cost" of the team formed by persons i and j.
- We wish to minimize the overall cost of the pairings.
- We can represent this problem on an *undirected graph* $G = (N, E)$.
- The nodes represent the people and the edges represent pairings.
- We have $x_e = 1$ if the endpoints of e are matched, $x_e = 0$ otherwise.

$$
\begin{aligned}
min \quad & \sum_{e \in E} c_e x_e \\
s.t. \quad & \sum_{\{j | e = \{i,j\} \in E\}} x_e = 1, \qquad \forall i \in N \\
& x_e \in \{0, 1\}, \ \forall e \in E.
\end{aligned}
$$

Fixed-charge Problems

- In many instances, there is a fixed cost and a variable cost associated with a particular decision.
- Example: Fixed-charge Network Flow Problem (FCNFP)
  - We are given a directed graph $G = (N, A)$ and a demand/supply at each node.
  - There is a fixed cost c_{ij} associated with building arc (i, j).
  - There is also a variable cost d_{ij} for each unit of flow along arc (i, j).
  - We want to minimize the sum of these two costs while meeting demand.
- Example: Cable-Trench Problem (CTP)
  - A FCNFP with only one supply node (the communications hub).
  - All other nodes must be connected to the hub by a cable.
  - The fixed cost is the cost of digging the trenches.
  - The variable cost is the cost of laying the cable.

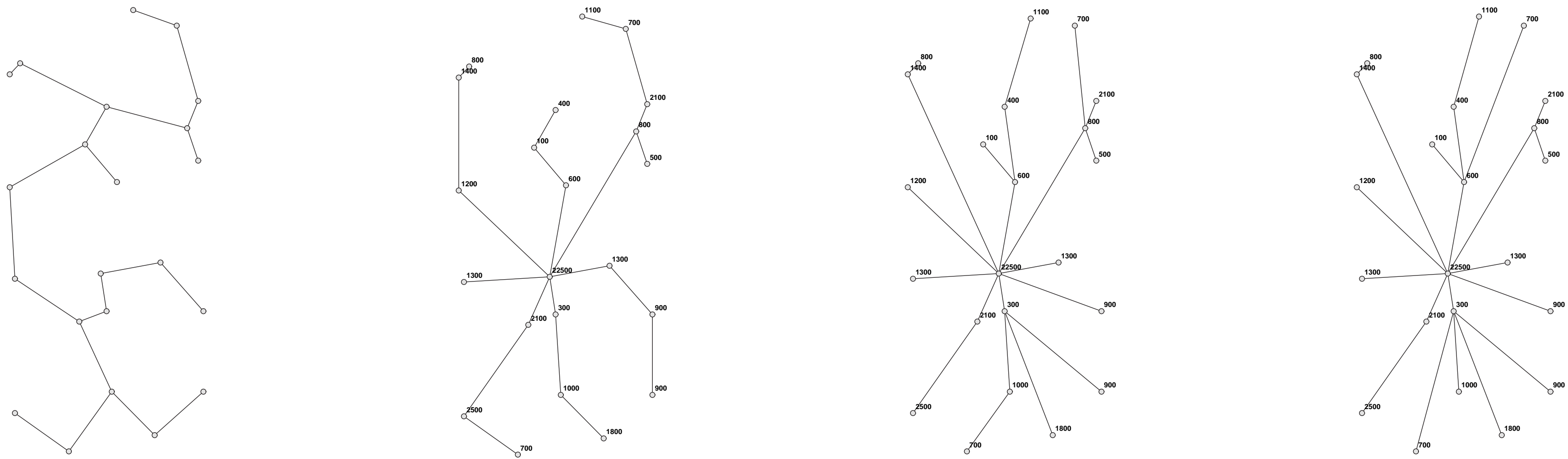


Figure 1: Sample CTP Solutions

Modeling the Fixed-charge Network Flow Problem

- To model the FCNFP, we associate two variables with each arc.
  - x_{ij} (*fixed-charge variable*) indicates whether arc (i, j) is open.
  - f_{ij} (*flow variable*) represents the flow on arc (i, j).
  - Note that we have to ensure that $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

$$
\begin{aligned}
min \sum_{(i,j)\in A} & c_{ij}x_{ij} + d_{ij}f_{ij} \\
s.t. \sum_{j\in O(i)} f_{ij} - \sum_{j\in I(i)} & f_{ji} = b_i \quad \forall i \in N \\
& f_{ij} \leq Cx_{ij} \quad \forall (i,j) \in A \\
& f_{ij} \geq 0 \quad \forall (i,j) \in A \\
& x_{ij} \in \{0,1\} \quad \forall (i,j) \in A
\end{aligned}
$$

Facility Location Problem

- We are given n potential facility locations and m customers that must be serviced from those locations.
- There is a fixed cost c_j of opening facility j.
- There is a cost d_{ij} associated with serving customer i from facility j.
- We have two sets of binary variables.
  - y_j is 1 if facility j is opened, 0 otherwise.
  - x_{ij} is 1 if customer i is served by facility j, 0 otherwise.

$$
\begin{aligned}
min \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\
s.t. \ \sum_{j=1}^{n} x_{ij} = K & \qquad \forall i \\
x_{ij} \leq y_j & \qquad \forall i, j \\
x_{ij}, y_j \in \{0, 1\} & \qquad \forall i, j
\end{aligned}
$$

Traveling Salesman Problem

- We are given a set of cities and a cost associated with traveling between each pair of cities.
- We want to find the least cost route traveling through every city and ending up back ate the starting city.
- Applications of the Traveling Salesman Problem
  - Drilling Circuit Boards
  - Gene Sequencing

Formulating The Traveling Salesman Problem

The TSP is a combinatorial problem $(E, \mathcal{F})$ whose ground set is the edge set of a graph $G = (V, E)$.

- V is the set of customers.
- E is the set of travel links between the customers.

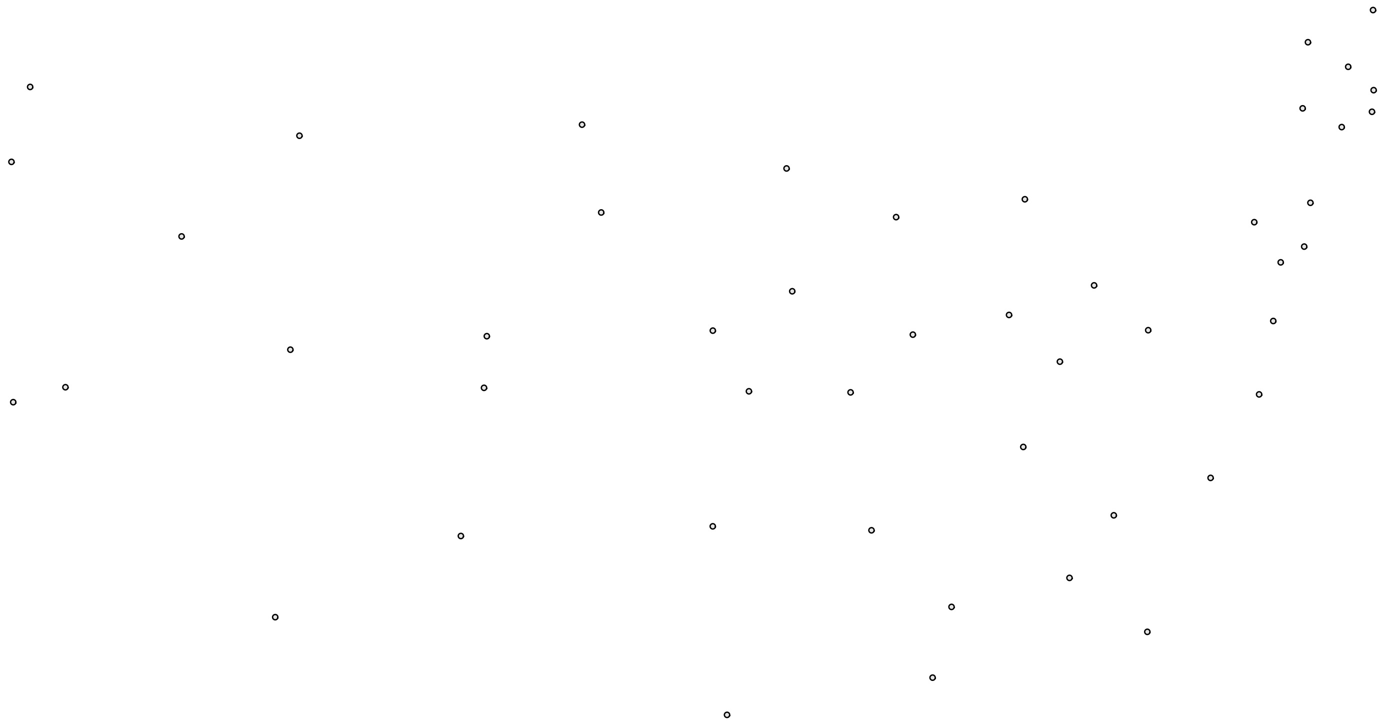
A feasible solution is a permutation σ of V specifying the order of the customers.

ILP Formulation:

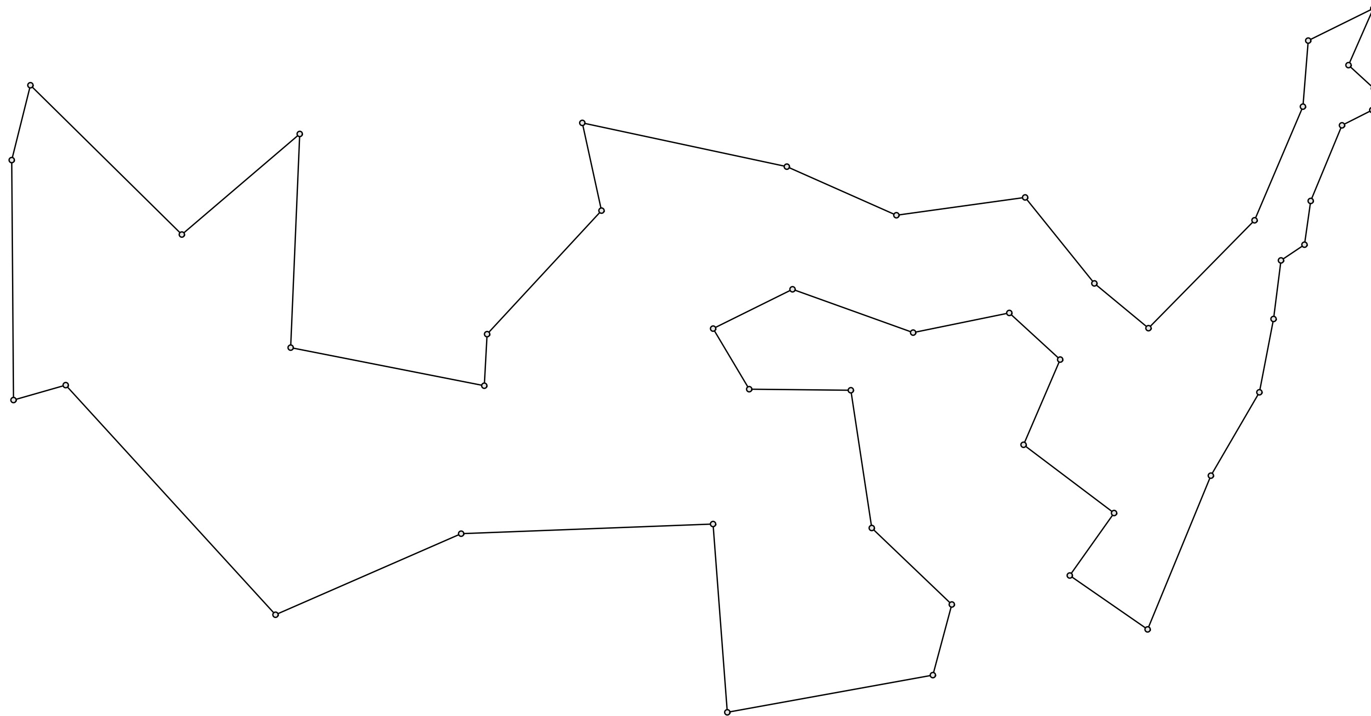
$$\begin{array}{rcll} \sum_{j=1}^{n} x_{ij} & = & 2 & \forall i \in N^{-} \\ \sum_{\substack{i \in S \\ j \notin S}} x_{ij} & \geq & 2 & \forall S \subset V,\ |S| > 1. \end{array}$$

where x_{ij} is a binary variable indicating $\sigma(i) = j$.

Example Instance of the TSP



Optimal Solution to the 48 City Problem



How Hard Are These Problems?

- In practice, they can be extremely difficult.
- The number of possible solutions for the TSP is $n!$ (that's HUGE).
- We cannot afford to enumerate all these possibilities.
- But there is no direct, efficient way to solve these problems.

How do we solve these hard problems?

- Try to reduce them to something easier
  - Integer Linear Program $\Rightarrow$ Linear Program
  - Divide and conquer
- Use a bigger hammer
  - Faster processors
  - More memory
  - Parallelism

Branch and Bound

- Suppose F is the feasible region for some MILP and we wish to solve $\min_{x \in F} c^T x$.
- Consider a partition of F into subsets $F_1, \ldots F_k$. Then

$$\min_{x \in F} c^T x = \min_{1 \leq i \leq k} \{\min_{x \in F_i} c^T x\}$$

.

- In other words, we can optimize over each subset separately.
- Idea: If we can't solve the original problem directly, we might be able to solve the smaller *subproblems* recursively.
- Dividing the original problem into subproblems is called *branching*.
- Taken to the extreme, this scheme is equivalent to complete enumeration.

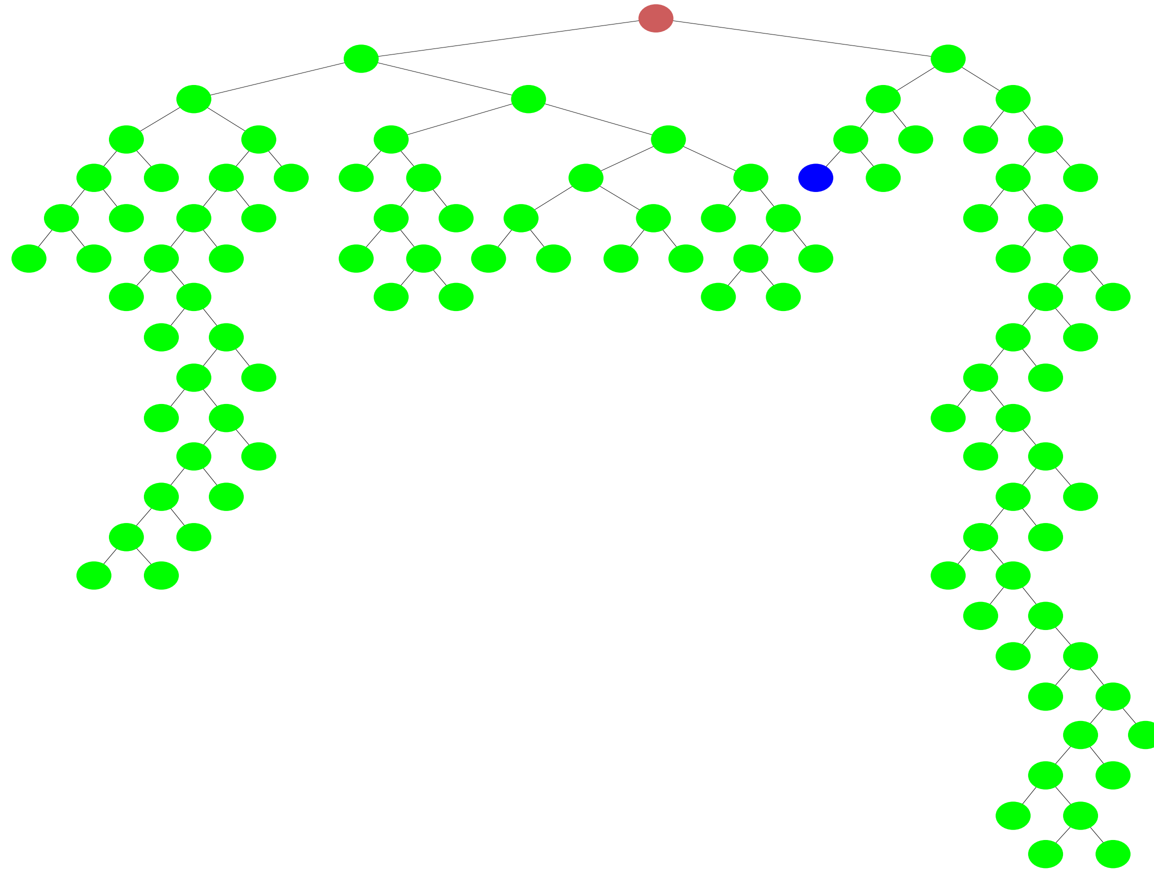
Relaxations

- A *relaxation* of an ILP is an auxiliary mathematical program for which
  - the feasible region contains the feasible region for the original ILP, and
  - the objective function value of each solution to the original ILP is not increased.
- In branch and bound, we typically first solve a relaxation of the original problem to obtain a *bound*.
- The result is one of the following:
  1. The relaxation is infeasible $\Rightarrow$ MILP is infeasible.
  2. We obtain a feasible solution for the MILP $\Rightarrow$ optimal solution.
  3. We obtain an optimal solution to the relaxation that is not feasible for the MILP $\Rightarrow$ lower bound.
- In the first two cases, we are finished.
- In the third case, we must branch and recursively solve the resulting subproblems.

Continuing the Algorithm After Branching

- After branching, we solve each of the subproblems recursively.
- Now we have an additional factor to consider.
- If the optimal value of the relaxation is greater than the current upper bound, we can *prune* the subproblem.
- This is the key to the efficiency of the algorithm.

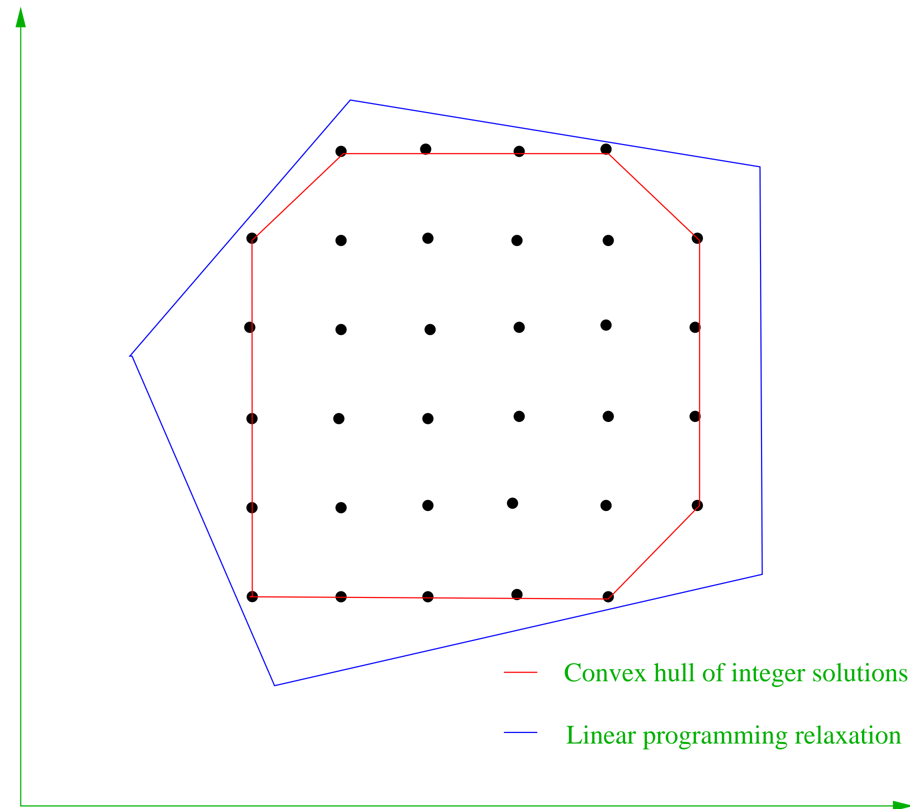
Branch and Bound Tree



Types of Relaxations

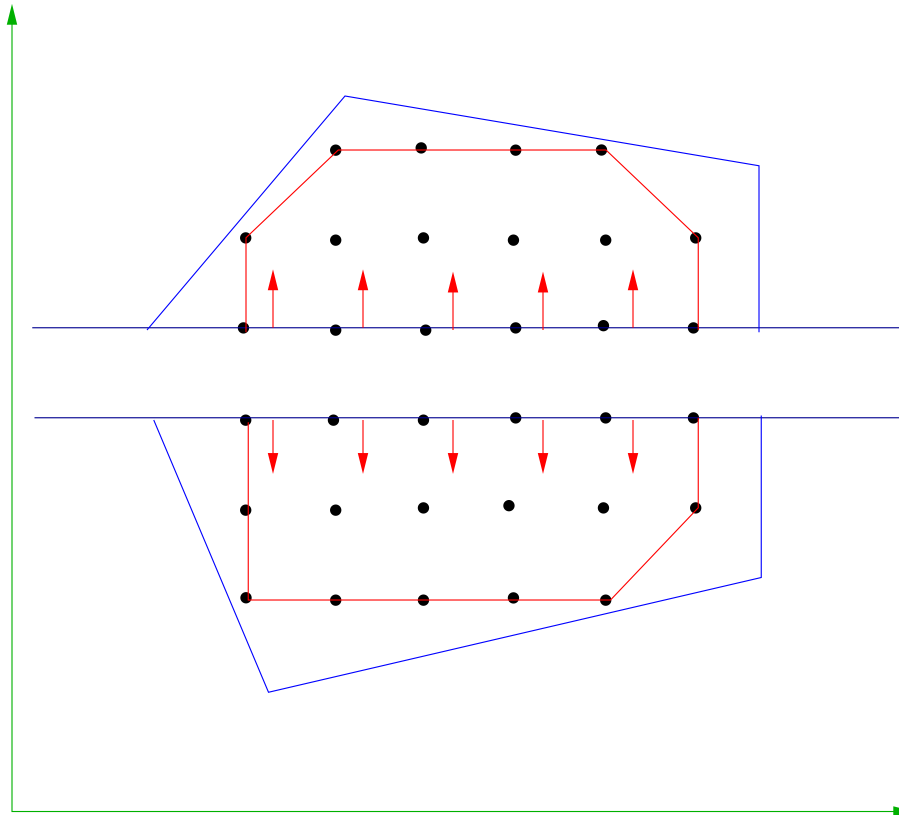
- *Linear programming* relaxations
  - Obtained by dropping the integrality constraints.
  - Easy to solve.
  - Bounds weak in general.
- *Lagrangian* relaxations.
  - Obtained by dropping some of the linear constraints.
  - Violation of these constraints is then penalized in the objective function.
  - Bound strength depends on what constraints are dropped.

Improving the Bound with Cutting Planes

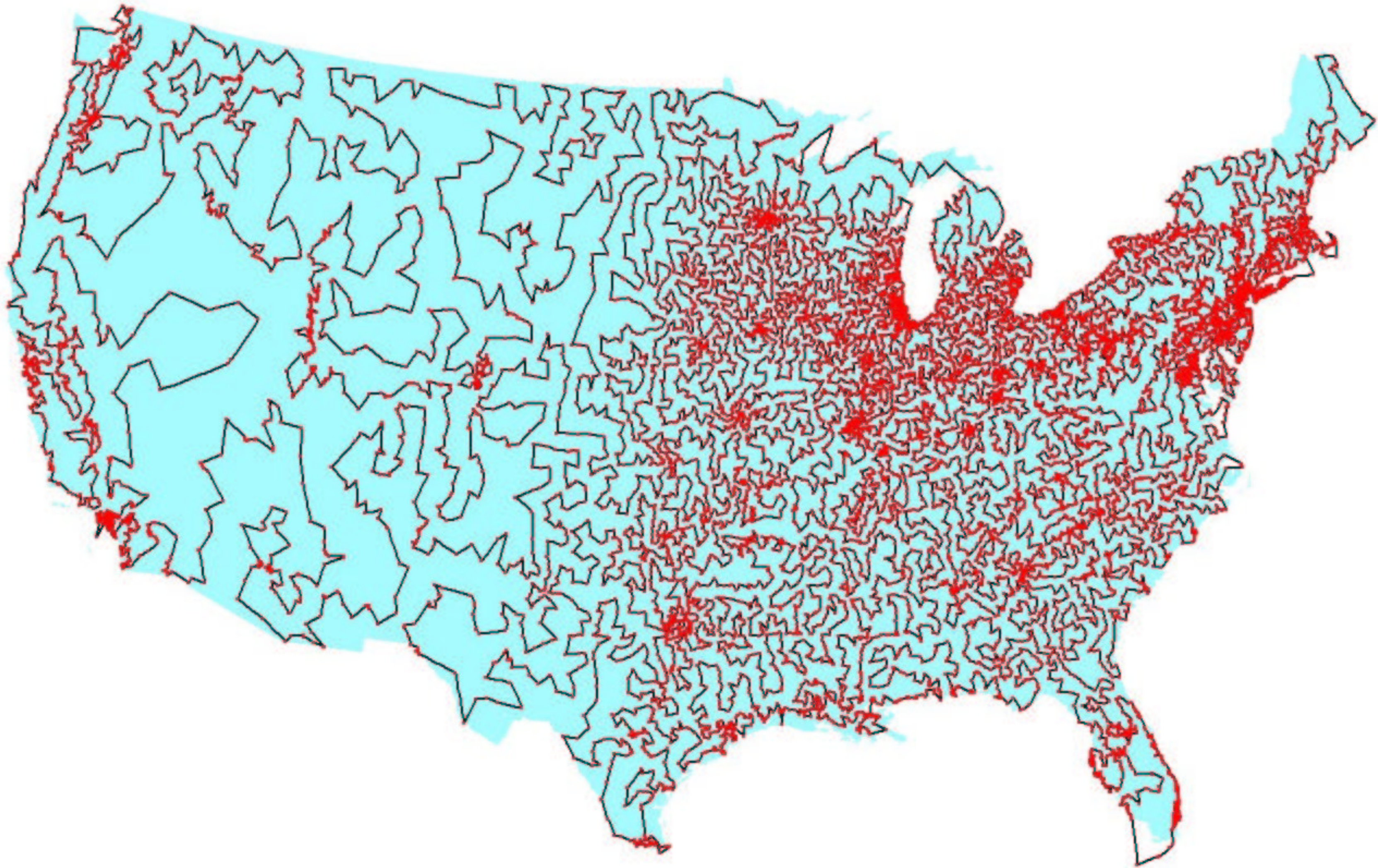


Branch and Cut Methods

If the cutting plane approach fails, then we divide and conquer (branch).



Current State of the Art



Current Research

- Theory and Methodology
  - Branch, cut, and price algorithms for large-scale discrete optimization
  - Decomposition-based algorithms for discrete optimization
  - Parallel algorithms
- Software Development
  - COIN-OR Project (Open Source Software for Operations Research)
  - SYMPHONY (C library for parallel branch, cut, and price)
  - ALPS (C++ library for scalable parallel search algorithms)
  - BiCePS (C++ library for parallel branch, constrain, and price)
  - BLIS (C++ library built for solving MILPs)
  - DECOMP (Framework for decomposition-based algorithms)
- Applications
  - Logistics (routing and packing problems)
  - High tech manufacturing
  - Computational biology