

# Making Molehills Out of Mountains: A Guided Tour of Discrete Optimization

TED RALPHS  
LEHIGH UNIVERSITY



CSIRO, Melbourne, Australia, 15 December 2011

**Thanks:** Work supported in part by the National Science Foundation

# Outline

- 1 Introduction
- 2 What is Optimization?
- 3 Applications of Optimization
- 4 Basic Solution Framework
- 5 Advanced Methods
  - Decomposition
  - Parallelization
- 6 Software

## 1 Introduction

## 2 What is Optimization?

## 3 Applications of Optimization

## 4 Basic Solution Framework

## 5 Advanced Methods

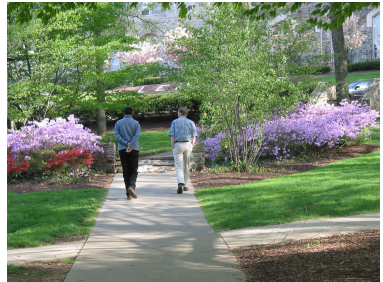
- Decomposition
- Parallelization

## 6 Software

# Lehigh University

## Profile

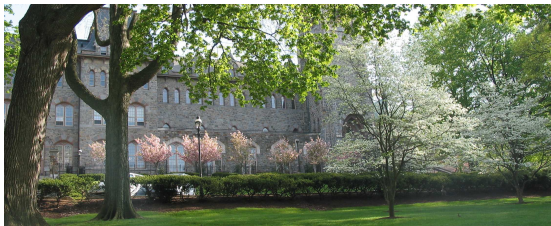
- Small private university.
- Founded in 1865 by railroad baron Asa Packer.
- Students
  - 5000 undergraduate students
  - 2000 graduate students
- Located in historic Bethlehem, PA



# Industrial and Systems Engineering at Lehigh

## Profile

- IE department established in 1948.
- Among the oldest in the U.S.
- In Mohler Lab, namesake of former president and CEO of Hershey Foods.
- Department offers two BS programs, five MS programs, and a Ph.D Program
- We have about 160 undergraduate, 175 masters, and 40 Ph.D students.



# My Background

- B.S. and M.S. in Mathematics from Carnegie Mellon University.
- Ph.D from Cornell in Operations Research with major in optimization and minors in computer science and statistics.
- Faculty member at Lehigh for 10 years, director of COR@L Lab.
- Research interests
  - Computational optimization
  - Parallel/Grid computing
  - Open source software development



# Outline

1 Introduction

2 What is Optimization?

3 Applications of Optimization

4 Basic Solution Framework

5 Advanced Methods

- Decomposition
- Parallelization

6 Software

# What Is Optimization?

## Definition

Optimization involves the use of mathematical models to analyze the operations of large-scale systems and improve their efficiency.

## Optimization is everywhere!

- Airlines and railways use it to determine schedules.
- Logistics companies use it to determine routes.
- Mapping software uses it to calculate travel routes.
- Cell phone companies use it figure out where to place towers.
- Brokerage firms use it to make investment decisions.
- Doctors use it to plan medical treatments.
- Biologists use it to sequence genomes.
- Even nature operates according to principles of optimization.



To analyze a system, we start with a mathematical model.

*Variables* are quantities that determine the operating state.

How many widgets should we manufacture?

How many employees should we hire?

*Constraints* are the specs that determine the allowable operating states.

We only have \$100K in operating capital.

We can only purchase raw materials for 10,000 watchamacallits.

*Objectives* specify the goal(s) of the system.

Maximize profit.

Minimize risk.

# A Mathematical Optimization Model

The general form of a *mathematical optimization model* is

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

where  $X \subseteq \mathbb{R}^n$  is a set that may be discrete.

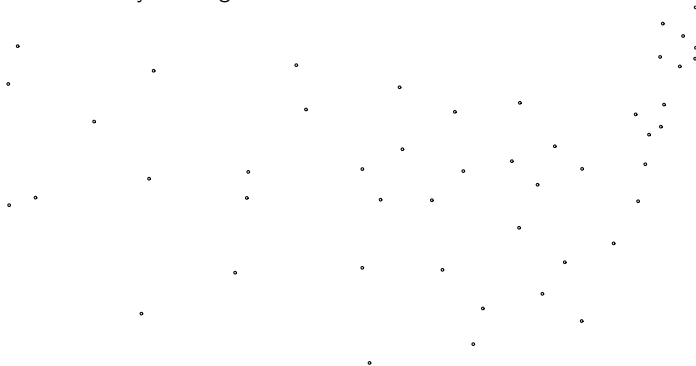
- A *mathematical optimization problem* is a problem that can be expressed using a mathematical optimization model (called the *formulation*).
- The form of the functions used to define and the constraints and the form of the set  $X$  determine the appropriate solution method.
- In this talk, we consider models with linear functions and for which  $X = \mathbb{R}^{n-p} \times \mathbb{Z}^p$ .

## Possible Outcomes

- When we say we are going to “solve” mathematical program, we mean to determine
  - whether it is feasible, and
  - whether it has an optimal solution.
- We may also want to know some other things, such as the status of its “dual” or about sensitivity.

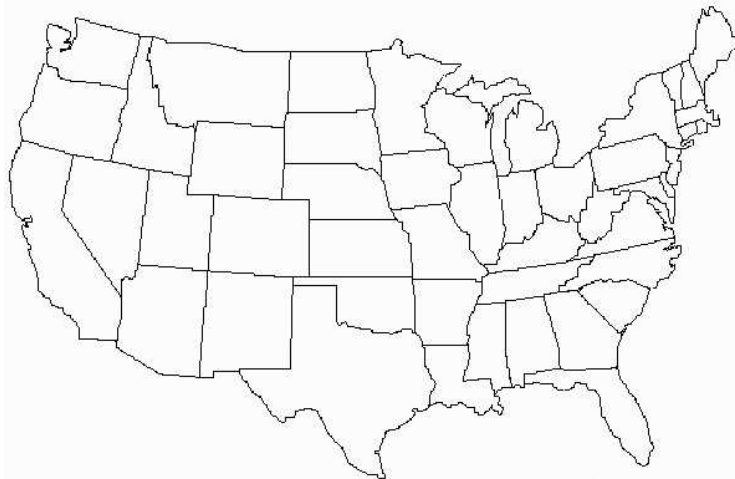
# How Hard Can It Be?

- The number of possibilities is typically **MUCH TOO LARGE** to consider each one explicitly.
- The Traveling Salesman Problem (TSP) is one of the most well-known optimization problems.
  - A traveling salesman must visit  $n$  cities and then return home.
  - He wants to minimize the total distance traveled.
  - How many orderings are there?



# A Big Number

12413915592536072670862289047373375038521486354677761457318634!



# A Needle in a Haystack

- Analyzing these models involves implicitly searching the space of all possible states.
- Two avenues for improvement
  - **Smarter**: More sophisticated methodology
  - **Stronger**: Use a bigger hammer (parallel computing)
- Successful application involves a marriage of methodology, software, and hardware.



# Outline

1 Introduction

2 What is Optimization?

3 Applications of Optimization

4 Basic Solution Framework

5 Advanced Methods

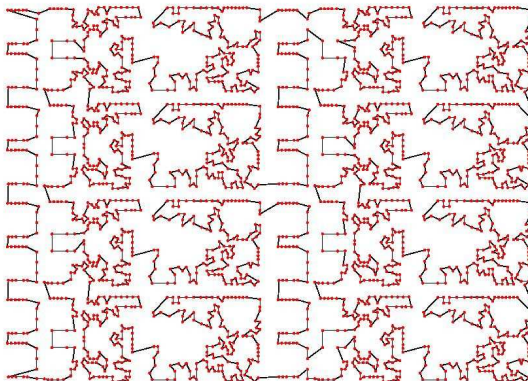
- Decomposition
- Parallelization

6 Software

# Applications: Traveling Salesman Problem

The TSP is one of the most well-known and well-studied optimization can be used to model numerous real-world problems

- Routing and scheduling
- Manufacture of integrated circuits
- Gene sequencing
- Design of sonet rings





# Applications: Location, Routing, and Scheduling

Vehicle routing is a well-studied generalization of the TSP.

## Problem data

- A set of **customers** with known demands for a single commodity.
- A fleet of identical **trucks** with fixed capacity.
- A single fixed **depot**.
- Travel times between pairs of locations.

## Vehicle Routing Problem (VRP)

The problem is to

- assign customers to trucks such that capacity is not exceeded and
  - sequence the customers assigned to each truck
- such that the *total travel time is minimized*.

More complex models can involve additional decisions

- Where to locate warehouses in order to allow efficient routes to be designed.
- How to do scheduling of deliveries over a multi-day time horizon.

# Applications: Location, Routing, and Scheduling Software

**Routes! - [Maps]**

8/11/2004 Date

Service Area: WASHINGTON STATE

Route Type: Date

Date: 8/11/2004

Stops:

R#	WP	Frz	Time	Street	
0	0	<input type="checkbox"/>	0 h 15 m	5610 176TH ST SW	LYNI
1	2	<input type="checkbox"/>	0 h 15 m	2959 SARATOGA ROAD	LANI
1	3	<input type="checkbox"/>	0 h 15 m	308 ALDERWOOD ST	COU
1	4	<input type="checkbox"/>	0 h 15 m	1399 W. BEACH RD.	OAK
1	5	<input type="checkbox"/>	0 h 15 m	2040 PINEWOOD WAY	OAK
1	6	<input type="checkbox"/>	0 h 15 m	585 BIRCH ST	OAK
1	7	<input type="checkbox"/>	0 h 30 m	736 TILLAMUK DRIVE	LA C
1	8	<input type="checkbox"/>	0 h 15 m	106 SOUTH 3RD STREET	LA C
1	9	<input type="checkbox"/>	0 h 15 m	19586 SKYDDA LANE	MT V
1	10	<input type="checkbox"/>	0 h 15 m	1122 SEAMIST LANE	CAM
1	11	<input type="checkbox"/>	0 h 15 m	3324 BERNIE RD	CAM
1	12	<input type="checkbox"/>	0 h 15 m	1444 OKSENOHP RD	CAM
1	13	<input type="checkbox"/>	0 h 15 m	1110-316 N.W. STREET	STA
1	14	<input type="checkbox"/>	0 h 15 m	2732 177TH PL. NE	ARLI
2	2	<input type="checkbox"/>	0 h 15 m	58 UNTER ST	SNO
2	3	<input type="checkbox"/>	0 h 30 m	58 UNTER ST	SNO
2	4	<input type="checkbox"/>	0 h 15 m	7402 MOON VALLEY ROAD	NOR
2	5	<input type="checkbox"/>	0 h 15 m	26916 SE 9TH WAY	SAM
2	6	<input type="checkbox"/>	0 h 15 m	22500 SE 56TH ST APT # 13	ISSA
2	7	<input type="checkbox"/>	0 h 15 m	24266 SE 47TH PL	ISSA

Auto Stop Times

Map

History Waybill

Area: WASHINGTON STATE

Customer: UNASSIGNED

Location: 11801 120TH AVE NE

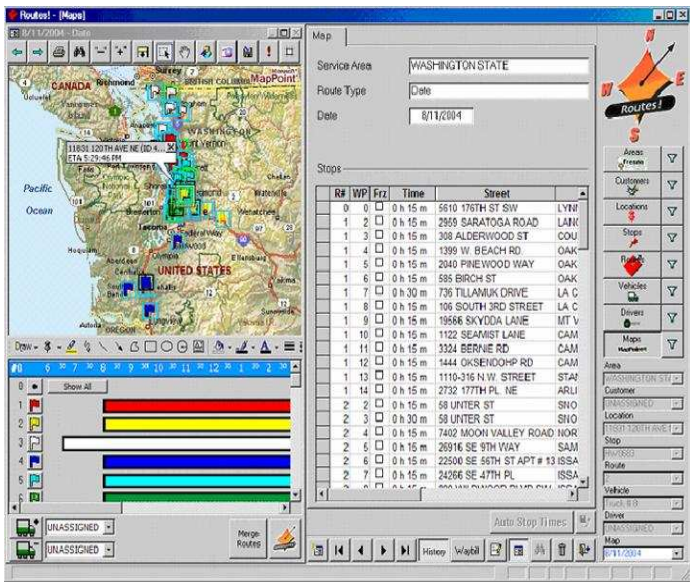
Stop: UNASSIGNED

Route: UNASSIGNED

Vehicle: UNASSIGNED

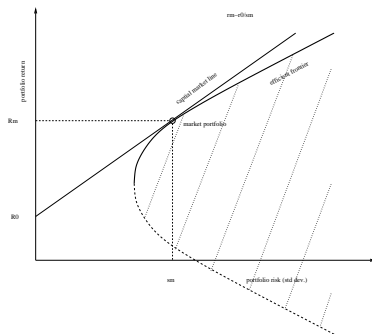
Driver: UNASSIGNED

Map: 8/11/2004



# Applications: Financial Optimization

- Since Markowitz's pioneering work on [portfolio optimization](#), mathematical models have been heavily used in financial markets.
- Unfortunately, it was the use of optimization to create financial derivatives that in part lead to the recent world financial crisis.
- Optimization models are driving the investment decisions of many institutional investors these days.



# Applications: Biology, Healthcare, Medicine

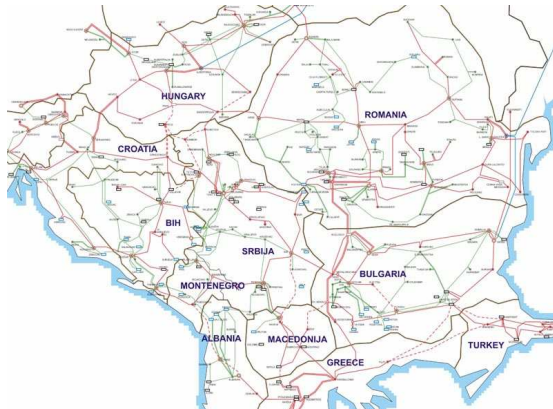
- Optimization is being used increasingly in the healthcare industry.
  - Hospital operations
  - Design of treatment plans
  - Determination of risk factors for disease
  - Drug discovery
  - Computational biology
- In some cases, optimization is being used prescriptively to determine the course of medical treatment.
- The most successful example of this has been in cancer radiation therapy.
- Optimization techniques are also used to solve classification problems associated with identifying risk factor for disease.
- Optimization holds promise as a future method for drug discovery.



# Applications: Network Design and Analysis

A wide range of optimization models apply to the design and analysis of networks (power, telecommunications, highway, distribution, etc.) with the goal of improving

- Survivability/Robustness
- Latency
- Congestion
- Speed



# Applications: Nash and Stackelberg Games

- Many game theoretic models can be formulated as optimization problems involving multiple decision makers.
- In a *Nash game*, the players are treated as equals and take simultaneous action.
- Computationally, one often wishes to find a *Nash equilibrium*, in which the action of each player is optimal, given the actions of all other players.
- In a *Stackelberg game*, there is a dominant player, called the *leader*, who acts first and other players react.
- In this case, one is concerned with determining the leader's decision, given the assumption that the *followers* will react optimally.
- This can often be modeled as a bilevel program.



# Applications: Competition and Markets

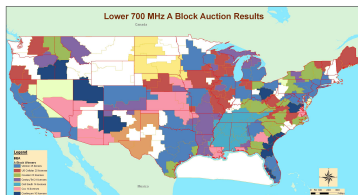
- Hierarchical decision systems
  - Government agencies
  - Large corporations with multiple subsidiaries
  - Markets with a single “market-maker.”
  - Decision problems with recourse
- Parties in direct conflict
  - Zero sum games
  - Interdiction problems
- Modeling “robustness”: leader represents external phenomena that cannot be controlled.
  - Weather
  - External market conditions
- Controlling optimized systems: follower represents a system that is optimized by its nature.
  - Electrical networks
  - Biological systems

## Applications: Combinatorial Auctions

- A **combinatorial auction** is an auction in which participants are allowed to bid on subsets of available goods.
- This accounts for the fact that some items have a greater (or lesser) worth when combined with other items.
- Study of such auctions originated as methods for fairly distributing public goods.
- A set of items along with an offered price constitutes a **bid**.
- Given a set of bids, determining an **efficient allocation** is an optimization problem.

## Example: Spectrum auctions

- In many countries, the government uses auctions to allocate licenses for certain frequency spectra by region.
- The value of a set of licenses is increased if they are in contiguous regions.





# Outline

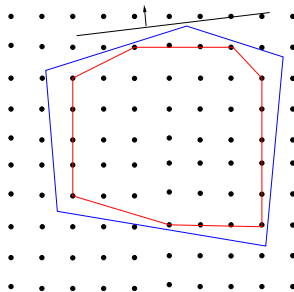
- 1 Introduction
- 2 What is Optimization?
- 3 Applications of Optimization
- 4 Basic Solution Framework**
- 5 Advanced Methods
  - Decomposition
  - Parallelization
- 6 Software

# Discrete Optimization

(Integer) Linear Optimization: Minimize/Maximize a linear *objective function* over a (discrete) set of *solutions* satisfying specified *linear constraints*.

$$z_{\text{IP}} = \min_{x \in \mathbb{Z}_+^n} \left\{ c^\top x \mid Ax \geq b \right\} \quad (\text{MIP})$$

$$z_{\text{LP}} = \min_{x \in \mathbb{R}_+^n} \left\{ c^\top x \mid Ax \geq b \right\} \quad (\text{LP})$$



## Textbook Example: Facility Location Problem

- We are given  $n$  potential facility locations and  $m$  customers.
- 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.
- Here is one formulation:

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

# Solving Discrete Optimization Problems

- In general, *convex optimization problems* are “easy” to solve.
- In essence, this is because convex problems have only one local minimum.
- Discrete optimization problems are particularly challenging
  - the feasible region is *nonconvex* and
  - the description of the feasible region, though compact, is *implicit*.
- More computationally useful descriptions of the feasible region can be obtained by either
  - *Convexification*  $\Rightarrow$  iteratively construct an explicit description of the convex hull of feasible solutions (cutting plane method)
  - *Disjunction*  $\Rightarrow$  using a set of judiciously chosen logical disjunctions to represent the feasible region as a finite union of polyhedra (branch and bound)
- In general, both of these approaches lead to descriptions of exponential size (bad).
- We typically only need a small part of the description to solve the problem.
- Modern state-of-the-art algorithms effectively combine these two techniques.

# Implicit Enumeration

- *Implicit enumeration* methods enumerate the solution space in an intelligent way.
- The most common algorithm of this type is *LP-based branch and bound*.
- Suppose  $F$  is the set of feasible solutions for a given MILP. We wish to solve  $\min_{x \in F} c^\top x$ .

## Divide and Conquer

Consider a *partition* of  $F$  into subsets  $F_1, \dots, F_k$ . Then

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

We can then solve the resulting *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 by solving a convex relaxation.

# Branch and Bound

- 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.
- Relaxations can be used to efficiently derive bounds on the optimal value.
- Types of Relaxations
  - Convex/Continuous relaxations
  - Combinatorial relaxations
  - Lagrangian relaxations

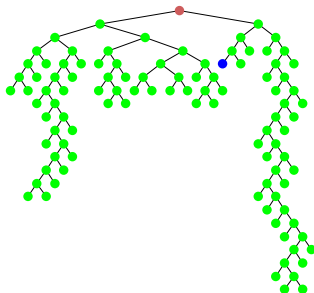
## Branch and Bound

Initialize the queue with the root  $F$ . While there are subproblems in the queue, do

- 1 Remove a subproblem and solve its relaxation.
- 2 The relaxation is infeasible  $\Rightarrow$  subproblem is infeasible and can be pruned.
- 3 Solution is feasible for the MILP  $\Rightarrow$  subproblem solved (update upper bound).
- 4 Solution is not feasible for the MILP  $\Rightarrow$  lower bound.
  - If the lower bound exceeds the global upper bound, we can *prune the node*.
  - Otherwise, we *branch* and add the resulting subproblems to the queue.

# The Search Tree

- 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*.
- Throughout the algorithm, we have global upper and lower bounds that are growing together.
- The goal of the algorithm is to achieve equality of these bounds.

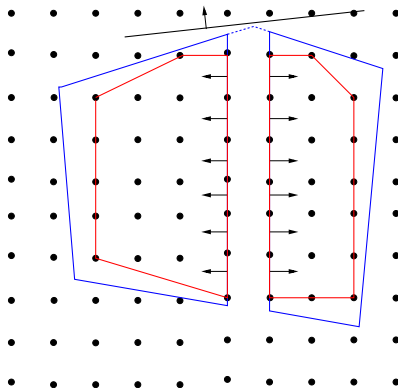


Research Question: How do we manage the tradeoff between improving the upper and lower bounds.

## First Ingredient: Branching

Branching involves partitioning the feasible region using a logical disjunction 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.



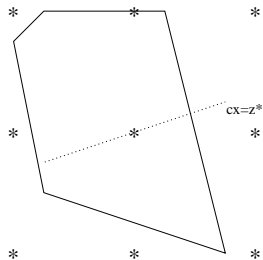
Research Question: How do we generate effective branching disjunctions?



# Solving MILPs with Branch and Bound

$$\text{MILP: } \min_{x \in S} cx$$

$$S: \begin{array}{lcl} Ax & \geq & b \\ x & \in & \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$



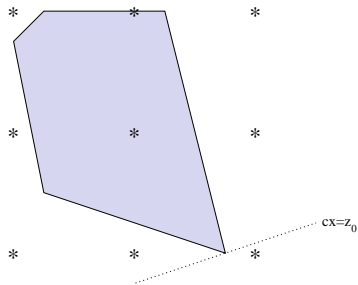
# Solving MILPs with Branch and Bound

$$\text{MILP: } \min_{x \in S} cx$$

$$S: \begin{array}{lcl} Ax & \geq & b \\ x & \in & \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$

$$\text{LP: } \min_{x \in \mathcal{P}} cx$$

$$\mathcal{P}: \begin{array}{lcl} Ax & \geq & b \\ x & \in & \mathbb{R}^n, \end{array}$$



$$\text{LB} = z_0$$

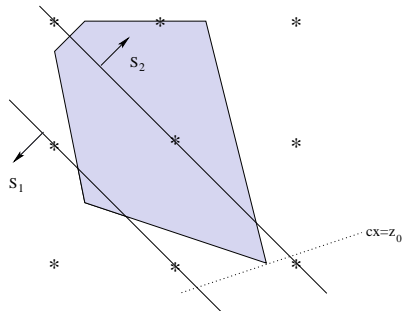
# Solving MILPs with Branch and Bound

$$\text{MILP: } \min_{x \in S} cx$$

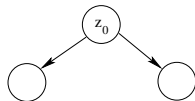
$$S: \begin{array}{l} Ax \geq b \\ x \in \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$

$$\text{LP: } \min_{x \in \mathcal{P}} cx$$

$$\mathcal{P}: \begin{array}{l} Ax \geq b \\ x \in \mathbb{R}^n, \end{array}$$



$$\text{LB} = z_0, x^* \in S_1 \cup S_2$$



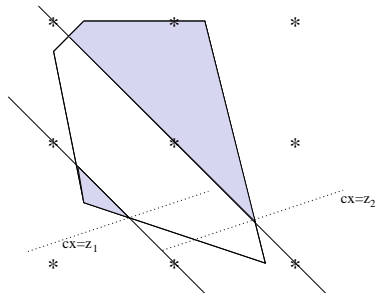
# Solving MILPs with Branch and Bound

$$\text{MILP: } \min_{x \in S} cx$$

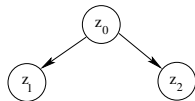
$$S: \begin{array}{l} Ax \geq b \\ x \in \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$

$$\text{LP: } \min_{x \in \mathcal{P}} cx$$

$$\mathcal{P}: \begin{array}{l} Ax \geq b \\ x \in \mathbb{R}^n, \end{array}$$



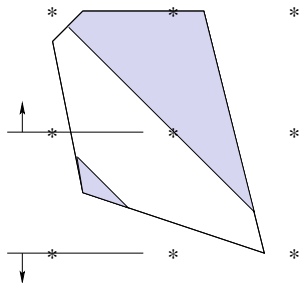
$$\text{LB} = \min(z_1, z_2)$$



# Solving MILPs with Branch and Bound

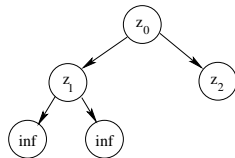
$$\text{MILP: } \min_{x \in S} cx$$

$$S: \begin{array}{l} Ax \geq b \\ x \in \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$



$$\text{LP: } \min_{x \in \mathcal{P}} cx$$

$$\mathcal{P}: \begin{array}{l} Ax \geq b \\ x \in \mathbb{R}^n, \end{array}$$

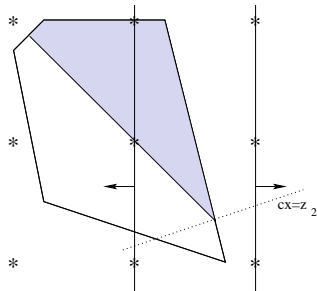


$$\text{LB} = z_2$$

## Solving MILPs with Branch and Bound

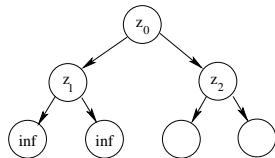
$$\text{MILP: } \min_{x \in S} cx$$

$$S: \begin{array}{lcl} Ax & \geq & b \\ x & \in & \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$



$$\text{LP: } \min_{x \in \mathcal{P}} cx$$

$$\mathcal{P}: \begin{array}{ll} Ax & \geq b \\ x & \in \mathbb{R}^n, \end{array}$$

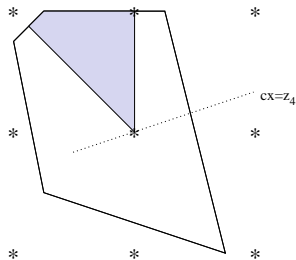


$$\text{LB} = z_2$$

# Solving MILPs with Branch and Bound

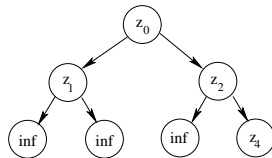
$$\text{MILP: } \min_{x \in S} cx$$

$$S: \begin{array}{l} Ax \geq b \\ x \in \mathbb{Z}^d \times \mathbb{R}^{n-d}, \end{array}$$



$$\text{LP: } \min_{x \in \mathcal{P}} cx$$

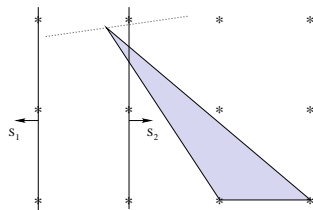
$$\mathcal{P}: \begin{array}{l} Ax \geq b \\ x \in \mathbb{R}^n, \end{array}$$



$$\text{LB} = z_4 = \text{UB}$$

# The Branching Disjunction

- Most commonly used branching disjunction is  $x_i \leq \pi_0 \vee x_i \geq \pi_0 + 1$  for an  $i \in \{1, \dots, d\}$ .
- e.g.  $S_1 = \{x | x_1 \leq 0\}$ ,  $S_2 = \{x | x_1 \geq 1\}$ .
- This is called **Variable Disjunction**. Also denoted as:  $x_1 \leq 0 \vee x_1 \geq 1$ .

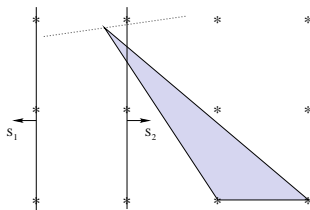


A variable disjunction

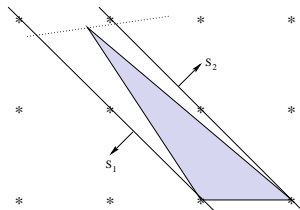


# The Branching Disjunction

- Most commonly used branching disjunction is  $x_i \leq \pi_0 \vee x_i \geq \pi_0 + 1$  for an  $i \in \{1, \dots, d\}$ .
- e.g.  $S_1 = \{x | x_1 \leq 0\}$ ,  $S_2 = \{x | x_1 \geq 1\}$ .
- This is called **Variable Disjunction**. Also denoted as:  $x_1 \leq 0 \vee x_1 \geq 1$ .
- Disjunctions like  $x_1 + x_2 \leq 4 \vee x_1 + x_2 \geq 5$  are also valid.
- A **General Disjunction** is of the form  $\pi x \leq \pi_0 \vee \pi x \geq \pi_0 + 1$ , where  $(\pi, \pi_0) \in \mathbb{Z}^d \times 0^{n-d} \times \mathbb{Z}$ .



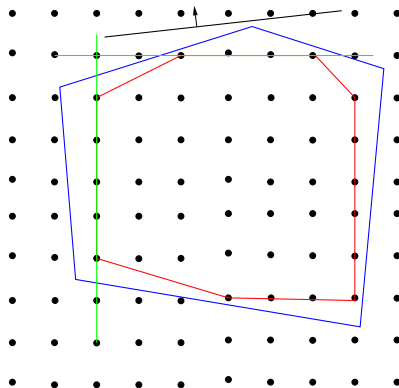
A variable disjunction



A general disjunction

## Second Ingredient: Bounding

- The method by which bounds are derived in branch and bound is perhaps the most crucial element of an effective algorithm.
- The stronger the bound, the fewer nodes have to be enumerated.
- The most common method of bounding is to develop an outer approximation of the convex hull of feasible solutions, yielding a convex relaxation.
- This often done by analyzing relaxations arising from disjunctions.



## Facility Location Problem (Alternative Formulation)

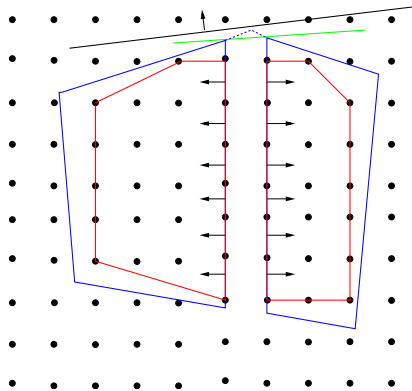
- Here is another formulation for the same problem:

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

- Notice that the set of integer solutions contained in each of the polyhedra is the same (why?).
- However, the second polyhedron is strictly included in the first one.
- Therefore, the second polyhedron will yield a **better lower bound**.
- The second polyhedron is a **better approximation** to the convex hull of integer solutions.

## A Common Framework: Disjunctive Optimization

- Disjunctions can also be used to generate outer approximations by taking the convex hull of the union of the polyhedra obtained by imposing the disjunction.
- This gives a tighter polyhedral approximation than the original formulation.
- The procedure can be iterated to obtain progressive improvements.



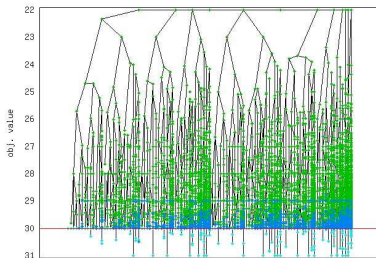
Research Question: Given a disjunction, should we use it to branch or to tighten our approximation?

# Other Supporting Ingredients

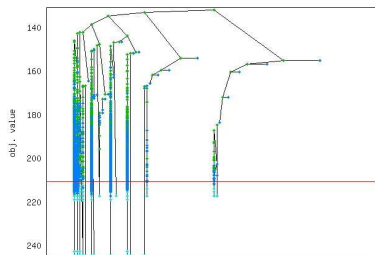
There are a number of other ingredients that also play important roles in the effectiveness of enumeration algorithms in practice:

- Preprocessing/Reformulation
- Search strategy
- Primal heuristics

B&B tree (stein45.dat 8221s SYMPHONY)

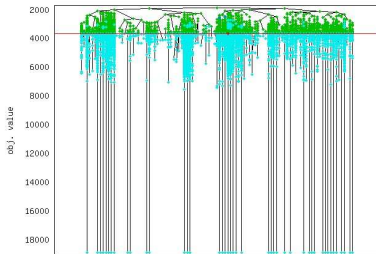


B&B tree (neos-803219.dat 712s SYMPHONY)

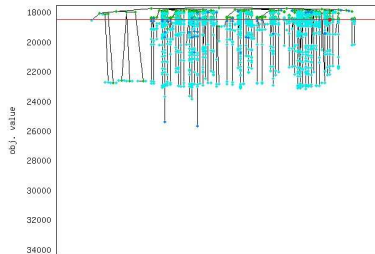


# A Couple Thousand Words

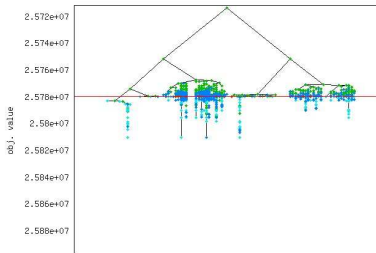
B&B tree (neos-504674.dat 1619s SYMPHONY)



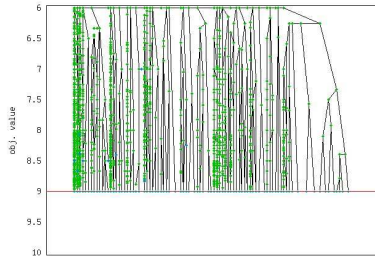
B&B tree (neos-598183.dat 3082s SYMPHONY)



B&B tree (gesa2o.dat 1197s SYMPHONY)

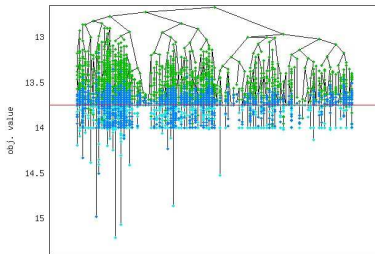


B&B tree (neos11.dat 6124s SYMPHONY)

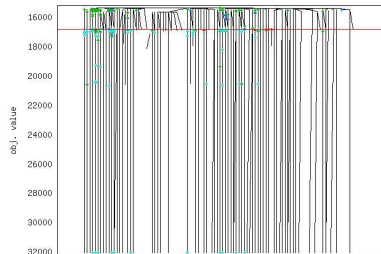


# Another Couple Thousand Words

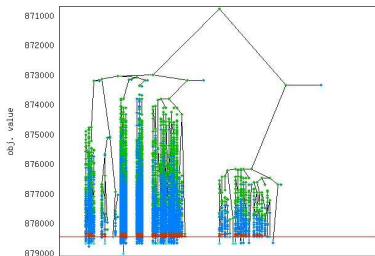
B&B tree (vpm2.dat 1044s SYMPHONY)



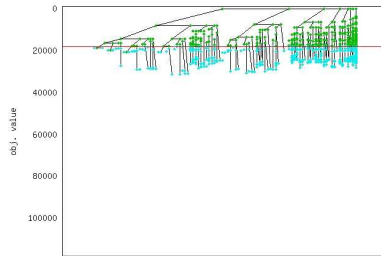
B&B tree (neos-603073.dat 3749s SYMPHONY)



B&B tree (bell13a.dat 296s SYMPHONY)



B&B tree (neos-522351.dat 1850s SYMPHONY)



# Outline

- 1 Introduction
- 2 What is Optimization?
- 3 Applications of Optimization
- 4 Basic Solution Framework
- 5 Advanced Methods**
  - Decomposition
  - Parallelization
- 6 Software

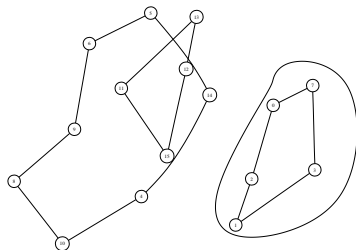
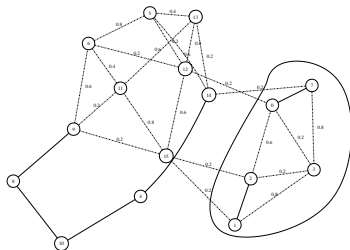


# Outline

- 1 Introduction
- 2 What is Optimization?
- 3 Applications of Optimization
- 4 Basic Solution Framework
- 5 Advanced Methods
  - Decomposition
  - Parallelization
- 6 Software

# What is the Goal of Decomposition?

- **Basic Idea:** Exploit knowledge of underlying structural components of model to improve bound.
- Many complex models are built up from multiple underlying substructures.
  - Subsystems linked by global constraints.
  - Complex combinatorial structures obtained by combining simple ones.
- We want to exploit knowledge of efficient, customized solution methodology for substructures.
- This can be done in two primary ways (with many variants).
  - Identify independent subsystems.
  - Identify subsets of constraints that can be dealt with efficiently.



## Example: Exposing Combinatorial Structure

### Traveling Salesman Problem Formulation

$$\begin{aligned}x(\delta(\{u\})) &= 2 && \forall u \in V \\x(E(S)) &\leq |S| - 1 && \forall S \subset V, 3 \leq |S| \leq |V| - 1 \\x_e &\in \{0, 1\} && \forall e \in E\end{aligned}$$



## Example: Exposing Combinatorial Structure

### Traveling Salesman Problem Formulation

$$\begin{aligned}x(\delta(\{u\})) &= 2 && \forall u \in V \\x(E(S)) &\leq |S| - 1 && \forall S \subset V, 3 \leq |S| \leq |V| - 1 \\x_e &\in \{0, 1\} && \forall e \in E\end{aligned}$$



### Two relaxations

Find a spanning subgraph with  $|V|$  edges ( $\mathcal{P}' = \text{1-Tree}$ )

$$\begin{aligned}x(\delta(\{0\})) &= 2 \\x(E(V)) &= |V| \\x(E(S)) &\leq |S| - 1 && \forall S \subset V \setminus \{0\}, 3 \leq |S| \leq |V| - 1 \\x_e &\in \{0, 1\} && \forall e \in E\end{aligned}$$



## Example: Exposing Combinatorial Structure

### Traveling Salesman Problem Formulation

$$\begin{aligned}x(\delta(\{u\})) &= 2 && \forall u \in V \\x(E(S)) &\leq |S| - 1 && \forall S \subset V, 3 \leq |S| \leq |V| - 1 \\x_e &\in \{0, 1\} && \forall e \in E\end{aligned}$$



### Two relaxations

Find a spanning subgraph with  $|V|$  edges ( $\mathcal{P}' = \text{1-Tree}$ )

$$\begin{aligned}x(\delta(\{0\})) &= 2 \\x(E(V)) &= |V| \\x(E(S)) &\leq |S| - 1 && \forall S \subset V \setminus \{0\}, 3 \leq |S| \leq |V| - 1 \\x_e &\in \{0, 1\} && \forall e \in E\end{aligned}$$



Find a 2-matching that satisfies the subtour constraints ( $\mathcal{P}' = \text{2-Matching}$ )

$$\begin{aligned}x(\delta(\{u\})) &= 2 && \forall u \in V \\x_e &\in \{0, 1\} && \forall e \in E\end{aligned}$$



## Example: Exposing Block Structure

- A key original motivation for decomposition is to *relax linking constraints*, leaving a separable relaxation.
- The key is to identify *block structure* in the constraint matrix.
- The separability lends itself nicely to *parallel implementation*.

$$\begin{pmatrix} A_1'' & A_2'' & \cdots & A_n'' \\ \hline A_1' & & & \\ & A_2' & & \\ & & \ddots & \\ & & & A_n' \end{pmatrix}$$

## Example: Exposing Block Structure

- A key original motivation for decomposition is to *relax linking constraints*, leaving a separable relaxation.
- The key is to identify *block structure* in the constraint matrix.
- The separability lends itself nicely to *parallel implementation*.

$$\begin{pmatrix} A_1'' & A_2'' & \cdots & A_\kappa'' \\ A_1' & & & \\ & A_2' & & \\ & & \ddots & \\ & & & A_\kappa' \end{pmatrix}$$

## Example: Exposing Block Structure

- A motivation for decomposition is to expose *independent subsystems*.
- The key is to identify *block structure* in the constraint matrix.
- The separability lends itself nicely to *parallel implementation*.

### Generalized Assignment Problem (GAP)

- The problem is to assign  $m$  tasks to  $n$  machines subject to *capacity constraints*.
- An IP formulation of this problem is

$$\begin{aligned} \min \quad & \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} \\ & \sum_{j \in N} w_{ij} x_{ij} \leq b_i \quad \forall i \in M \\ & \sum_{i \in M} x_{ij} = 1 \quad \forall j \in N \\ & x_{ij} \in \{0, 1\} \quad \forall i, j \in M \times N \end{aligned}$$

- The variable  $x_{ij}$  is one if task  $i$  is assigned to machine  $j$ .
- The “profit” associated with assigning task  $i$  to machine  $j$  is  $c_{ij}$ .



# How Do We Exploit the Decomposition?

We can exploit the decompositions in a number of ways.

- We exploit our knowledge of underlying structure to get better bounds, primarily by taking advantage of our knowledge of a structured relaxation.

## Decomposition-Based Bounding Methods

- Lagrangian Relaxation: Solve the relaxation using a specialized algorithm with an objective function that imposes a penalty on the violation of the relaxed constraints.
  - Dantzig-Wolfe Decomposition: Reformulate the problem as one of taking combinations of the solutions to a relaxation.
  - Cutting Plane Method: Use knowledge of the structure of the relaxation to generate good outer approximations of the feasible region of the original problem.
- The above methods can be combined in a unified framework to yield a rich set of bounding methods.
  - We can also exploit existing block structure
    - To break the problem up into smaller chunks that can be solved in parallel using a standard MILP solver.
    - To reformulate in a simpler way by exploiting identical blocks.

# Outline

1 Introduction

2 What is Optimization?

3 Applications of Optimization

4 Basic Solution Framework

5 Advanced Methods

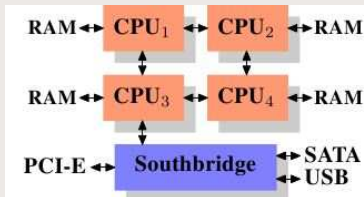
- Decomposition
- Parallelization

6 Software

# What Will Future Architectures Look Like?

- Moore's Law has now moved from clock speeds to numbers of cores.
- To take advantage of the capabilities of new hardware, effective parallelization will be the key.
- It seems clear that the next generation(s) of hardware will be clusters (of clusters) of machines with multiple multi-core chips.
- The result will be a memory hierarchy of ever-increasing complexity.

- Cache memory
- Main memory (local to core)
- Main memory (attached to other cores)
- Local disk
- Co-located distributed memory
- Remotely located distributed memory



- How do we efficiently this complex hierarchy?

# Measuring Performance of a Parallel System

- **Parallel System:** Parallel algorithm + parallel architecture.
- **Scalability:** How well a parallel system takes advantage of increased computing resources.

## Terms

- Sequential runtime:  $T_s$
  - Parallel runtime:  $T_p$
  - Parallel overhead:  $T_o = NT_p - T_s$
  - Speedup:  $S = T_s/T_p$
  - Efficiency:  $E = S/N$
- Standard analysis considers change in efficiency on a fixed test set as number of processors is increased.
  - This analysis is difficult to employ in practice, but the principle is clear.

# Parallel Overhead

- The amount of *parallel overhead* determines the scalability.
- “Knowledge sharing” is the main driver of efficiency.

## Major Components of Parallel Overhead in Tree Search

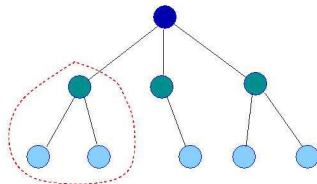
- Communication Overhead
  - Idle Time
    - Handshaking/Synchronization
    - Task Starvation
    - Memory Contention
    - Ramp Up Time
    - Ramp Down Time
  - Performance of Redundant Work
- The main challenge is for the data to be in the right place at the right time.

A supercomputer is a machine for turning a compute-bound problem into an I/O-bound problem. –Ken Barker

# Sources of Parallelism in Branch and Bound

Parallelization of tree search seems easy in principle...but in practice, it is not!

- **Tree parallelism:** Process different part of the tree simultaneously.
- **Decomposition:** Apply a decomposition approach in order to parallelize the bounding procedure.
- **Task parallelism**
  - Special procedures for the ramp-up/ramp-down phases
  - Parallelize primal heuristics
  - Parallelize cut generation
  - Process multiple trees simultaneously



# Basic Parallelization Approaches

## Local/shared memory computation (SYMPHONY)

- Work on one local copy of the tree and use threads to process multiple “chains” simultaneously.
- Pro: Implementation (load balancing) is much easier.
- Con: Expensive and limited computational resources.

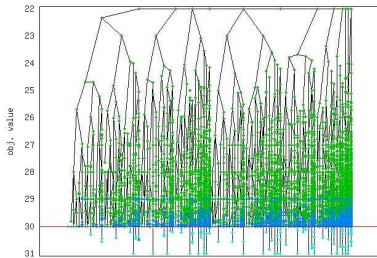
## Distributed memory computation (CHiPPS)

- Partition the tree and process subtrees asynchronously.
- Pro: No limit to hardware access, inexpensive.
- Con: Efficiency requires load balancing.

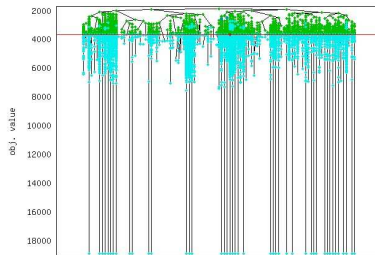
- To take advantage of modern hardware, a *hybrid approach* is required.
  - Partition the tree and process subtrees in a distributed fashion.
  - Locally, subtrees are processed in parallel using the shared memory approach.
- With this approach, we hope to do much of the parallel computation at the local level.

# A Couple Thousand Words

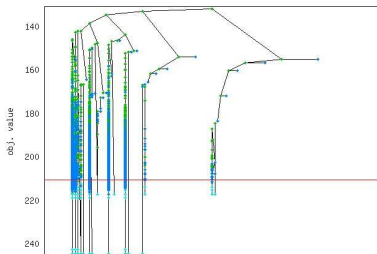
B8B tree (stein45.dat 8221s SYMPHONY)



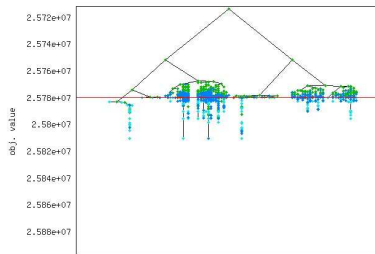
B8B tree (neos-504674.dat 1619s SYMPHONY)



B8B tree (neos-803219.dat 712s SYMPHONY)



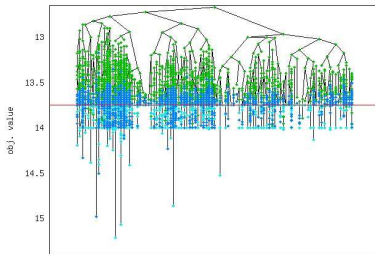
B8B tree (gesa2o.dat 1197s SYMPHONY)



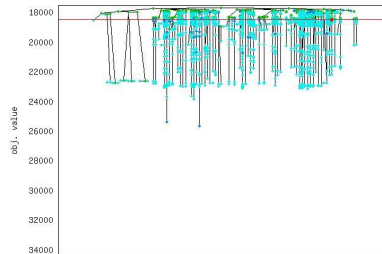


# Another Couple Thousand Words

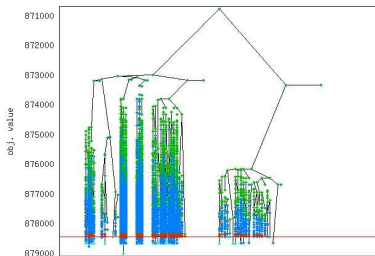
B8B tree (vpm2.dat 1044s SYMPHONY)



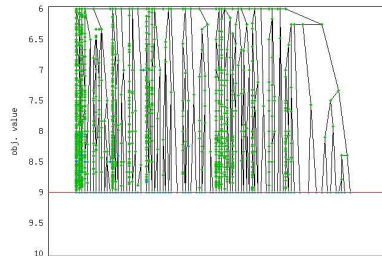
B8B tree (neos-598183.dat 3082s SYMPHONY)



B8B tree (bell13a.dat 296s SYMPHONY)



B8B tree (neos11.dat 6124s SYMPHONY)



# Outline

- 1 Introduction
- 2 What is Optimization?
- 3 Applications of Optimization
- 4 Basic Solution Framework
- 5 Advanced Methods
  - Decomposition
  - Parallelization
- 6 Software

# What is COIN-OR?

## The COIN-OR Foundation

- A **non-profit foundation** promoting the development and use of interoperable, open-source software for operations research.
- A **consortium** of researchers in both industry and academia dedicated to improving the state of computational research in OR.
- A **venue** for developing and maintaining standards.
- A **forum** for discussion and interaction between practitioners and researchers.

## The COIN-OR Repository

- A **collection** of interoperable software tools for building optimization codes, as well as a few stand alone packages.
- A **venue for peer review** of OR software tools.
- A **development platform** for open source projects, including a wide range of project management tools.

See [www.coin-or.org](http://www.coin-or.org) for more information.

# What You Can Do With COIN

- We currently have 40+ projects and more are being added all the time.
- Most projects are now licensed under the [EPL](#) (very permissive).
- COIN has solvers for most common optimization problem classes.
  - Linear programming
  - Nonlinear programming
  - Mixed integer linear programming
  - Mixed integer nonlinear programming (convex and nonconvex)
  - Stochastic linear programming
  - Semidefinite programming
  - Graph problems
  - Combinatorial problems (VRP, TSP, SPP, etc.)
- COIN has various utilities for reading/building/manipulating/preprocessing optimization models and getting them into solvers.
- COIN has overarching frameworks that support implementation of broad algorithm classes.
  - Parallel search
  - Branch and cut (and price)
  - Decomposition-based algorithms

# Brief Overview of SYMPHONY

- **SYMPHONY** is an open-source software package for solving and analyzing mixed-integer linear programs (MILPs).
- **SYMPHONY** can be used in three distinct modes.

- Black box solver: From the command line or shell.
- Callable library: From a C/C++ code.
- Framework: Develop a customized solver or callable library.

- Advanced features

- Warm starting
- Sensitivity analysis
- Bicriteria solve
- **Parallel execution**

## What's Available

- Available at [projects.coin-or.org/SYMPHONY](http://projects.coin-or.org/SYMPHONY).
- An extensive user's manual on-line and in PDF.
- A tutorial illustrating the development of a custom solver.
- Configuration and compilation files
- Examples and Applications

### SYMPHONY Solvers

- |                              |                            |
|------------------------------|----------------------------|
| • Generic MILP               | • Mixed Postman Problem    |
| • Multicriteria MILP         | • Set Partitioning Problem |
| • Multicriteria Knapsack     | • Matching Problem         |
| • Traveling Salesman Problem | • Network Routing          |
| • Vehicle Routing Problem    |                            |

- CHiPPS stands for COIN-OR High Performance Parallel Search.
- CHiPPS is a set of C++ class libraries for implementing **tree search** algorithms for both sequential and parallel environments.
- The basic goal is to generalize notions from SYMPHONY and enable large-scale computation.
- Available at [projects.coin-or.org/CHiPPS](http://projects.coin-or.org/CHiPPS)

## CHiPPS Components

### ALPS (Abstract Library for Parallel Search)

- is the search-handling layer (parallel and sequential).
- provides various search strategies based on node priorities.

### BiCePS (Branch, Constrain, and Price Software)

- is the data-handling layer for relaxation-based optimization.
- adds notion of **variables** and **constraints**.
- assumes iterative bounding process.

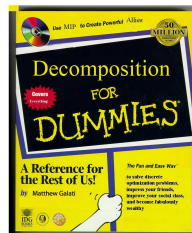
### BLIS (BiCePS Linear Integer Solver)

- is a concretization of BiCePS.
- specific to models with **linear** constraints and objective function.

## DIP Framework

**DIP** (Decomposition for Integer Programming) is an open-source software framework for implementing various decomposition methods.

- Built on the CHiPPS tree search engine.
- Emphasis on ease-of-use: minimal burden placed on the user.
- Allows direct comparison in one framework
- DIP abstracts the common, generic elements of these methods
- **Key:** The user defines any necessary methods in the space of the compact formulation, greatly simplifying the API
  - All that's required is to specify what constraints to relax in order to decompose the problem.
  - If desired, specialized methods for the relaxation, cut generation, branching, heuristics can be specified if desired.
- DIP handles all of the required reformulations, amppings, etc.
- There is a modeling-language front-end that is an extension to PuLP, a python-based modeling language.
- Available at [projects.coin-or.org/Dip](http://projects.coin-or.org/Dip)





# DIP Framework: Applications

Application	Description	$\mathcal{P}'$	$\text{OPT}(c)$	$\text{SEP}(x)$	Input
<b>AP3</b>	3-index assignment	AP	Jonker	user	user
<b>ATM</b>	cash management (SAS COE)	MILP(s)	CBC	CGL	user
<b>GAP</b>	generalized assignment	KP(s)	Pisinger	CGL	user
<b>MAD</b>	matrix decomposition	MaxClique	Cliquer	CGL	user
<b>MILP</b>	random partition into $A', A''$	MILP	CBC	CGL	mps
<b>MILPBlock</b>	user-defined blocks for $A'$	MILP(s)	CBC	CGL	mps, block
<b>MMKP</b>	multi-dim/choice knapsack	MCKP	Pisinger	CGL	user
		MDKP	CBC	CGL	user
<b>SILP</b>	intro example, tiny IP	MILP	CBC	CGL	user
<b>TSP</b>	traveling salesman problem	1-Tree	Boost	Concorde	user
		2-Match	CBC	Concorde	user
<b>VRP</b>	vehicle routing problem	$k$ -TSP	Concorde	CVRPSEP	user
		$b$ -Match	CBC	CVRPSEP	user

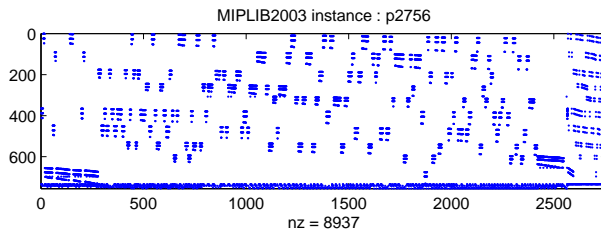
# MILPBlock: Decomposition-based MILP Solver

- Many difficult MILPs have a block structure, but this structure is not part of the input (MPS) or is not exploitable by the solver.
- In practice, it is common to have models composed of independent subsystems coupled by global constraints.
- The result may be models that are highly symmetric and difficult to solve using traditional methods, but would be easy to solve if the structure were known.

$$\begin{pmatrix} A_1'' & A_2'' & \cdots & A_\kappa'' \\ A_1' & & & \\ & A_2' & & \\ & & \ddots & \\ & & & A_\kappa' \end{pmatrix}$$

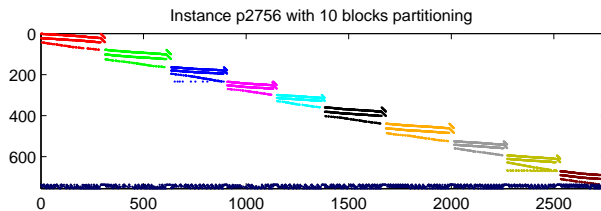
- MILPBlock provides a black-box solver for applying **integrated methods** to generic MILP
- Input is an MPS/LP and a *block file* specifying structure.
- Optionally, the block file can be automatically generated using the hypergraph partitioning algorithm of HMetis.

# Hidden Block Structure



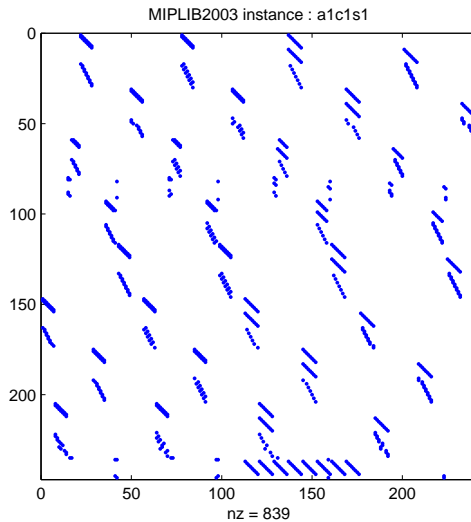
Detected block structure for  $p2756$  instance

# Hidden Block Structure



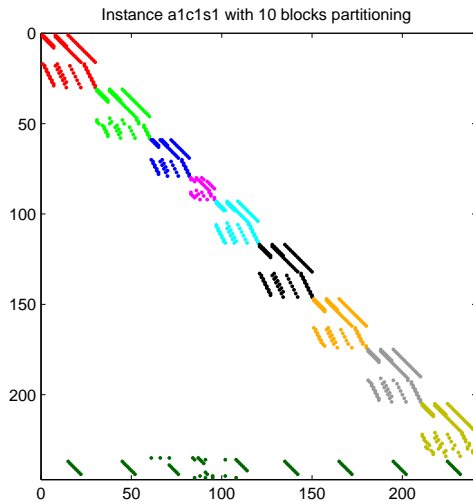
Detected block structure for  $p_{2756}$  instance

# Hidden Block Structure



Detected block structure for *a1c1s1* instance

# Hidden Block Structure



Detected block structure for *a1c1s1* instance

MibS is an open source solver for Bilevel integer programs built on top of the BLIS layer of CHiPPS. Features include

- Branch and Cut for IBLPs
  - *Bounding problems*
  - *Bilevel feasibility cuts*
  - Several *primal heuristics*
  - Simple *preprocessing*
- Specialized methods (primarily cuts) for specific problem classes
  - *Pure binary at the upper level*
  - *Interdiction problems*
- Standalone heuristics
  - *Greedy* method for interdiction problems
  - *Weighted sums* method for general problems
  - *Stationary point* method for general problems

MibS is available for download at

<http://coral.ie.lehigh.edu/projects/MibS>

Thanks!

That's It! Questions?

