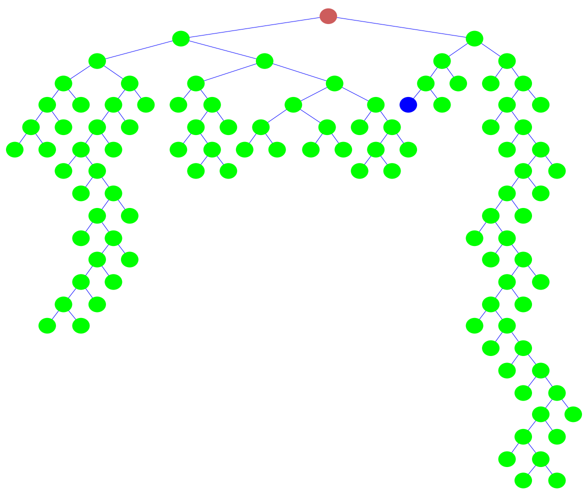
Integer Programming: A Research Overview



Ted Ralphs

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

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

Outline of Talk

- Introduction
- Applications
- Research
  - Solution methodology
  - User interfaces
  - Computation
- Conclusions

Introduction

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:

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

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

- A *mathematical programming problem* is a problem that can be expressed using a mathematical programming model (called the *formulation*).

- A single mathematical programming problem can be represented using many different formulations (important).

Types of Mathematical Programming Models

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

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

Linear Models

- Generally speaking, linear models are easier to solve than more general types of models.
- If $X = \mathbb{R}^n$, the model is called a *linear program* (LP).
- Linear programming models can be solved effectively.
- If some of the variables in the model are required to take on integer values, the model is called a *mixed integer linear programs* (MILPs).
- MILPs can be extremely difficult to solve in practice.

Modeling with Integer Variables

- Why do we need integer variables?
- If a variable represents the quantity of a physical resource that only comes in discrete units, then it must be assigned an integer value.
  - Product mix problem.
  - Cutting stock problem.
- We can use *0-1 (binary) variables* for a variety of purposes.
  - Modeling yes/no decisions.
  - Enforcing disjunctions.
  - Enforcing logical conditions.
  - Modeling fixed costs.
  - Modeling piecewise linear functions.
- The simplest form of ILP is a *combinatorial optimization problem* (COP), where all variables are binary.

Example: Perfect Matching Problem

- We are given a set N 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.
- The nodes represent the people and the edges represent pairings.
- We have $x_{ij} = 1$ if i and j are matched, $x_{ij} = 0$ otherwise.
- To simplify the presentation, we assume that $x_{ij} = 0$ if $i \geq j$.

$$
\begin{aligned}
min \quad & \sum_{\{i,j\} \in N \times N} c_{ij} x_{ij} \\
s.t. \quad & \sum_{j \in N} x_{ij} = 1, \qquad \forall i \in N \\
& x_{ij} \in \{0, 1\}, \; \forall \{i, j\} \in N \times N, i < j.
\end{aligned}
$$

Applications

Applications

To get a feel, we w'll sample applications from a few "hot" areas.

- Supply Chain Logistics
- Computational Biology
- Electronic Commerce

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.

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

Traveling Salesman Problem

- We are given a set of cities and a cost associated with traveling between each pair of cities.
- The *Traveling Salesman Problem* (TSP) is that of finding the least cost route traveling through every city and ending up back at the starting city.
- Applications of the TSP
  - Drilling Circuit Boards
  - DNA Sequencing

DNA Sequencing and the TSP

- The *DNA sequencing problem* is to find the sequence of base pairs in a large fragment of DNA (length N).
- It is not practical to simply examine the DNA and determine the sequence.
- One approach is *sequencing by hybridization*: mix the DNA sequence with short oligonucleotides that bind with subsequences of the DNA.
- This results in an approximate list of all subsequences of length l that occur in the larger sequence.
- To reconstruct the original sequence, we simply have to correctly order the subsequences.
- This is easy if there are no errors.
- In the presence of errors, it becomes an optimization problem that is a variant of the TSP.

The Set Partitioning Problem

- We are given a finite set S and subsets $S_1, \ldots, S_n$ of S, each with an associated cost $c(S_i), i \in [1..n]$.
- The *Set Partitioning Problem* is to determine $I \subset [1..n]$ such that $\cup_{i \in I} S_i = S$ and $\sum_{i \in I} c(S_i)$ is minimized.
- We can formulating this as an integer program.
  - Construct a $0-1$ matrix A in which $a_{ij} = 1$ if and only if the i^{th} element of S belongs to S_j.
  - The integer program is then

$$
\begin{aligned}
& min\, c^T x \\
& s.t. \quad Ax = 1 \\
& \qquad\quad x \in \{0,1\}^n
\end{aligned}
$$

- These integer programs are very difficult to solve.

The Set Partitioning Problem and Combinatorial Auctions

- A *combinatorial auction* is an auction in which participants are allowed to bid on subsets of the available goods.
- This accounts for the fact that some items have a greater worth when combined with other items.
- Example: FCC bandwidth auction
  - The FCC wishes to sell the licenses by bandwidth and region.
  - The value of a set of bandwidth licenses is increased if they are in contiguous regions.
- A set of items along with a price constitutes a *bid*.
- Given a set of bids, determining the winners of the auction is a set partitioning problem.

Research

Show Me the Research

- Methodology
  - Solution methodology
    - Primal algorithms
    - Implicit enumeration algorithms
    - Heuristics
  - Other tools
    - Automatic reformulation
    - Decomposition
    - Preprocessing techniques
    - Generation of strong valid inequalities
    - Primal heuristics
- User interfaces
  - Modeling languages
  - Data interchange formats
  - Callable libraries
  - Algorithmic frameworks
- Computation

Methodology

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 efficient direct method for solving these problems.

Primal Algorithms

- Many optimization algorithms use an improving search paradigm.
  - Find a feasible starting solution.
  - In each iteration, determine an *improving feasible direction* and move in that direction to get to a better solution.
  - The step size is determined by performing a line search.
- Can this be made to work for integer programming?
- What are the difficulties?

Implicit Enumeration

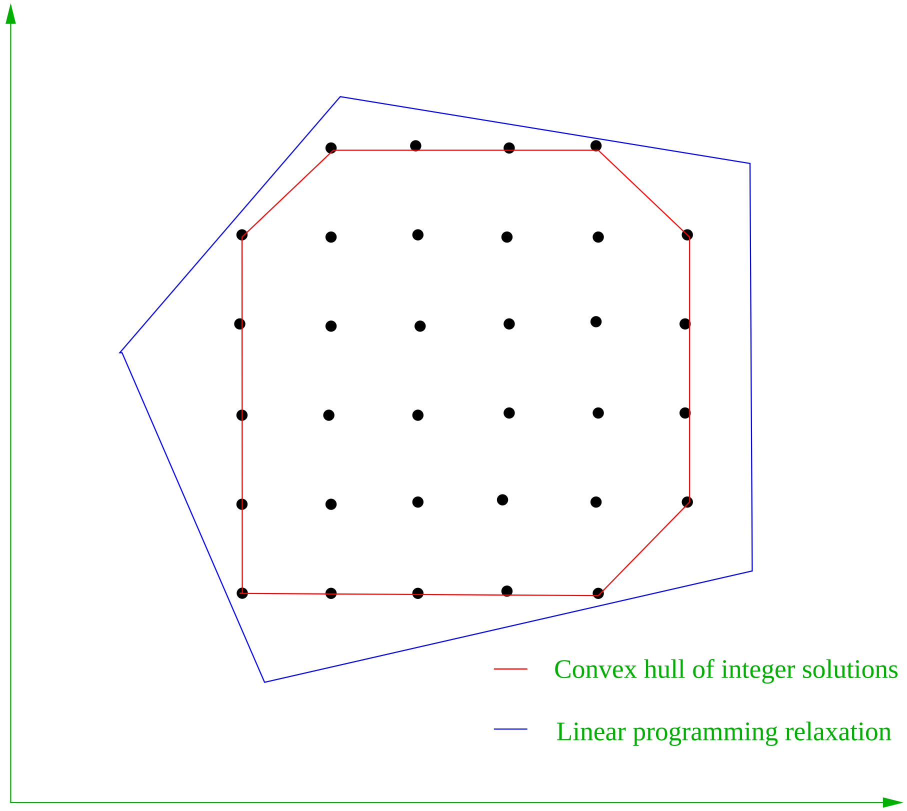
- *Implicit enumeration* techniques try to enumerate the solution space in an intelligent way.
- The most common algorithm of this type is *branch and bound*.
- Suppose F is the set of feasible solutions for some MILP and we wish to solve $\min_{x \in F} c^T x$.
- Consider a partition of F into subsets $F_1, \ldots F_k$. Then

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

.

- 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.
- We avoid complete enumeration primarily by deriving bounds on the value of an optimal solution to each subproblem.

The Geometry of Integer Programming



Bounding

- 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.
- Types of Relaxations
  - Continuous relaxations
    * Most common continuous relaxation is the *LP relaxation*.
    * Obtained by dropping some or all of the integrality constraints.
    * Easy to solve.
    * Bounds weak in general.
    * Other relaxations are possible using semi-definite programming, for instance.
  - Combinatorial relaxations
    * Obtained by dropping some of the linear constraints.
    * Violation of these constraints can then penalized in the objective function (*Lagrangian relaxation*)
    * Bound strength depends on what constraints are dropped.

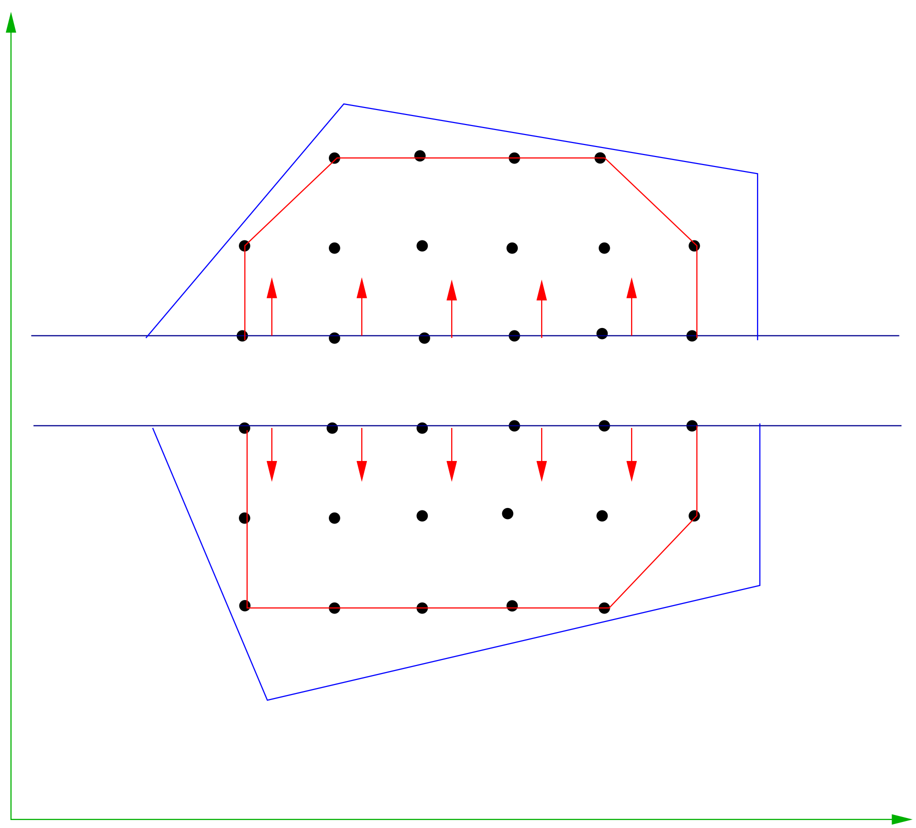
Branch and Bound Algorithm

- We maintain a queue of *active* subproblems initially containing just the *root subproblem*.
- We choose a subproblem from the queue and solve a relaxation of it to obtain a *bound*.
- The result is one of the following:
  1. The relaxation is infeasible $\Rightarrow$ subproblem is infeasible.
  2. We obtain a feasible solution for the MILP $\Rightarrow$ subproblem solved (new upper bound??).
  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 compare the lower bound to the global upper bound.
  - If it exceeds the upper bound, we discard the subproblem.
  - If not, we branch and add the resulting subproblems to the queue.

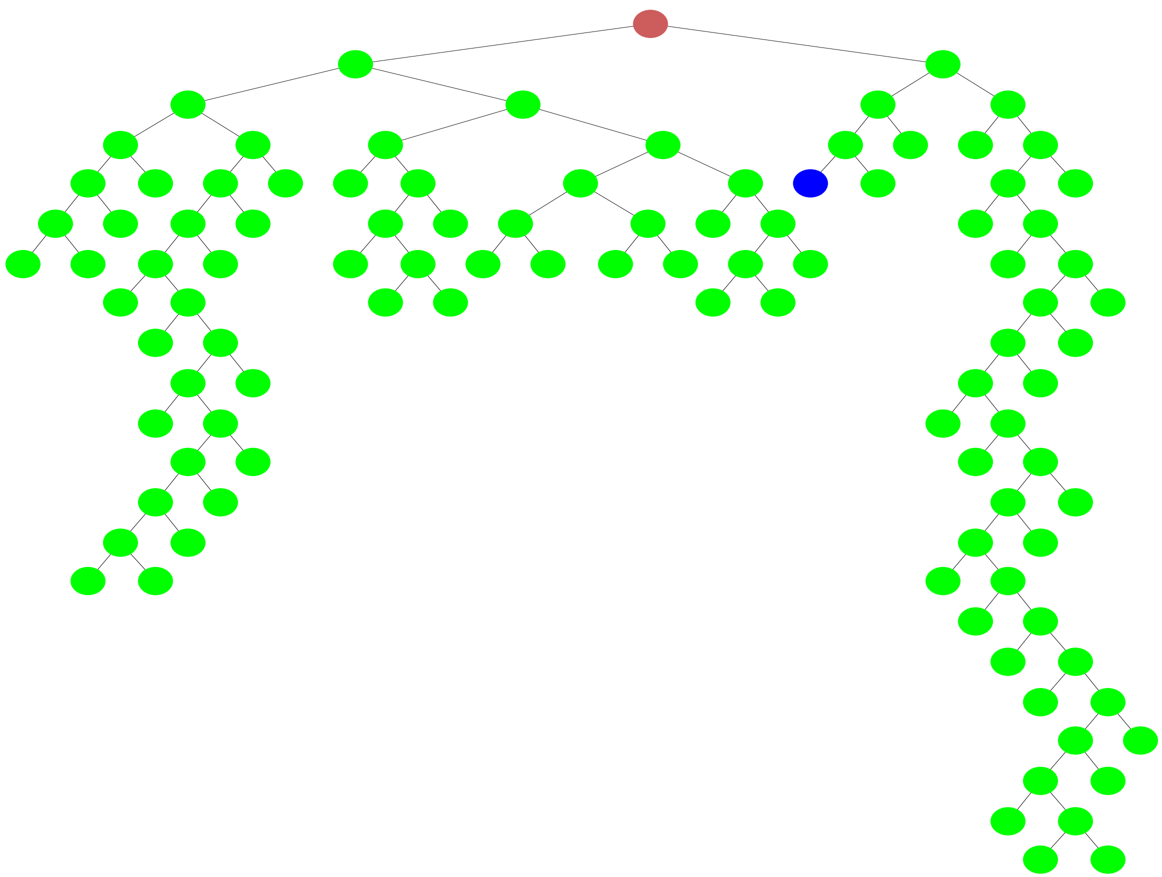
Branching

Branching involves partitioning the feasible region with hyperplanes such that:

- All optimal solutions are in one of the members of the partition.
- The solution to the current relaxation is not in any of the members of the partition.



Branch and Bound Tree



Heuristics

- *Heuristics* are fast methods for finding "good" feasible solutions to mathematical programs when other methods fail.
- Constructive Methods
  - Attempt to construct a solution by selecting items from the ground set one by one.
  - Usually done using a *greedy* selection criteria.
- Improvement Methods
  - Begin with a feasible solution obtained using another method and try to improve on it.
  - Typically done by discarding some of the items in the solution and choosing others in such a way that the solution improves.
  - These are also sometime called *local search methods* because they search in the local area of a given solution for better ones.
- Important: Heuristics do not "solve" problems, they just find good solutions!!!

Other Tools and Techniques

Reformulation

- Not all formulations are created equal.
- A given mathematical programming problem may have many alternative formulations.
- For MILPs, "stronger" formulations make problems easier to solve.
- Example: Facility Location revisited
  - Consider the constraints $\sum_{i=1}^{m} x_{ij} \leq m y_j \; \forall j$.
  - These can be replaced with $x_{ij} \leq y_j \; \forall i, j$
- Adding variables or recasting with a completely different set of variables can also help.
- Various *automatic reformulation techniques* have been successful in improving our ability to solve difficult problems.
- Decomposition, preprocessing, and cutting plane generation are three simple methods for strengthening initial formulations.

Decomposition

- *Decomposition methods* try to reduce solution of a difficult model to solution of a series of easier models.
- This is done by relaxing certain constraints and then trying to implicitly enforce them.
- One typical approach, called *Lagrangian relaxation*, assigns a penalty in the objective function for violating the relaxed constraints.
- The difficult part is identifying which constraints to relax.
- Methods for automatically detecting structure and applying a decomposition method are a promising area for research.

Preprocessing and Logical Tightening

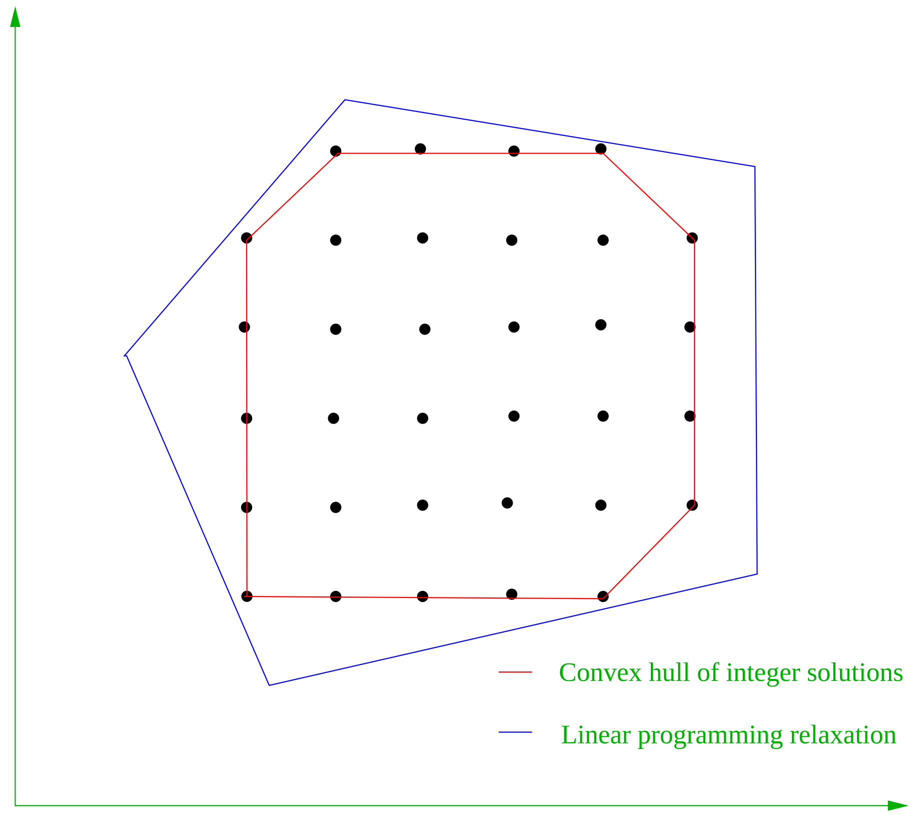
- *Preprocessing* techniques use various logical rules to tighten bounds on both variables and constraints.
- Example: We can derive *implied bounds* for variables from each constraint $ax \leq b$. If $a_0 > 0$, then

$$x_1 \leq (b - \sum_{j:a_j>0} a_j l_j - \sum_{j:a_j<0} a_j u_j)/a_0$$

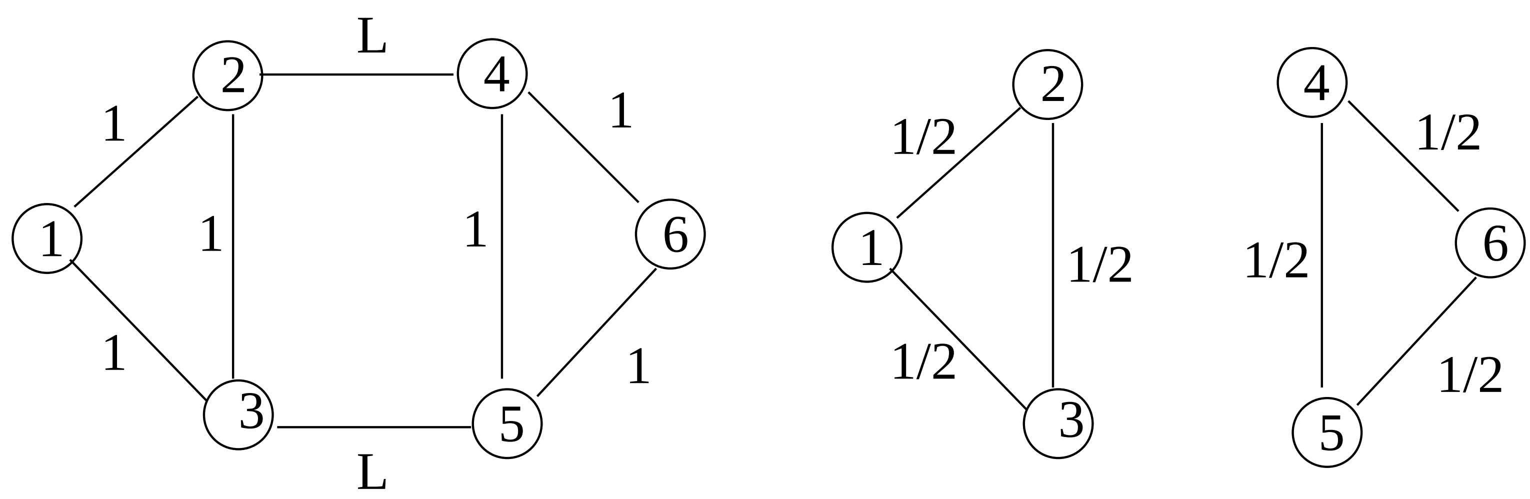
- Many other such rules can be applied in order to strengthen the formulation and obtain better bounds.
- Similar techniques are used in *constraint logic programming*.

Strong Valid Inequalities

Any inequality valid for all optimal solutions to a given MILP can be used to obtain improved bounds.



Valid Inequality Example



- Consider the graph on the left above.
- The optimal perfect matching has value $L + 2$.
- The optimal solution to the LP relaxation has value 3.
- This formulation can be extremely weak.
- Add the *valid inequality* $x_{24} + x_{35} \geq 1$.
- Every perfect matching satisfies this inequality.

Primal Heuristics

- *Primal heuristics* are used to perturb a solution to a relaxation into a solution feasible for the original MILP.
- The most common primal heuristics are based on rounding the solution obtained by solving an LP relaxation.
- The success of such methods depends on
  - how easy it is to achieve feasibility, and
  - how much rounding increases the objective function value.
- Another possibility is to use a constructive method initialized by fixing all the variables that are integer-valued in the current LP solution.
- This could be followed by an improvement method.

User Interfaces

Bringing MILP to the Masses

- User interfaces are the bridge between theory and practice.
- User interfaces allow practitioners to apply methodology developed by academics to *real problems*.
- How do practitioners use these tools?
  - As a "black box" to solve a given model specified in a *modeling language*.
  - As a "black box" embedded within a larger application using a callable library.
  - As building blocks within customized solvers.

Common Infrastructure for Operations Research (COIN-OR)

- The *COIN-OR* project is a consortium of researchers from both industry and academia.
- We are dedicated to promoting the development and use of interoperable, open-source software for operations research.
- We are also dedicated to defining standards and interfaces that allow software components to interoperate with other software, as well as with users.
- Check out the Web site for the project at

  `http://www.coin-or.org`

- There is also a Lehigh site devoted to COIN tutorial materials at

  `http://sagan.ie.lehigh.edu/coin`

- COIN-OR is involved in research in all of the areas in the second-half of the talk.

Modeling Languages and Data Interchange Formats

- A *modeling language* is a human-readable syntax for specifying a model.
- Modeling languages typically strive to be "solver independent."
- Using a modeling language, the user can write a text-based description of the model that could be read from a file by a solver.
- Some existing modeling languages
  - AMPL
  - GAMS
  - MPL
  - OPL
  - GMPL
- Non-human-readable formats are also needed to store model data or pass it to other applications.
- Current modeling languages are somewhat limited in the integer programming models they can express.
- One area prime for research is increasing the richness of modeling languages to allow specification of more complex (combinatorial) models.

Callable Libraries

- More sophisticated users may prefer to access the solver directly from application code without going through a modeling language.
- This requires specifying an API.
- The Open Solver Interface (OSI) is a uniform API available from COIN-OR that provides a common interface to numerous solvers.
- Using the OSI improves portability and eliminates dependence on third-party software.

Algorithmic Frameworks

- An *algorithmic framework* allows the user to easily modify the inner workings of the algorithm.
- In branch and bound, a user might want to develop a new branching rule or generate some cutting planes without implementing from scratch.
- In a framework, functionality must be modularized and interfaces well-defined.
- Existing frameworks for MILP
  - MINTO
  - ABACUS
  - SYMPHONY
  - COIN/BCP
  - ALPS (under development)

Computation

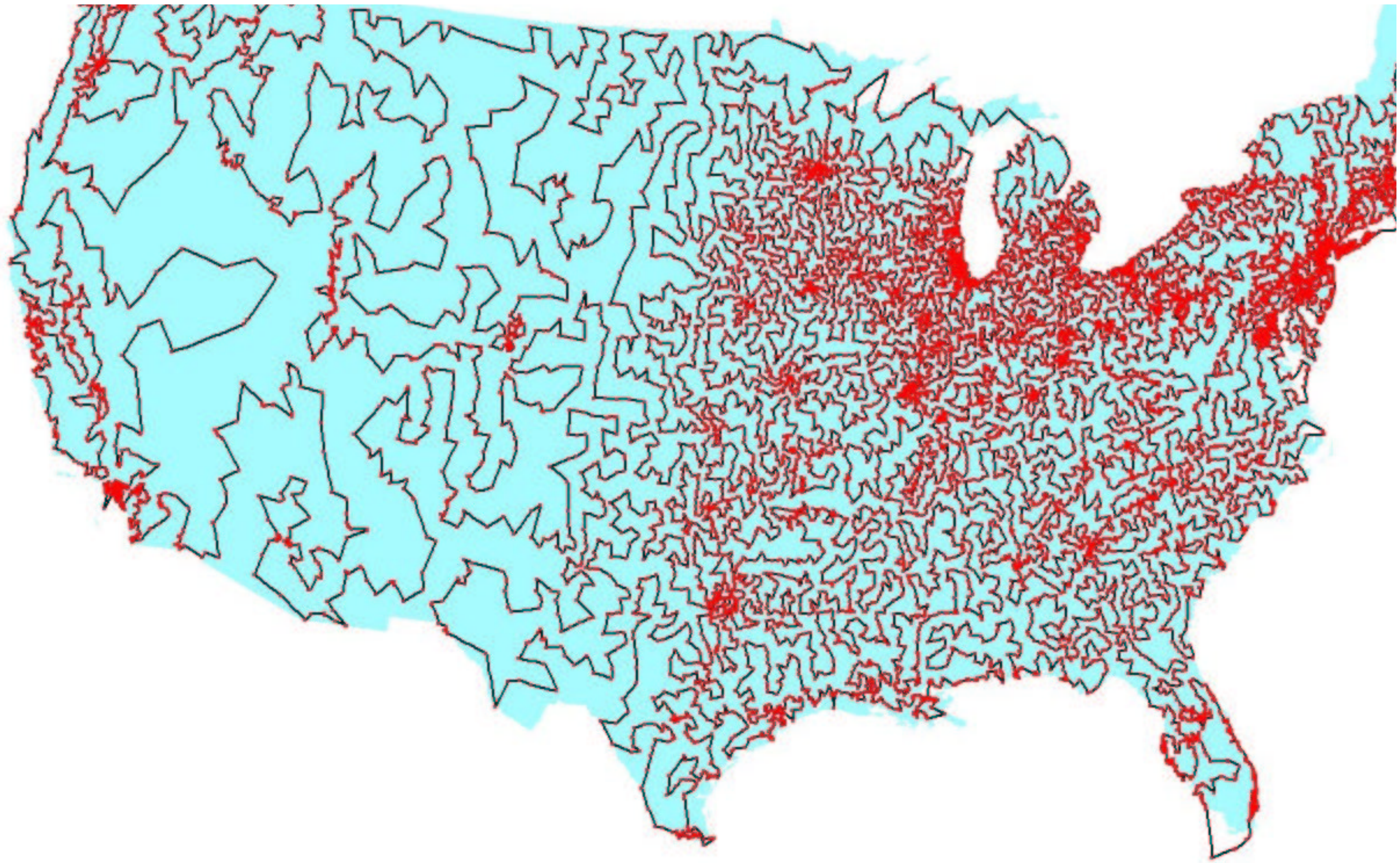
Large-scale Computation

- With large-scale computation, many practical issues arise.
  - speed
  - memory usage
  - numerical stability
- Dealing with these issues can be more of an art than a science.
- This is an area in which we have a great deal to learn.

Distributed Computing

- High-performance computing is becoming increasingly affordable.
- The use of parallel algorithms for solving large-scale problems has become a realistic option for many users.
- Developing parallel algorithms raises a range of additional issues.
- The name of the game in parallel computation is to avoid doing unnecessary computations (right hand doesn't know what the left hand is doing).
- To void unnecessary work, processing units have to share information.
- Information sharing also has a cost, so there is a tradeoff.
- Achieving the correct balance is challenging.
- This is an area of active research.

Current State of the Art



Outlook

- Currently, IP researchers are in search of the "next big breakthrough" in methodology.
- More work is needed on improving user interfaces and making IP technology accessible to practitioners.
- There is still a lot to be learned about computation and large-scale IP.
- Applying IP to a new application area can have a big impact.
- Many application areas remain untapped.

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)
  - Electronic commerce/combinatorial auctions
  - Computational biology