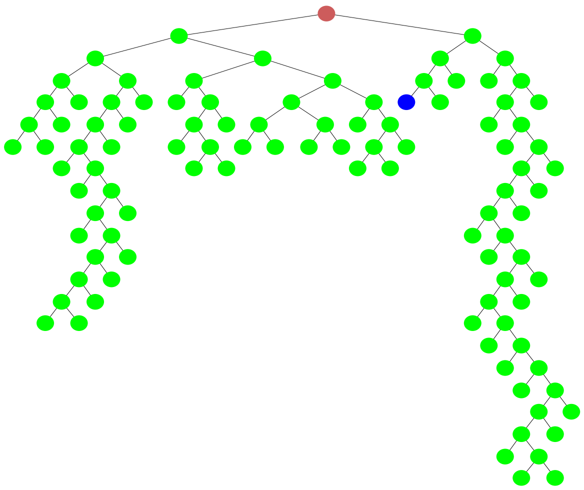
Solving Hard Combinatorial Problems: A Research Overview



Ted Ralphs

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

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

Outline of Talk

- Introduction to *mathematical programming*
- Examples of discrete optimization problems
- Methods for discrete optimization
- Current research

What is a model?

mod·el: *A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics.*

–American Heritage Dictionary of the English Language

- Two types of models
  - Concrete
  - Abstract
- Mathematical models
  - Are abstract models
  - Describe the mathematical relationships among elements in a system.

Why do we model systems?

- The exercise of building a model can provide insight.
- It's possible to do things with models that we can't do with "the real thing."
- Analyzing models can help us decide on a course of action.

Examples of Model Types

- Simulation Models
- Probability Models
- Financial Models
- Mathematical Programming Models

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 *math programming model* is:

$$\begin{aligned} &\min \text{ or } \max && f(x_1, \ldots, x_n) \\ &s.t. && g_i(x_1, \ldots, x_n) \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i \end{aligned}$$

We might also require the values of the variables to belong to a discrete set X.

Solutions

- A *solution* is an assignment of values to variables.
- A solution can be thought of as a vector.
- A *feasible solution* is an assignment of values to variables such that all the constraints are satisfied.
- The *objective function value* of a solution is obtained by evaluating the objective function at the given solution.
- An *optimal solution* (assuming minimization) is one whose objective function value is less than or equal to that of all other feasible solutions.

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.
  - Whether the input data is considered "known".
- In this talk, we will consider linear programs.
  - The objective function is linear.
  - The constraints are linear.
  - Linear programs are specified by a cost vector $c \in \mathbf{R}^n$ a constraint matrix $A \in \mathbf{R}^{m \times n}$ and a right-hand side vector $b \in \mathbf{R}^m$ and have the form

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

Two Crude Petroleum Example

- Two Crude Petroleum distills crude from two sources:
  - Saudi Arabia
  - Venezuela
- They have three main products:
  - Gasoline
  - Jet fuel
  - Lubricants
- Yields

	Gasoline	Jet fuel	Lubricants
Saudi Arabia	0.3 barrels	0.4 barrels	0.2 barrels
Venezuela	0.4 barrels	0.2 barrels	0.3 barrels

Two Crude Petroleum Example (cont.)

- Availability and cost

	Availability	Cost
Saudi Arabia	9000 barrels	$20/barrel
Venezuela	6000 barrels	$15/barrel

- Production Requirements (per day)

Gasoline	Jet fuel	Lubricants
2000 barrels	1500 barrels	500 barrels

- Objective: Minimize production cost.

Modeling the Two Crude Production Problem

- What are the decision variables?
- What is the objective function?
- What are the constraints?

Modeling the Two Crude Production Problem

- The *decision variables* are the amount of each type of crude to refine.

  $x_1 =$ thousands of barrels of Saudi crude refined per day.
  $x_2 =$ thousands of barrels of Venezuelan crude refined per day.

- What is the objective function?

- What are the constraints?

Modeling the Two Crude Production Problem

- The decision variables are the amount of each type of crude to refine.

  $x_1 =$ thousands of barrels of Saudi crude refined per day.
  $x_2 =$ thousands of barrels of Venezuelan crude refined per day.

- The *objective function* is $20x_1 + 15x_2$.

- What are the constraints?

Modeling the Two Crude Production Problem

- The decision variables are the amount of each type of crude to refine.

  $x_1 =$ thousands of barrels of Saudi crude refined per day.
  $x_2 =$ thousands of barrels of Venezuelan crude refined per day.

- The objective function is $20x_1 + 15x_2$.

- In words, the *production constraints* are

$$\sum(\text{yield per barrel})(\text{barrels refined}) \geq \text{production requirements}$$

- In addition, we have *bounds* on the variables.

Linear Programming Formulation of Two Crude Example

- This yields the following LP formulation:

$$
\begin{aligned}
\min \quad & 20x_1 + 15x_2 \\
\text{s.t.} \quad & 0.3x_1 + 0.4x_2 \geq 2.0 \\
& 0.4x_1 + 0.2x_2 \geq 1.5 \\
& 0.2x_1 + 0.3x_2 \geq 0.5 \\
& 0 \leq x_1 \leq 9 \\
& 0 \leq x_2 \leq 6
\end{aligned}
$$

Solving Linear Programs

- Generally speaking, we can solve linear programs efficiently.
- However, in many situations, the variables must take on discrete values, usually integral values.
- These programs are called *integer programs* and are an example of a discrete optimization problem.
- Integer programs can be extremely difficult to solve in practice.
- The simplest form of integer programming is *combinatorial optimization*.
- In a combinatorial problem, all 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* $c \in \mathbf{Z}^\mathbf{E}$ (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 person i and person 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.

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

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
  - We are given a directed graph $G = (N, A)$.
  - There is a fixed cost c_{ij} associated with "opening" arc (i, j) (think of this as the cost to "build" the link).
  - There is also a variable cost d_{ij} associated with each unit of flow along arc (i, j).
  - Think of the fixed charge as the construction cost and the variable charge as the operating cost.
  - We want to minimize the sum of these two costs.

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 $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

$$
\begin{aligned}
Min \sum_{(i,j)\in A} & c_{ij}x_{ij} + d_{ij}f_{ij} \\
s.t. \quad \sum_{j\in O(i)} f_{ij} - \sum_{j\in I(i)} & f_{ji} = b_i \qquad \forall i \in N \\
& f_{ij} \le C x_{ij} \quad \forall (i,j) \in A \\
& f_{ij} \ge 0 \qquad\;\; \forall (i,j) \in A \\
& x_{ij} \in \{0,1\} \; \forall (i,j) \in A
\end{aligned}
$$

Example: 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.

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

The 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 $G = (V, E)$.

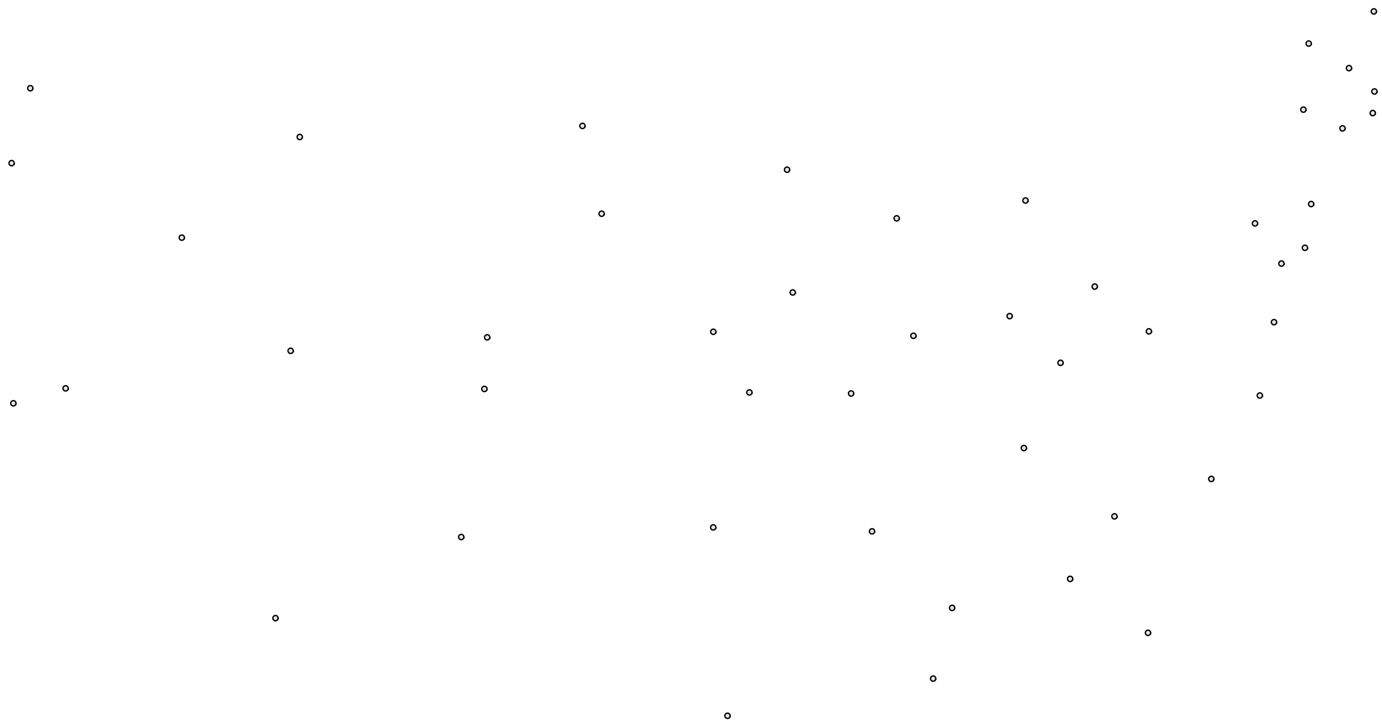
- V is the set of customers.
- E is the set of travel links between the customers.

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

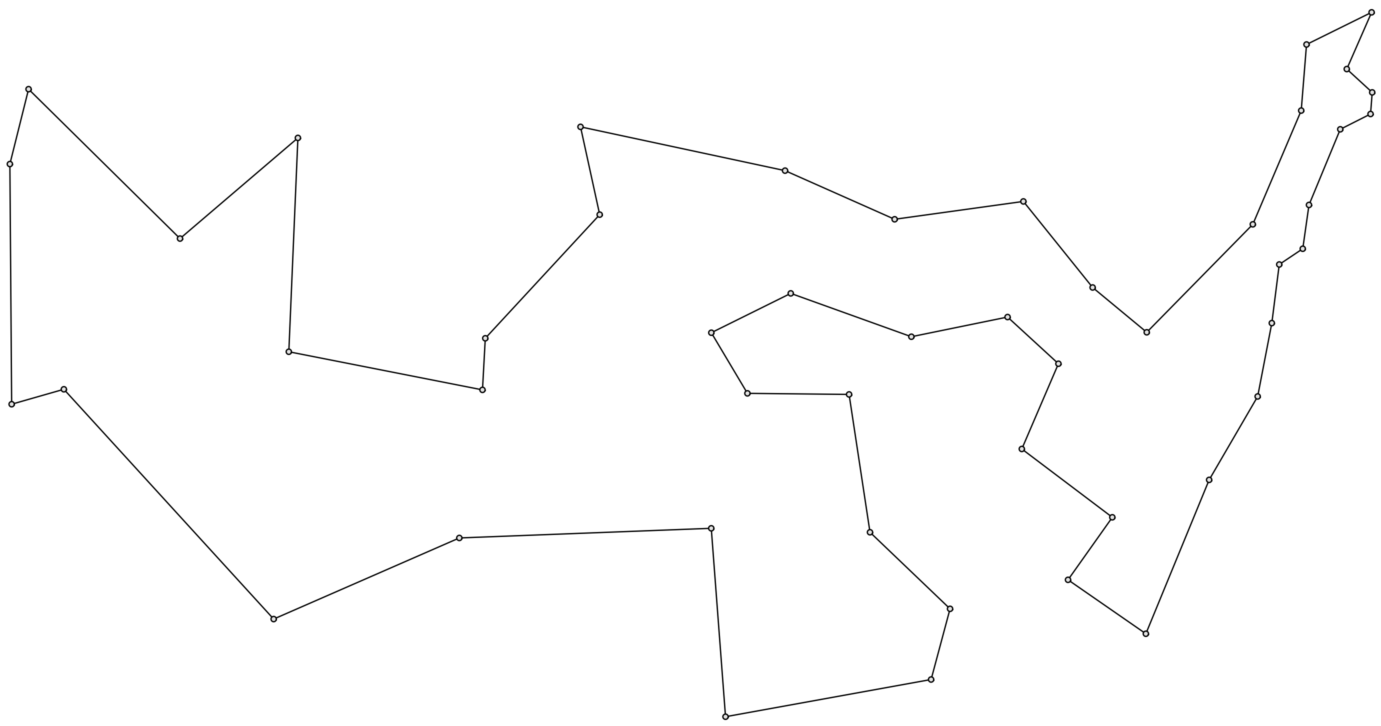
$$\begin{array}{rcl} \sum_{j=1}^{n} x_{ij} & = & 2 \quad \forall i \in N^{-} \\ \sum_{\substack{i \in S \\ j \notin S}} x_{ij} & \geq & 2 \quad \forall S \subset V,\ |S| > 1. \end{array}$$

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

Example Instance of the TSP



Optimal Solutions to the 48 City Problem



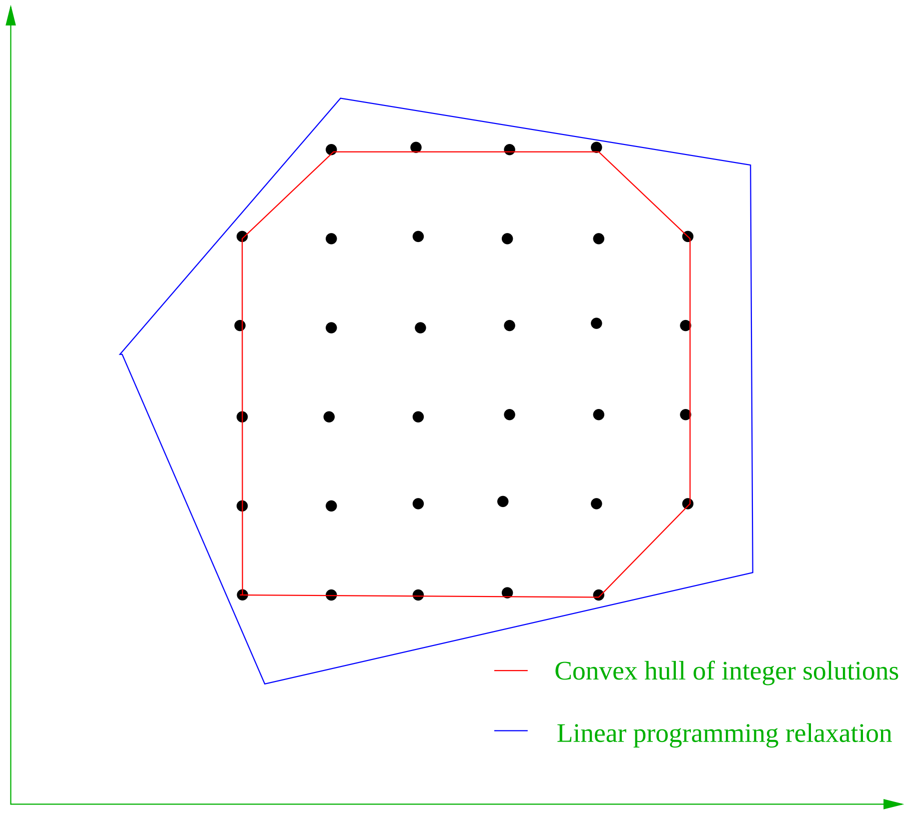
How hard are these problems?

- In practice, these can be extremely difficult.
- The number of possible solutions for the TSP is $n!$ where n is the number of cities.
- 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?

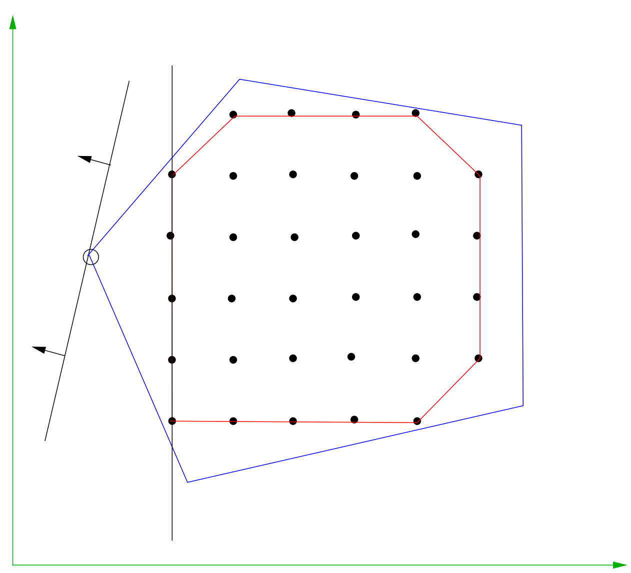
- Try to reduce them to something easier
  - Integer Program $\Rightarrow$ Linear Program
  - Divide and conquer
- Use a bigger hammer
  - Faster processors
  - More memory
  - Parallelism

Integer Programming



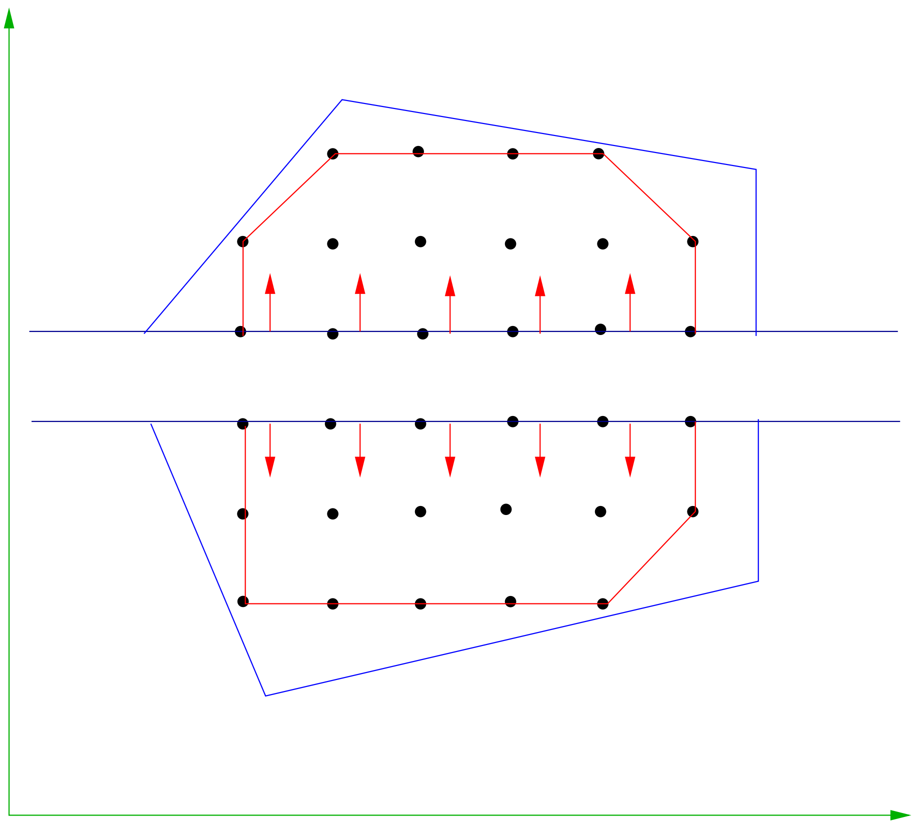
Cutting Plane Method

- Basic cutting plane algorithm
  - Relax the integrality constraints.
  - Solve the relaxation. Infeasible $\Rightarrow$ STOP.
  - If $\hat{x}$ integral $\Rightarrow$ STOP.
  - Separate $\hat{x}$ from $\mathcal{P}$.
  - No cutting planes $\Rightarrow$ algorithm fails.
- The key is good separation algorithms.



Branch and Cut Methods

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



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.

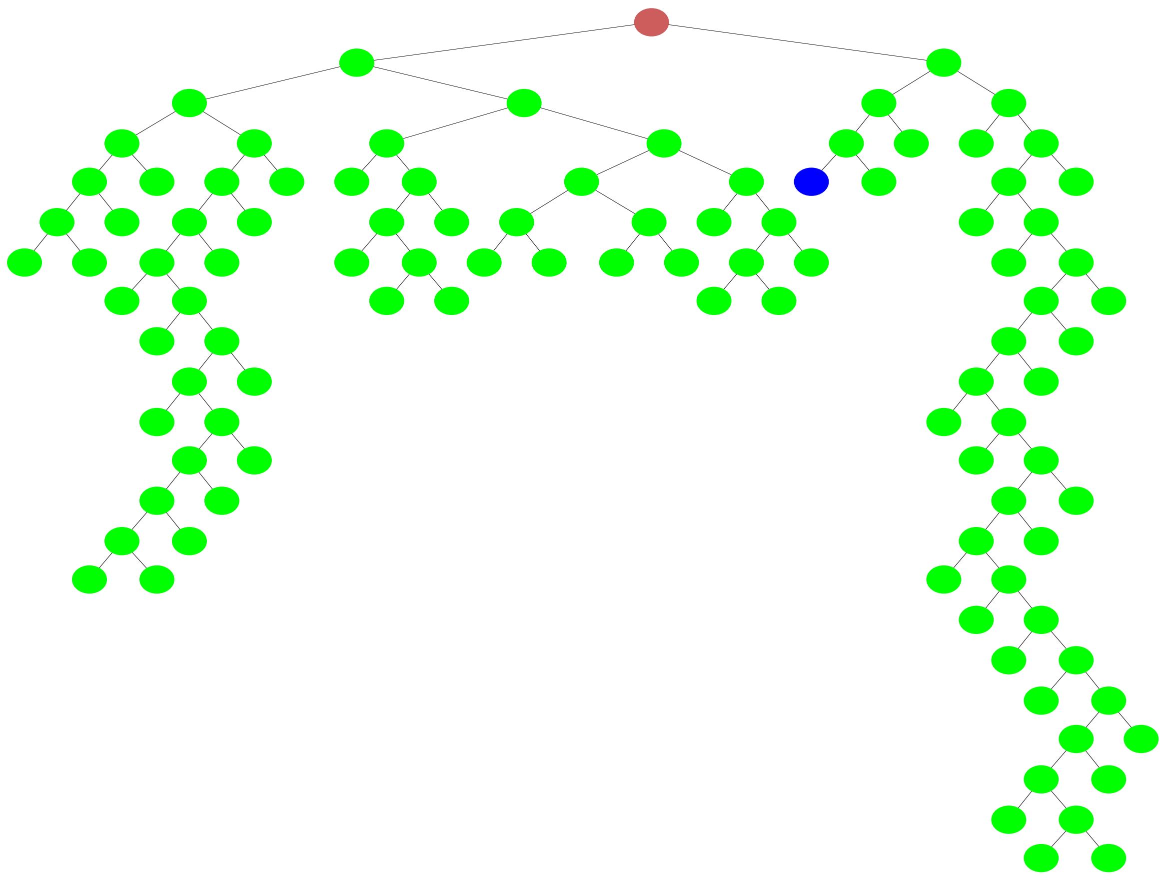
LP-based Branch and Bound

- In LP-based branch and bound, we first solve the LP relaxation of the original problem. The result is one of the following:
  1. The LP 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 LP that is not feasible for the MILP $\Rightarrow$ upper bound.
- In the first two cases, we are finished.
- In the third case, we must branch and recursively solve the resulting subproblems.

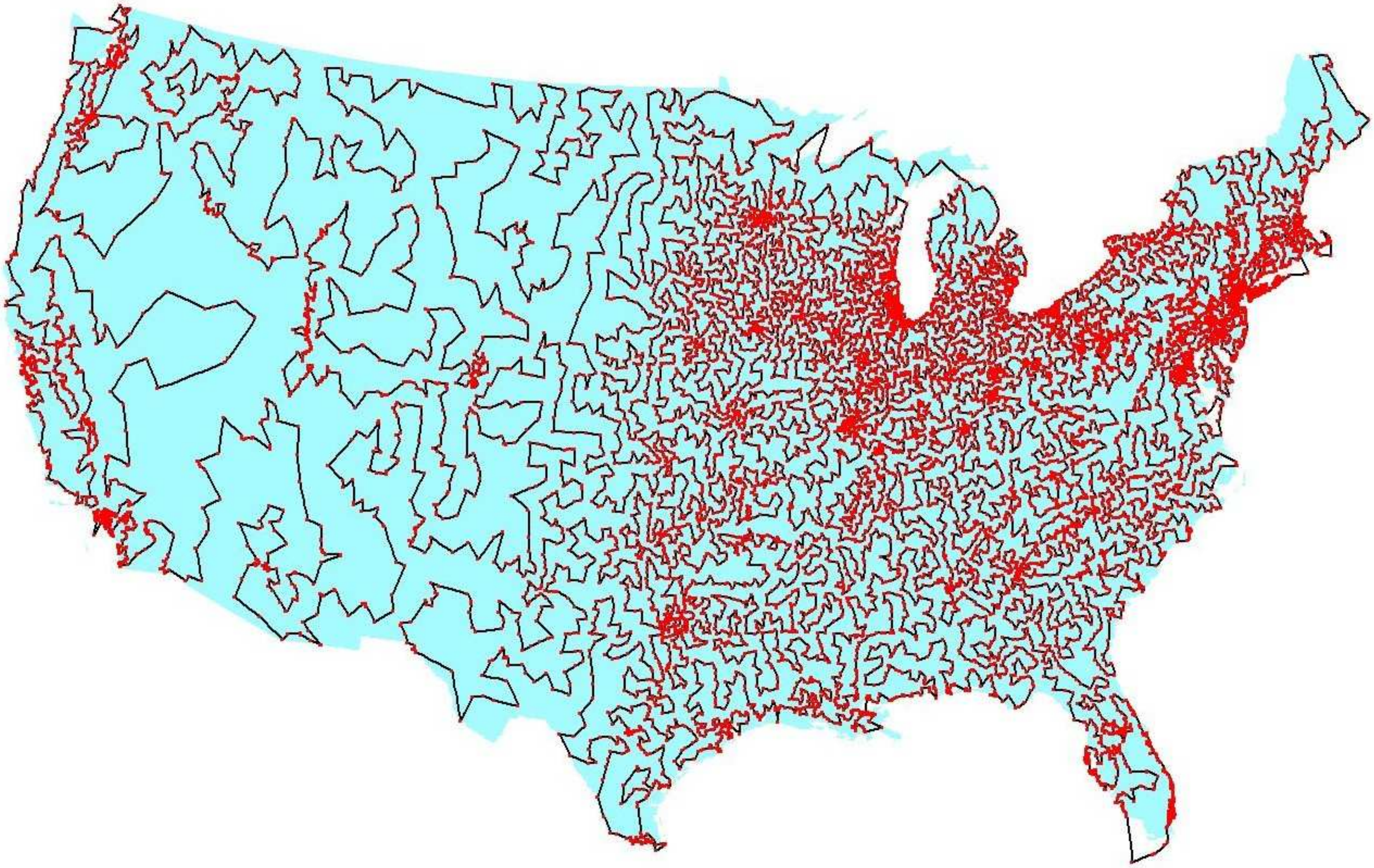
Continuing the Algorithm After Branching

- After branching, we solve each of the subproblems recursively.
- Now we have an additional factor to consider.
- If the optimal solution value to the LP relaxation is greater than the current upper bound, we need not consider the subproblem further.
- This is the key to the efficiency of the algorithm.
- Terminology
  - If we picture the subproblems graphically, they form a *search tree*.
  - Each subproblem is linked to its *parent* and eventually to its *children*.
  - Eliminating a problem from further consideration is called *pruning*.
  - The act of bounding and then branching is called *processing*.
  - A subproblem that has not yet been considered is called a *candidate* for processing.
  - The set of candidates for processing is called the *candidate list*.

Branch and Bound Tree



Current State of the Art



Current Research

- The theory of integer programming is fairly well developed.
- Computationally, however, these methods are very difficult to implement effectively.
- Also, each problem requires *different methods of separation*.
- It is therefore extremely expensive to implement an efficient solver for a new application.
- Overcoming these challenges and developing generic solvers capable of automatically analyzing problem structure and performing separation is an active area of research.
- These methods also depend heavily on our ability to obtain good bounds.
- Finding new/better methods of bounding is another active research area.
- I am currently involved in research in both of these areas.