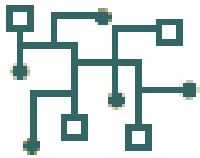
The Complexity of Search: What We Can Learn from Games

Ted Ralphs[1]

Joint work with Aykut Bulut[1], Scott DeNegre[3], Menal Güzelsoy[2], Anahita Hassanzadeh[1]

[1]COR@L Lab, Department of Industrial and Systems Engineering, Lehigh University
[2]SAS Institute, Advanced Analytics, Operations Research R & D
[3]The Chartis Group

CPAIOR, T.J. Watson Research Center, 18 May, 2013



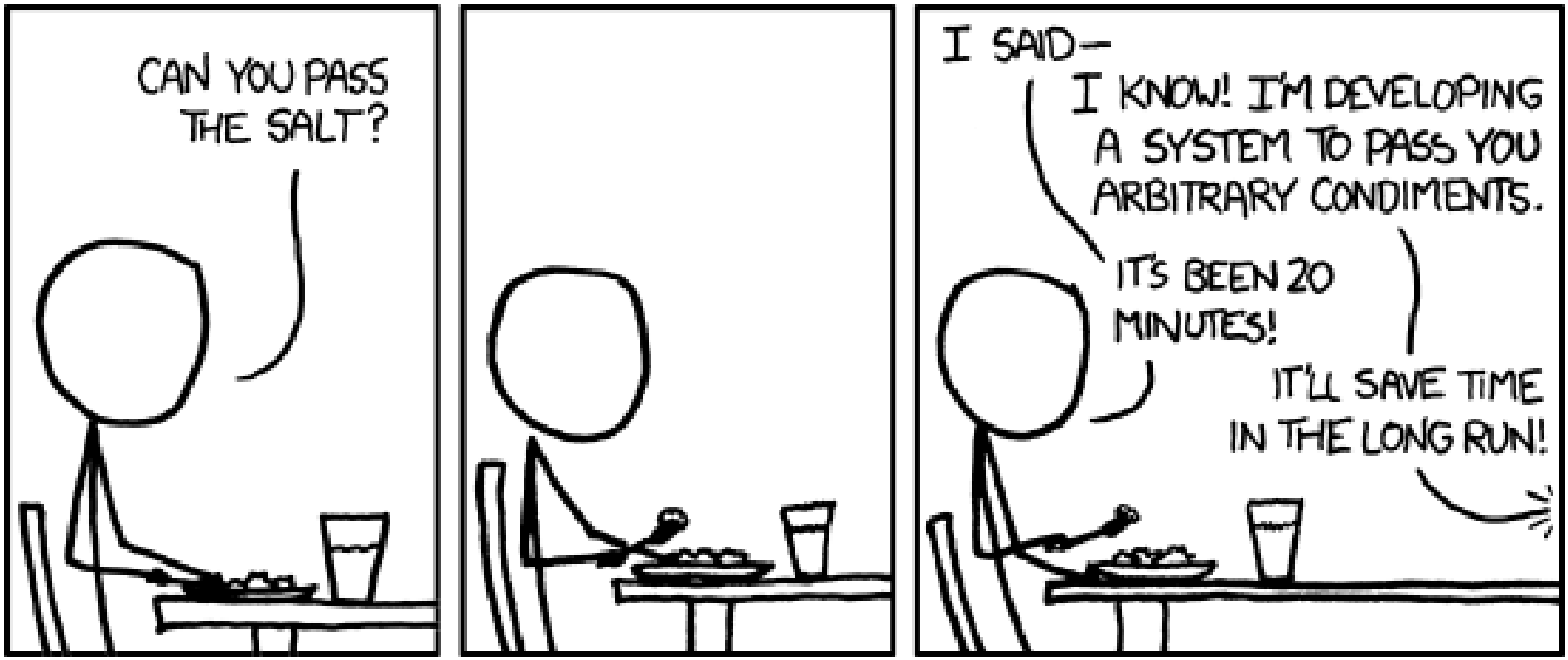
ISE Industrial and Systems Engineering

COR@L
COMPUTATIONAL OPTIMIZATION RESEARCH AT LEHIGH

Outline

Outline

Motivation


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

Tree Search

- The general class of algorithms we consider are *tree search algorithms*.
- Tree search algorithms systematically search the nodes of an acyclic graph for certain *goal nodes*.
- Most algorithms for solving NP-complete problems can be interpreted tree search algorithms.
  - Roughly speaking, this is because the possible execution paths of any computer program form a tree structure.
  - The nodes represent conditionals and the edges represent the resulting branches.

A Bit About Games

- We consider *finite extensive-form games*, which are sequential games involving n players.

Loose Definition

- The game is specified on a tree with each node corresponding to a move and the outgoing arcs specifying possible choices.
- The leaves of the tree have associated payoffs.
- Each player's goal is to maximize payoff.
- There may be *chance* players who play randomly according to a probability distribution and do not have payoffs (*stochastic games*).

- All players are rational and have perfect information.
- The problem faced by a player in determining the next move is a *multilevel/multistage* optimization problem.
- The move must be determined by taking into account the *responses of the other players*.

Multilevel and Multistage Games

- We use the term *multilevel* for competitive games in which there is no chance player.
- We use the term *multistage* for cooperative games in which all players receive the same payoff, but there are chance players.
- A *subgame* is the part of a game that remains after some moves have been made.

Stackelberg Game

- A Stackelberg game is a game with two players who make one move each.
- The goal is to find a *subgame perfect Nash equilibrium*, i.e., the move by each player that ensures that player's best outcome.

Recourse Game

- A cooperative game in which play alternates between cooperating players and chance players.
- The goal is to find a *subgame perfect Markov equilibrium*, i.e., the move that ensures the best outcome in a probabilistic sense.

Multilevel and Multistage Optimization

- A standard mathematical program models a (set of) decision(s) to be made *simultaneously* by a *single* decision-maker (i.e., with a *single* objective).
- Decision problems arising in sequential games and other real-world applications involve
  - multiple, independent decision-makers (DMs),
  - sequential/multi-stage decision processes, and/or
  - multiple, possibly conflicting objectives.
- Modeling frameworks
  - Multiobjective Programming $\Leftarrow$ multiple objectives, single DM
  - Mathematical Programming with Recourse $\Leftarrow$ multiple stages, single DM
  - Multilevel Programming $\Leftarrow$ multiple stages, multiple objectives, multiple DMs
- *Multilevel programming* generalizes standard mathematical programming by modeling hierarchical decision problems, such as finite extensive-form games.
- Such models arises in a remarkably wide array of applications.

Connection to Search

- Multilevel structure is inherent in many decision problems that occur *within* search algorithms.
- We would like to make the "most effective" algorithmic choice at each step, taking into account the effect of the choice on future iterations.
- The choice problem is an optimization problem that itself may have a multilevel structure similar to that of a multi-round game.
- Multilevel choice problems arise when the effectiveness or validity of the choice is evaluated by solving another optimization problem.
- The number of levels one chooses to "look ahead" determines the complexity of the exact version of the problem.
- Examples
  - Constructing a valid inequality for a given class that maximizes degree of violation.
  - Choosing a branching disjunction that achieves maximal bound improvement.

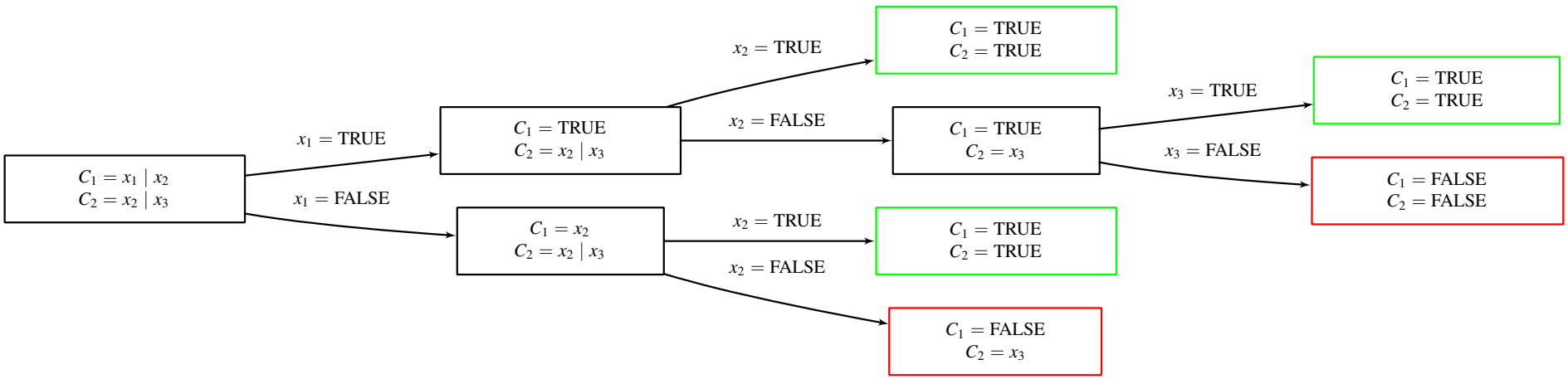
Outline

A Canonical Example: Satisfiability Game

- A canonical extensive-form game that illustrates many of the basic principles is the *k-player satisfiability game*.
  - k players determine the value of a set of Boolean variables with each in control of a specific subset.
  - In round i, player i determines the values of her variables.
  - Each player tries to choose values that force a certain end result, given that subsequent players may be trying to achieve the opposite result.
- Examples
  - k = 1: SAT
  - k = 2: The first player tries to choose values such that any choice by the second player will result in satisfaction.
  - k = 3: The first player tries to choose values such that the second player cannot choose values that will leave the third player without the ability to find satisfying values.
- Note that the odd players and the even players are essentially "working together" and the same game can be described with only two players.

A Simple SAT Example

- This diagram illustrates the search for solutions to the problem as a tree.
- The nodes in green represent settings of the truth values that satisfy all the given clauses; red represents non-satisfying truth values.
  - With one player, the solution is any path to one of the green nodes.
  - With two players, the solution is a subtree in which there are no red nodes.
- The latter requires knowledge of *all* leaf nodes (important!).



More Formally

- More formally, we are given a Boolean formula with variables partitioned into k sets $X_1, \ldots, X_k$.
- For k odd, the SAT game can be formulated as

$$\exists X_1 \forall X_2 \exists X_3 \ldots ? X_k \tag{1}$$

- for even k, we have

$$\forall X_1 \exists X_2 \forall X_3 \ldots ? X_k \tag{2}$$

- A more general form of this problem, known as the *quantified Boolean formula problem* (QBF) allows an arbitrary sequence of quantifiers.

From SAT Game to Multilevel Optimization

- For $k = 1$, SAT can be formulated as the (feasibility) integer program

$$\exists x \in \{0,1\}^n : \sum_{i \in C_j^0} x_i + \sum_{i \in C_j^1} (1 - x_i) \geq 1 \ \forall j \in J. \qquad \text{(SAT)}$$

- (SAT) can be formulated as the optimization problem

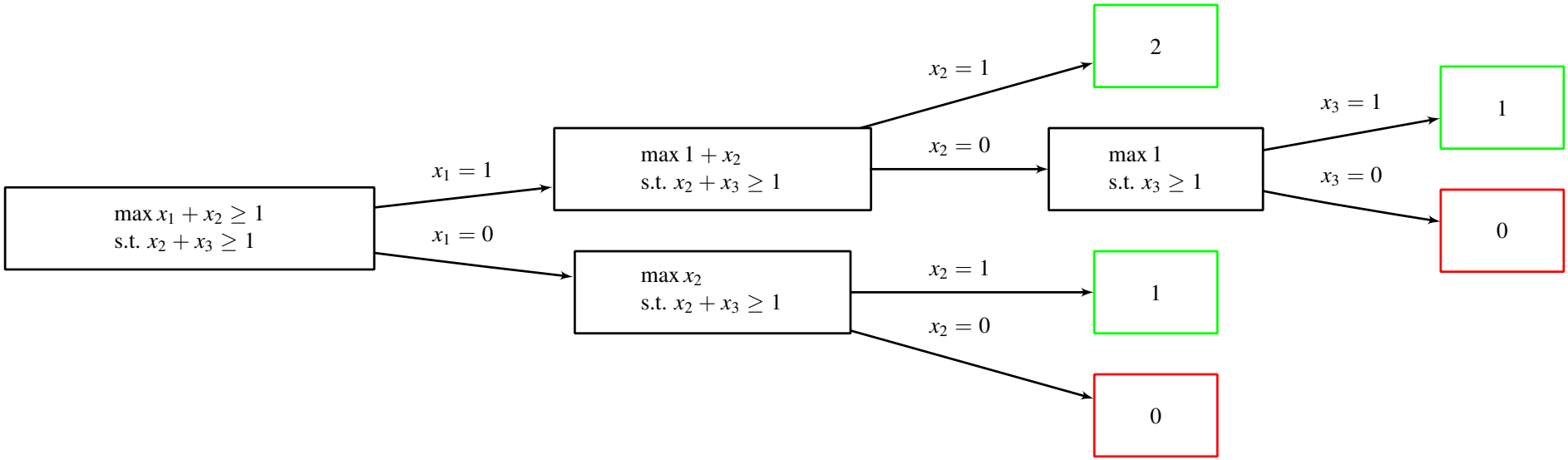
$$\max_{x \in \{0,1\}^n} \sum_{i \in C_0^0} x_i + \sum_{i \in C_0^1} (1 - x_i)$$
$$\text{s.t.} \sum_{i \in C_j^0} x_i + \sum_{i \in C_j^1} (1 - x_i) \geq 1 \ \forall j \in J \setminus \{0\}$$

- For $k = 2$, we then have

$$\max_{x_{I_1} \in \{0,1\}^{I_1}} \min_{x_{I_2} \in \{0,1\}^{I_2}} \sum_{i \in C_0^0} x_i + \sum_{i \in C_0^1} (1 - x_i)$$
$$\text{s.t.} \sum_{i \in C_j^0} x_i + \sum_{i \in C_j^1} (1 - x_i) \geq 1 \ \forall j \in J \setminus \{0\}$$

Branch and Bound for Optimization Version of SAT

- Consider the earlier example of the SAT game, now as an optimization problem.
- In the one player version, the goal is simply to maximize payoff.
- The two player game is zero-sum with the first player attempting to maximize while the second player attempts to minimize.
- The complexity of the two-player game comes from the requirement to account for the payoff at *all* leaf nodes.



How Difficult is the SAT Game?

- Fundamentally, we would like to know how difficult it is to solve player one's decision problem.
- It is well-known that the (single player) satisfiability problem is is in the complexity class *NP*-complete.
- It is perhaps to be expected that the k-player satisfiability game is in a different class.
  - The k^{th} player to move is faced with a satisfiability problem.
  - The $(k-1)^{th}$ player is faced with a 2-player subgame in which she must take into account the move of the k^{th} player.
  - And so on . . .
- Each player's decision problem appears to be exponentially more difficult than the succeeding player's problem.
- This complexity is captured formally in the hierarchy of complexity classes known as the *polynomial time hierarchy*.

Outline

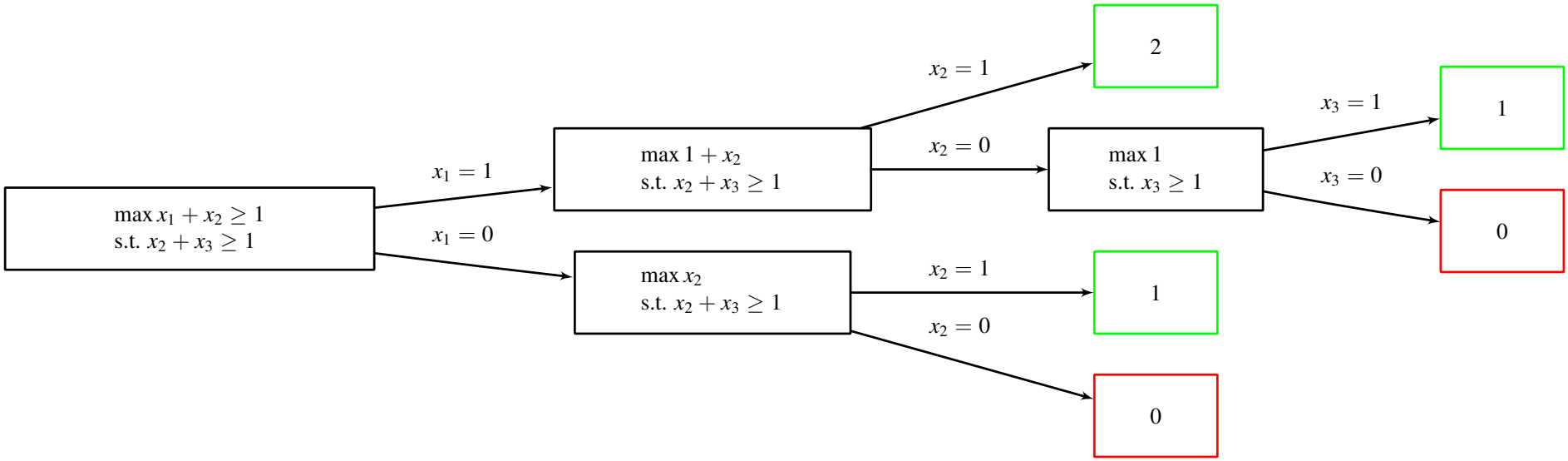
Complexity: Basic Notions

- The formal complexity framework traditionally employed in discrete optimization applies to *decision problems* (Garey and Johnson, 1979).
- The formal model of computation is a *deterministic Turing machine* (DTM).
  - A DTM specifies an *algorithm* computing the value of a Boolean function.
  - The DTM executes a program, reading the input from a *tape*.
  - We equate a given DTM with the program it executes.
  - The output is YES or NO.
  - A YES answer is returned if the machine reaches an *accepting state*.
- A problem is specified in the form of a *language*, defined to be the subset of the possible inputs over a given *alphabet* (Γ) that are expected to output YES.
- A DTM that produces the correct output for inputs w.r.t. a given language is said to *recognize the language*.
- Informally, we can then say that the DTM represents an "algorithm that solves the given problem correctly."

Non-deterministic Turing Machines

- The possible execution paths of a DTM can be thought of as forming a tree.
- For problems that are efficiently solvable, we know how to construct an execution path that is guaranteed to end in an accepting state.
- For more difficult problems, some enumeration is needed.
- A *non-deterministic Turing machine* (NDTM) can be thought of as a Turing machine with an infinite number of parallel processors.
- An NDTM follows all possible execution paths simultaneously.
- It returns YES if an accepting state is reached on *any* path.
- The running time of an NDTM is the *minimum* running time (length) of any execution paths that end in an accepting state.
- The "running time" is the minimum time required to verify that some path (given as input) leads to an accepting state.

Back to SAT


max $x_1 + x_2 \geq 1$
s.t. $x_2 + x_3 \geq 1$
$x_1 = 1$
$x_1 = 0$
max $1 + x_2$
s.t. $x_2 + x_3 \geq 1$
max x_2
s.t. $x_2 + x_3 \geq 1$
$x_2 = 1$
$x_2 = 0$
$x_2 = 1$
$x_2 = 0$
max 1
s.t. $x_3 \geq 1$
$x_3 = 1$
$x_3 = 0$
2
1
0
1
0

Primitive Complexity Classes

- Languages can be grouped into *classes* based on the *best worst-case running time* of any TM that recognizes the language.
  - The class P is the set of all languages for which there exists a DTM that recognizes the language in time polynomial in the length of the input.
  - The class NP is the set of all languages for which there exists an NDTM that recognizes the language in time polynomial in the length of the input.
  - The class $coNP$ is the set of languages whose complements are in NP.
  - Additional classes can be formed hierarchically by the use of *oracles*.
- A language L_1 can be *reduced* to a language L_2 if there is an output-preserving polynomial transformation of members of L_1 to members of L_2.
- A language L is said to be *complete* for a class if all languages in the class can be reduced to L.
- We are primarily talking here about time complexity, though space complexity must ultimately also be considered.

Outline

The Polynomial Hierarchy

The polynomial hierarchy is a scheme for classifying multi-level and multi-stage decision problems. We have

$$\Delta_0^p := \Sigma_0^p := \Pi_0^p := P, \tag{3}$$

where P is the set of decision problems that can be solved in polynomial time. Higher levels are defined recursively as:

$$\begin{aligned} \Delta_{k+1}^p &:= P^{\Sigma_k^p}, \\ \Sigma_{k+1}^p &:= NP^{\Sigma_k^p}, \textit{and} \\ \Pi_{k+1}^p &:= coNP^{\Sigma_k^p}. \end{aligned}$$

PH is the union of all levels of the hierarchy.

Collapsing the Hierarchy

In general, we have

$$\Sigma_0^p \subseteq \Sigma_1^p \subseteq \ldots \Sigma_k^p \subseteq \ldots$$
$$\Pi_0^p \subseteq \Pi_1^p \subseteq \ldots \Pi_k^p \subseteq \ldots$$
$$\Delta_0^p \subseteq \Delta_1^p \subseteq \ldots \Delta_k^p \subseteq \ldots$$

It is not known whether any of the inclusions are strict. We do have that

$$(\Sigma_k^p = \Sigma_{k+1}^p) \Rightarrow \Sigma_k^p = \Sigma_j^p \ \forall j \geq k \tag{4}$$

In particular, if $P = NP$, then every problem in the PH is solvable in polynomial time. Similar results hold for the Π and Δ hierarchies.

Complexity of Multilevel Games and Optimization

- The satisfiability games with k players is complete for Σ_k^p.
- For the corresponding k-level optimization problem, the optimal value is one if and only if the first player has a winning strategy.
- This means the satisfiability game can be reduced to the (decision) problem of whether the optimal value ≥ 1?
- Thus, the (the decision version of) k-level mixed integer programming is also complete for Σ_k^p.
- By swapping the "min" and the "max," we can get a similar decision problem that is complete for Π_k^p.

$$\min_{x_{N_1} \in \{0,1\}^{N_1}} \max_{x_{N_2} \in \{0,1\}^{N_2}} \sum_{i \in C_0^0} x_i + \sum_{i \in C_0^1} (1 - x_i)$$
$$\text{s.t.} \sum_{i \in C_j^0} x_i + \sum_{i \in C_j^1} (1 - x_i) \geq 1 \ \forall j \in J \setminus \{0\}$$

- The question remains whether the optimal value is ≥ 1, but now we are asking it with respect to a minimization problem.

Outline

(Standard) Mixed Integer Linear Programs

- In parts of the talk, we will need to consider a (standard) *mixed integer linear program* (MILP).
- To simplify matters, when we discuss a standard MILP, it will be of the form

MILP

$$\min\{c^\top x \mid x \in \mathcal{P} \cap (\mathbb{Z}^p \times \mathbb{R}^{n-p})\}, \qquad \text{(MILP)}$$

where $\mathcal{P} = \{x \in \mathbb{R}^n_+ \mid Ax = b\}$, $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$.

Bilevel (Integer) Linear Programming

Formally, a *bilevel linear program* is described as follows.

- $x \in X \subseteq \mathbb{R}^{n_1}$ are the *upper-level variables*
- $y \in Y \subseteq \mathbb{R}^{n_2}$ are the *lower-level variables*

Bilevel (Integer) Linear Program

$$\max \left\{ c^1 x + d^1 y \mid x \in \mathcal{P}_U \cap X, y \in \operatorname{argmin}\{ d^2 y \mid y \in \mathcal{P}_L(x) \cap Y \} \right\} \quad \text{(MIBLP)}$$

The *upper-* and *lower-level feasible regions* are:

$$\mathcal{P}_U = \left\{ x \in \mathbb{R}_+ \mid A^1 x \leq b^1 \right\} \text{ and}$$
$$\mathcal{P}_L(x) = \left\{ y \in \mathbb{R}_+ \mid G^2 y \geq b^2 - A^2 x \right\}.$$

We consider the general case in which $X = \mathbb{Z}^{p_1} \times \mathbb{R}^{n_1 - p_1}$ and $Y = \mathbb{Z}^{p_2} \times \mathbb{R}^{n_2 - p_2}$.

Recourse Problems

- If $d^1 = -d^2$, we can view this as a *mathematical program with recourse*.
- We can reformulate the bilevel program as follows.

$$\min\{-c^1 x + Q(x) \mid x \in \mathcal{P}_U \cap X\}, \tag{5}$$

where

$$Q(x) = \min\{d^1 y \mid y \in \mathcal{P}_L(x) \cap Y\}. \tag{6}$$

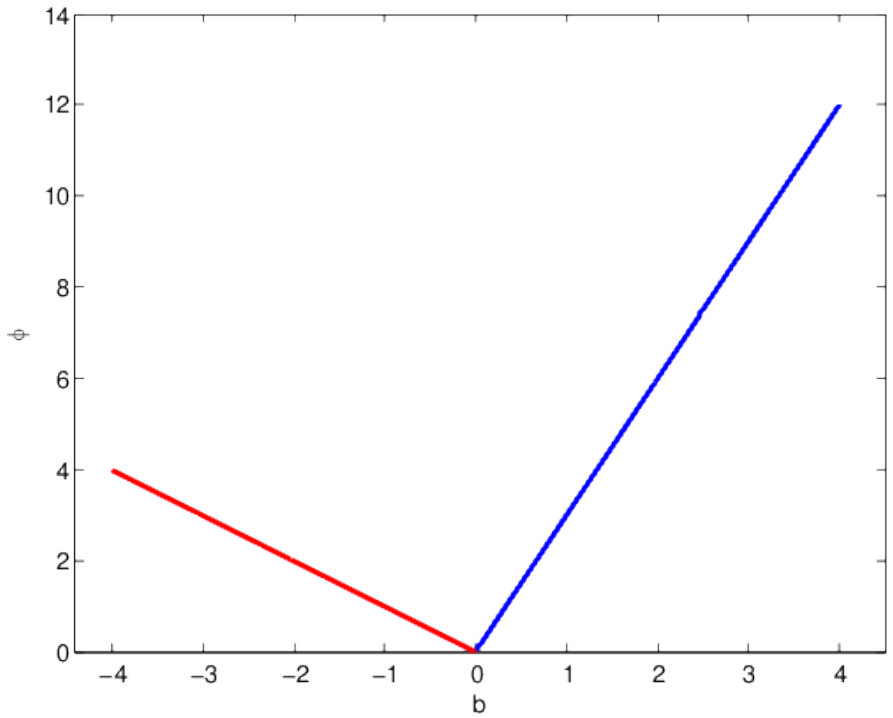
- The function Q is known as the *value function* of the recourse problem.

Outline

LP Value Function

Example

$$\phi_{LP}(b) = \min\ 6x_1 + 7x_2 + 5x_3$$
$$\text{s.t. } 2x_1 - 7x_2 + x_3 = b \qquad \text{(Ex.LP)}$$
$$x_1, x_2, x_3 \in \mathbb{R}_+$$



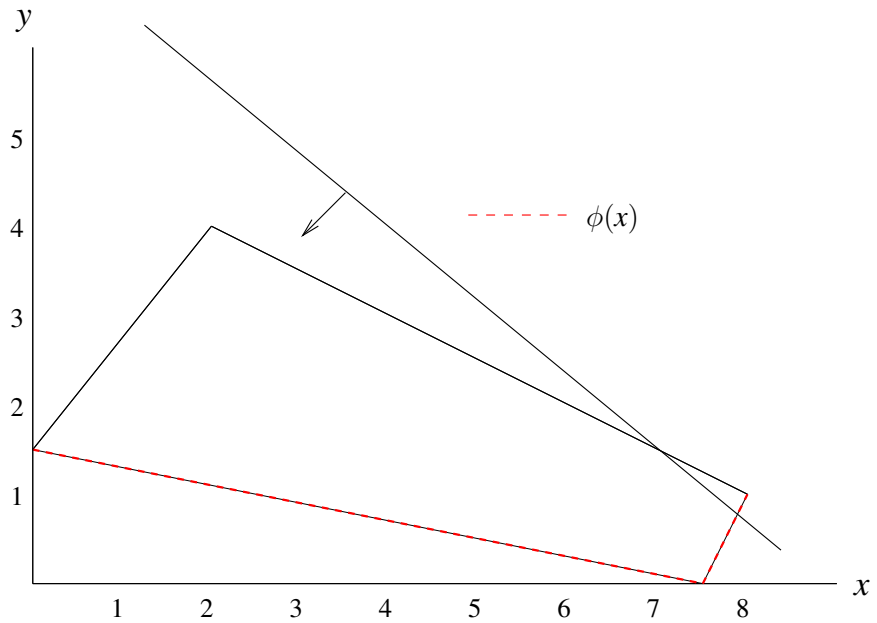
Benders' Principle (Linear Programming)

$$z_{\text{LP}} = \min_{(x,y)\in\mathbb{R}^n} \left\{ c'x + c''y \;\middle|\; A'x + A''y \geq b \right\}$$

$$= \min_{x\in\mathbb{R}^{n'}} \left\{ c'x + \phi(b - A'x) \right\},$$

where

$$\phi(d) = \min\ c''y$$
$$\text{s.t. } A''y \geq d$$
$$y \in \mathbb{R}^{n''}$$



Basic Strategy:

- The function ϕ is the *value function* of a linear program.
- The value function is piecewise linear and convex.
- We iteratively generate a lower approximation by sampling the domain.

MILP Value Function

Now we consider the MILP value function $\phi : \mathbb{R}^m \to \mathbb{R} \cup \{\pm\infty\}$

$$\begin{aligned} \phi(b) = \min\ & c^\top x \\ \text{s.t. } & Ax = b \\ & x \in \mathbb{Z}^r_+ \times \mathbb{R}^{n-r}_+ \end{aligned} \tag{MILP}$$

We define

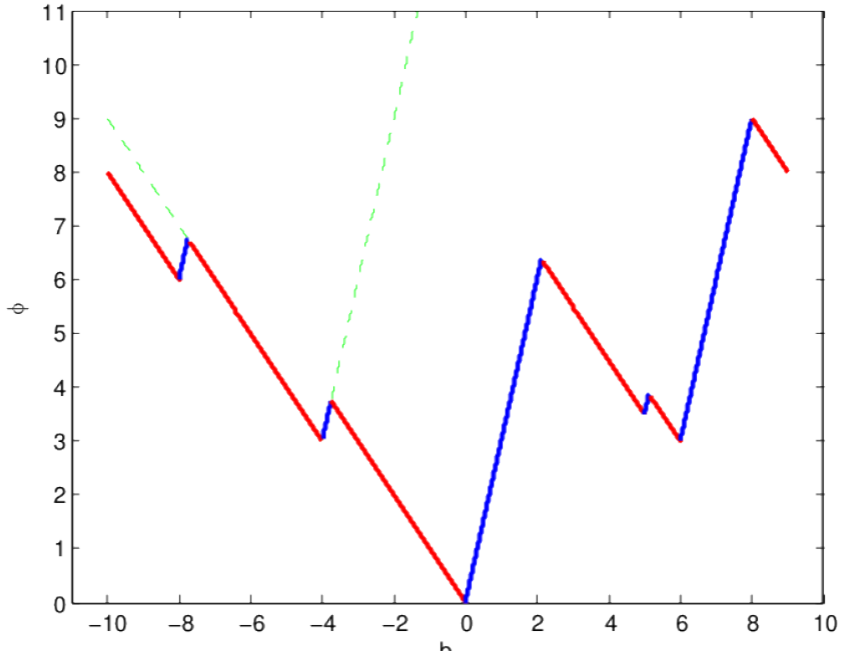
- $\mathcal{S}(b) = \{x \in \mathbb{Z}^r_+ \times \mathbb{R}^{n-r}_+ \mid Ax = b\}$.
- $B = \{b \in \mathbb{R}^m \mid S(b) \neq \emptyset\}$.

Example: MILP Value Function

The value function of a MILP is non-convex and discontinuous piecewise polyhedral.

Example

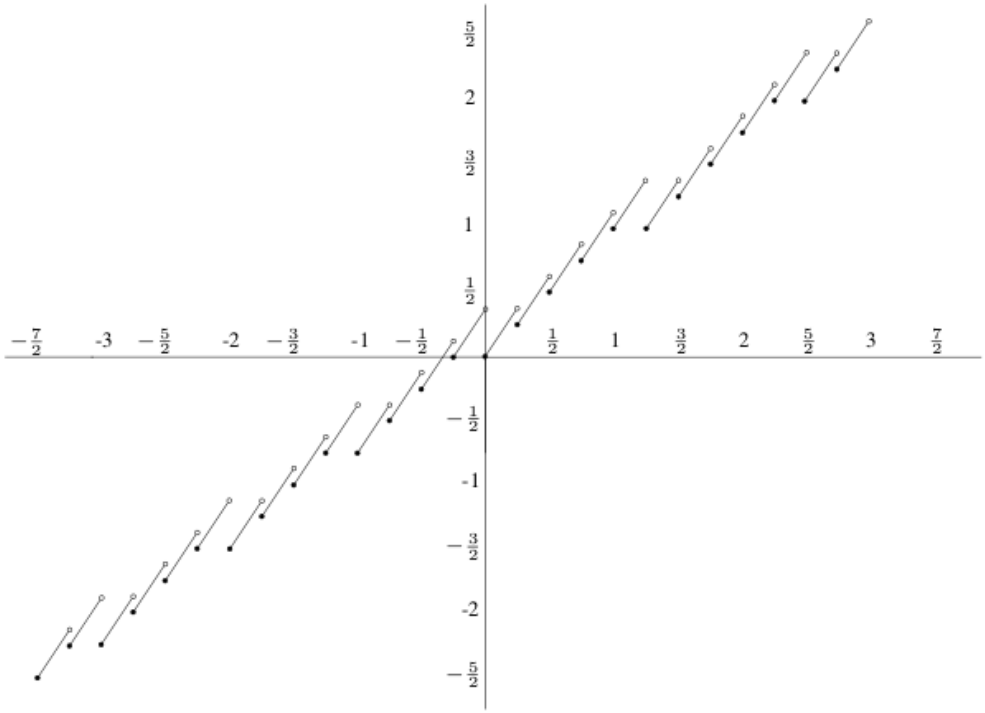
$$\phi(d) = \min\ 3x_1 + \frac{7}{2}x_2 + 3x_3 + 6x_4 + 7x_5 + 5x_6$$
$$\text{s.t. } 6x_1 + 5x_2 - 4x_3 + 2x_4 - 7x_5 + x_6 = d$$
$$x_1, x_2, x_3 \in \mathbb{Z}_+,\ x_4, x_5, x_6 \in \mathbb{R}_+$$



Example: MILP Value Function

Example

$$\phi(b) = \min\ x_1 - \frac{3}{4}x_2 + \frac{3}{4}x_3$$
$$\text{s.t. } \frac{5}{4}x_1 - x_2 + \frac{1}{2}x_3 = b \qquad \text{(Ex2.MILP)}$$
$$x_1, x_2 \in \mathbb{Z}_+,\ x_3 \in \mathbb{R}_+$$



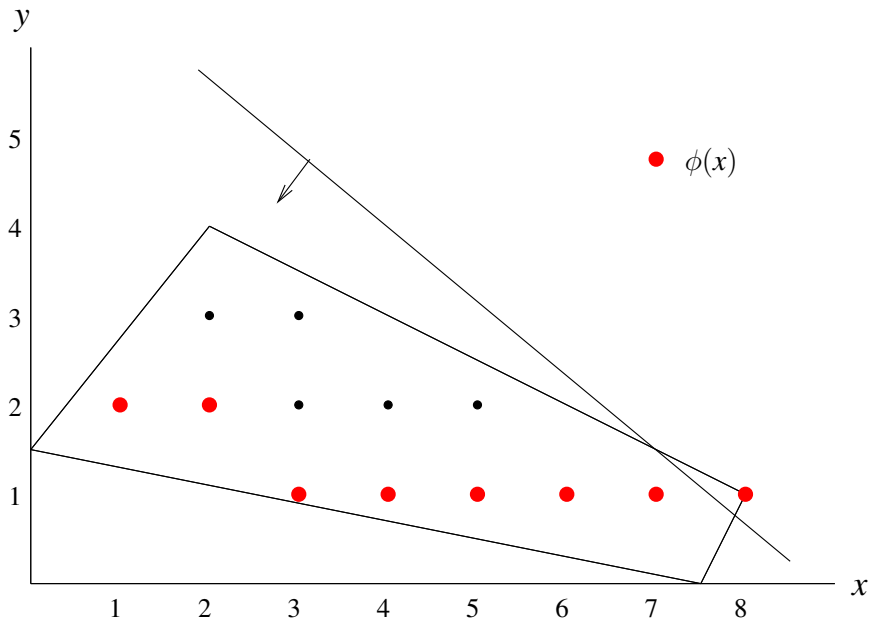
Benders' Principle (Integer Programming)

$$z_{IP} = \min_{(x,y)\in\mathbb{Z}^n} \left\{ c'x + c''y \;\middle|\; A'x + A''y \geq b \right\}$$

$$= \min_{x\in\mathbb{R}^{n'}} \left\{ c'x + \phi(b - A'x) \right\},$$

where

$$\phi(d) = \min\ c''y$$
$$\text{s.t. } A''y \geq d$$
$$y \in \mathbb{Z}^{n''}$$



Basic Strategy:

- Here, ϕ is the value function of an *integer program*.
- In the general case, the function ϕ is piecewise linear but not convex.
- Here, we also iteratively generate a lower approximation by evaluating ϕ.

Outline

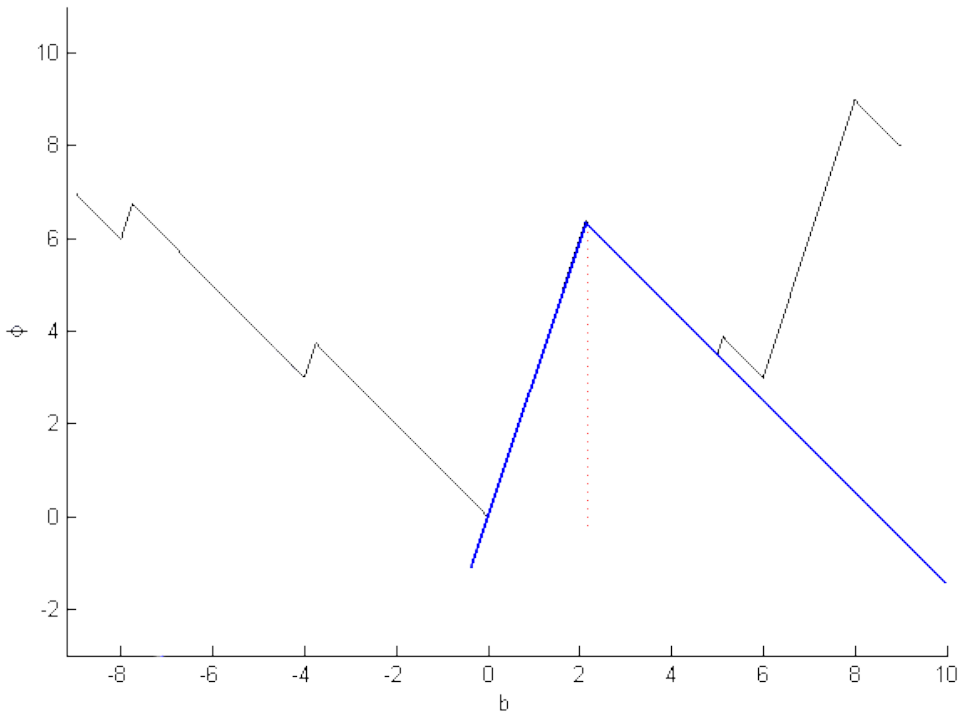
Dual Functions

A *dual function* $\varphi : \mathbb{R}^m \to \mathbb{R} \cup \{\pm\infty\}$ is and function such that

$$\varphi(b) \leq \phi(b) \quad \forall b \in \Lambda$$

For a particular value of $\hat{b}$, the dual problem is

$$\phi_D = \max\{\varphi(\hat{b}) : \varphi(b) \leq \phi(b) \quad \forall b \in \mathbb{R}^m,\ \varphi : \mathbb{R}^m \to \mathbb{R} \cup \{\pm\infty\}\}$$



Dual Functions from Branch-and-Bound

Let T be set of the terminating nodes of the tree. Then in a terminating node $t \in T$ we solve:

$$\begin{aligned} &\min c^\top x \\ &\text{s.t. } Ax = b, \\ &\quad l^t \leq x \leq u^t, x \geq 0 \end{aligned} \tag{7}$$

The dual at node t:

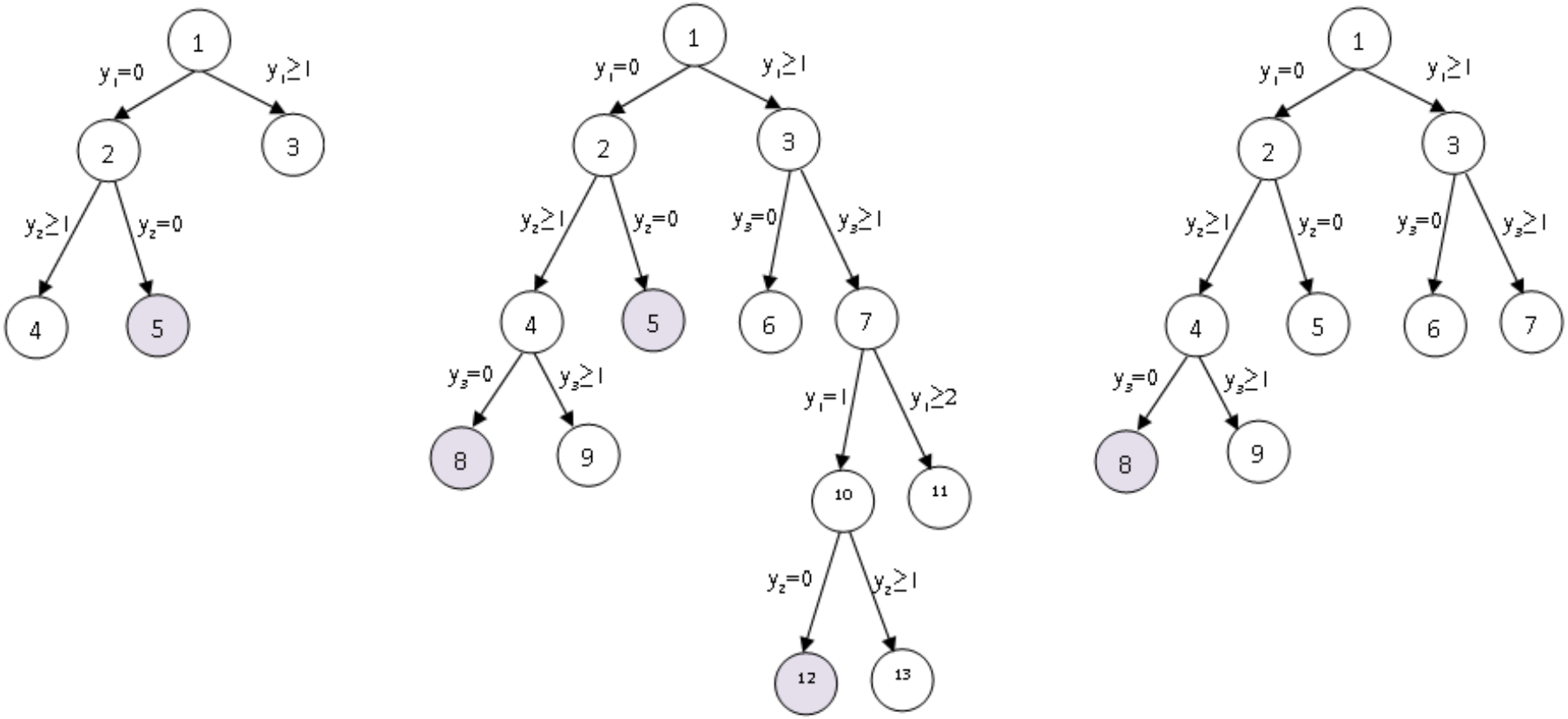
$$\begin{aligned} &\max \{\pi^t b + \underline{\pi}^t l^t + \bar{\pi}^t u^t\} \\ &\text{s.t. } \pi^t A + \underline{\pi}^t + \bar{\pi}^t \leq c^\top \\ &\quad \underline{\pi} \geq 0, \bar{\pi} \leq 0 \end{aligned} \tag{8}$$

We obtain the following strong dual function:

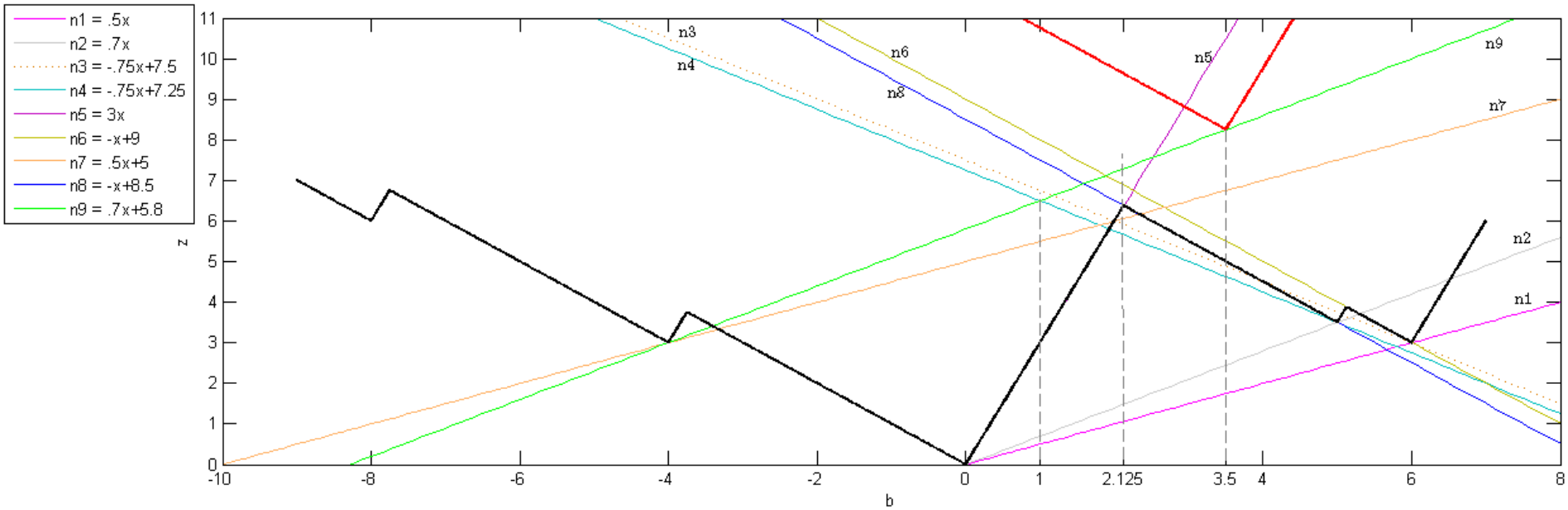
$$\min_{t \in T} \{\pi^t b + \underline{\pi}^t l^t + \bar{\pi}^t u^t\} \tag{9}$$

MILP Duals from Branch-and-Bound

Figure: Dual Functions from B&B for right hand sides 1, 2.125, 3.5



MILP Duals from Branch-and-Bound



Example

Consider

$$\begin{aligned} \min f(x) = \min \; & -3x_1 - 4x_2 + \sum_{s=1}^{2} 0.5Q(x,s) \\ & \text{s.t. } x_1 + x_2 \leq 5 \\ & x \in \mathbb{Z}_+ \end{aligned} \tag{10}$$

where

$$\begin{aligned} Q(x,s) = \min \; & 3y_1 + \frac{7}{2}y_2 + 3y_3 + 6y_4 + 7y_5 \\ & \text{s.t. } 6y_1 + 5y_2 - 4y_3 + 2y_4 - 7y_5 = h(s) - 2x_1 - \frac{1}{2}x_2 \\ & y_1, y_2, y_3 \in \mathbb{Z}_+, \; y_4, y_5 \in \mathbb{R}_+ \end{aligned} \tag{11}$$

with $h(s) \in \{-4, 10\}$.

Example

Iteration 1

Step 0

- $\mathcal{F} = \emptyset$
- $k = 1$.
- Solve

$$\begin{aligned} \min f(x) = \min \; & -3x_1 - 4x_2 \\ \text{s.t. } & x_1 + x_2 \leq 5 \\ & x_1, x_2 \in \mathbb{Z}_+ \end{aligned}$$

$f^0 = 20, x_1^* = 0, x_2^* = 5, \beta^1 = \frac{5}{2}$

Example

Step 1

- Solve the second stage problem for each scenario. That is, with $h(1) - \beta^1 = -6.5$ and $h(2) - \beta^1 = 7.5$.
- The respective dual functions are

  $F^1_{s=1}(\beta) = \min\{-\beta - 1, 0.5\beta + 10\}$ and
  $F^1_{s=2}(\beta) = \min\{3\beta - 15, -0.75\beta + 14.5\}$.

  Then, $\mathcal{F}(\beta) = \max\{F^1_{s=1}, F^1_{s=2}\}$.

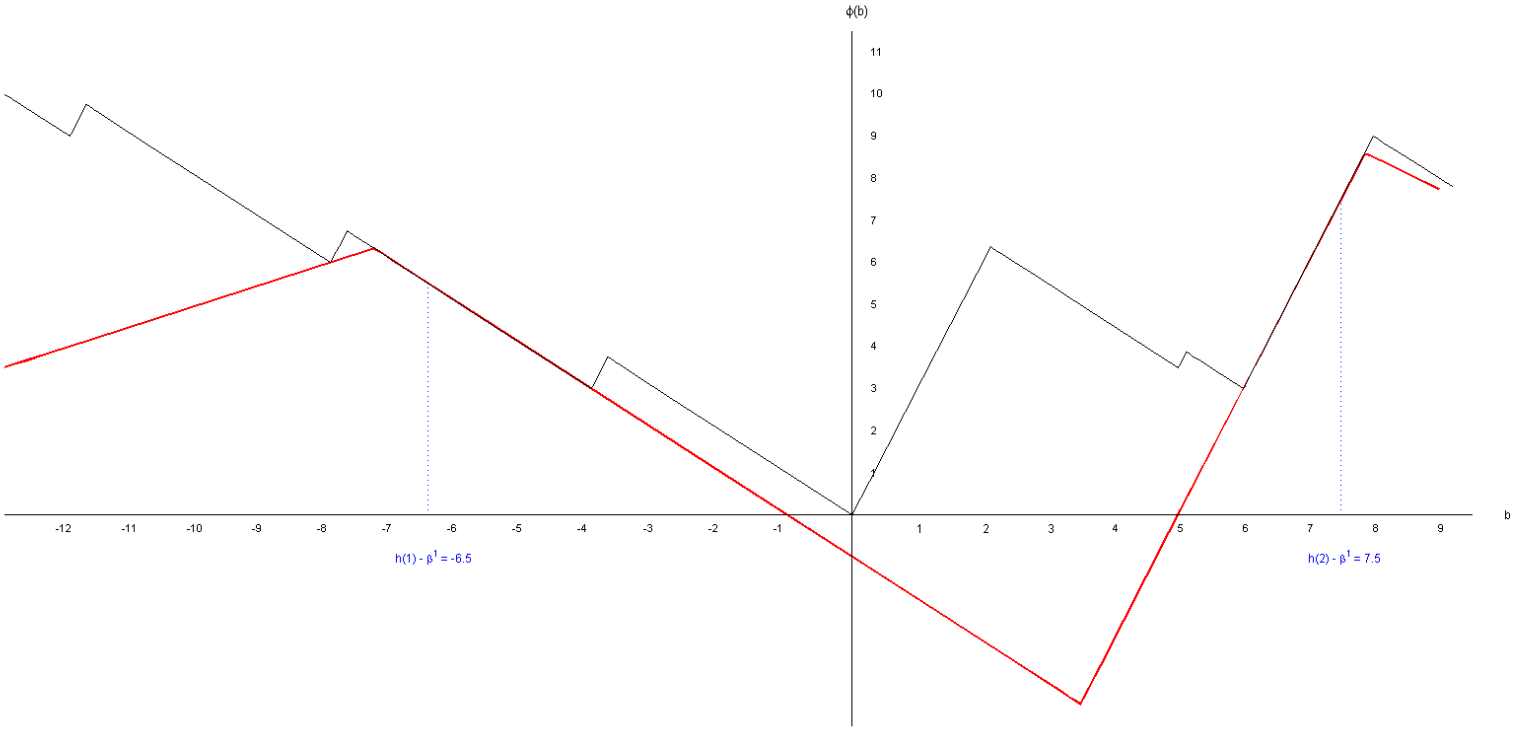
Step 2

- Solve the master problem

$$\begin{aligned} f^1 = \min \; & -3x_1 - 4x_2 + 0.5(\mathcal{F}_s(-4-\beta) + \mathcal{F}_s(10-\beta)) \\ \text{s.t. } & x_1 + x_2 \leq 5 \\ & 2x_1 + \frac{1}{2}x_2 = \beta \\ & x_1, x_2 \in \mathbb{Z}_+ \end{aligned}$$

- The solution to the master problem is $f^1 = -16.75$ with $\beta^1 = 7$.

Example


ϕ(b)
11
10
9
8
7
6
5
4
3
2
1
-12
-11
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6
7
8
9
b
h(1) - β1 = -6.5
h(2) - β1 = 7.5

Example

Iteration 2

Step 1

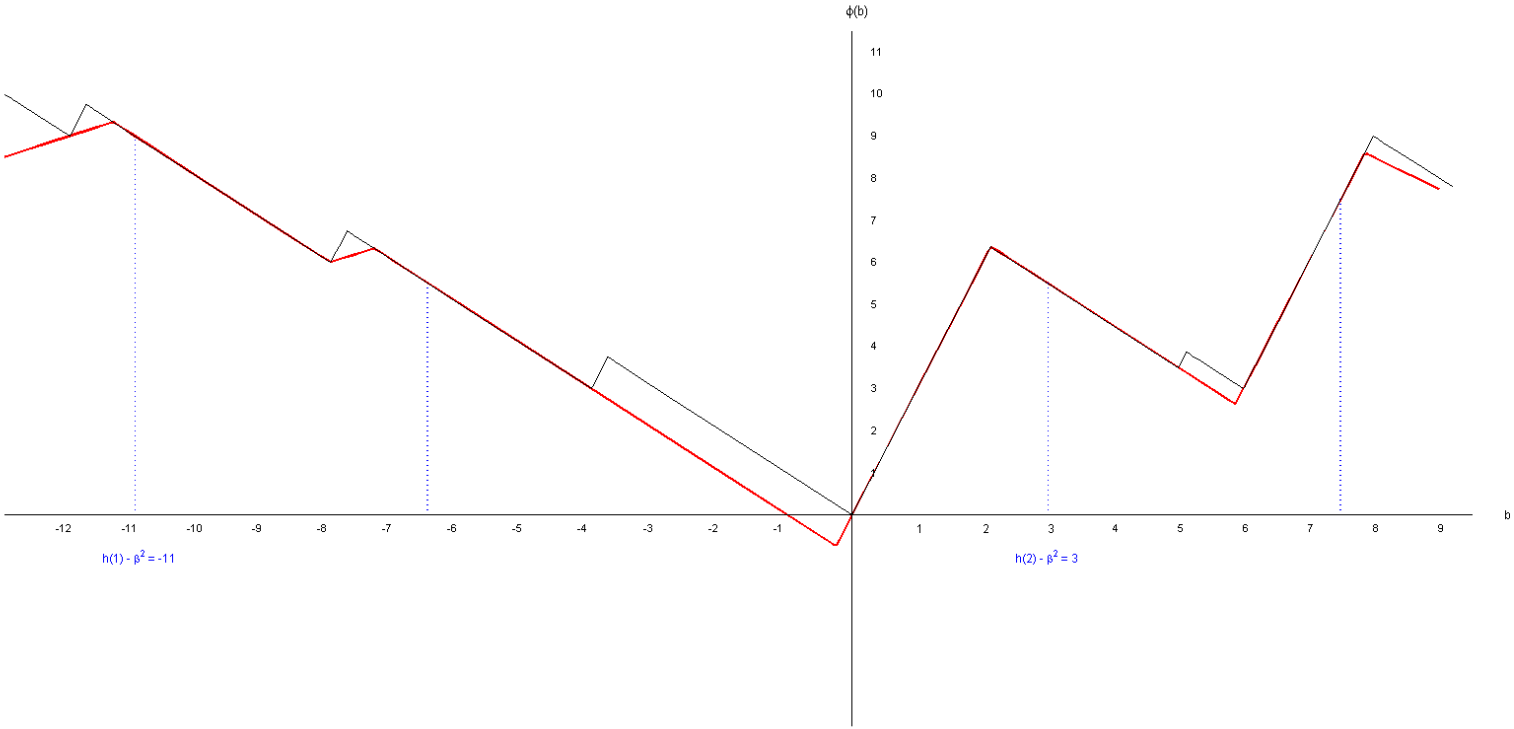
- Solve the second stage problem with right hand sides: -11 and 3.
- The respective dual functions are: $F^2_{s=1}(\beta) = \min\{-\beta - 2, 0.5\beta + 15\}$ and $F^2_{s=2}(\beta) = \min\{3\beta, -\beta + 8.5, 0.7\beta + 5.8\}$.

- Since $\mathcal{F}(-11) + \mathcal{F}(3) < F^2_{s=1}(-11) + F^2_{s=2}(3)$, we continue:
- Update $\mathcal{F}(\beta) = \max\{F^1_{s=1}, F^1_{s=2}, F^2_{s=1}, F^2_{s=2}\}$.

Step 2

- Solve the updated master problem. We obtain $f^2 = -14.5$ with $\beta^2 = 4$.

Example


ϕ(b)
11
10
9
8
7
6
5
4
3
2
1
-12
-11
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6
7
8
9
b
h(1) - β² = -11
h(2) - β² = 3

Example

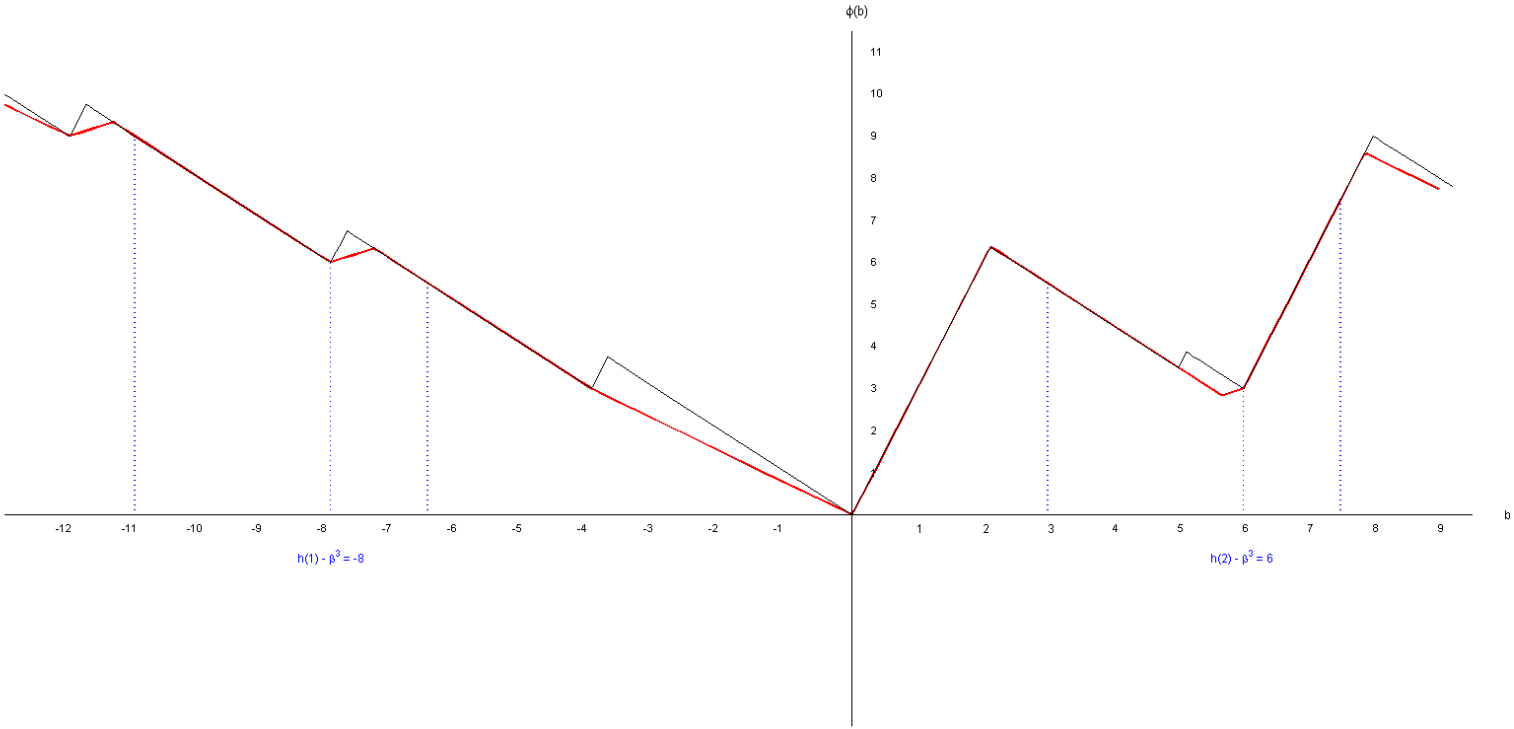
Iteration 3

Step 1

- Solve the second stage problem with right hand sides: -8 and 6.
- The respective dual functions are:
  $F^3_{s=1}(\beta) = -0.75\beta$ and $F^3_{s=2}(\beta) = 0.5\beta$.

- $\mathcal{F}(-8) + \mathcal{F}(6) = F^3_{s=1}(-8) + F^3_{s=2}(6) = 9$, the approximation is exact and the optimal solution to the problem is $f^3 = -14.5$ and $\beta^3 = 4$.

Example


ϕ(b)
11
10
9
8
7
6
5
4
3
2
1
-12
-11
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6
7
8
9
b
h(1) - β3 = -8
h(2) - β3 = 6

Outline

The Value Function in Branch and Bound

- Note that the value function and Benders' Principle is implicit in standard branch and bound for a single-level problem.
- Each time we branch, we change the bound of one of the variables in each child.
- This can be interpreted as either fixing the value of a variable or imposing a new bound.
- Either of these operations create a subproblem with a modified right-hand side, just as in the Benders' Algorithm.
- This subproblem has a value function that is a "shifted" version of the value function of the original problem.
- The branch-and-bound algorithm is implicitly constructing an approximation of the value function.
- The algorithm terminates when the approximation is strong at the right-hand side of interest.
- In a sense, branch and bound is a dynamic and recursive version of Benders' Algorithm.

Feasibility Problems

- Consider a tree search algorithm for solving a feasibility problem.
- Although intended to find the *optimal* solution, branch and bound can also be used in the search for *feasible* solutions.
- In principle, we only need to find a single path in the tree to a single feasible node.
- When there are many feasible nodes, this is a simple task; otherwise, it may not be.
  - The key is to avoid paths that cannot contain any feasible solution.
  - This amounts to proving *infeasibility* of certain subproblems.
  - This is the primary challenge.

How Do We Guide the Search?

- Multilevel problems are more difficult essentially because we are forced to explore more of the solution space.
- To exploit the lower complexity of feasibility problems, we must be able to efficiently stay on the "right path".
- In branch and cut, the search is guided primarily by the objective function.
- In related work, we have proposed branching rules that are guided by feasibility (such as maximum volume cutoff).
- Note again that this is primarily motivated by proving infeasibility, which is required to avoid going down blind alleys.
- However, the choice problems for these rules are difficult to solve.
- Our ability to improve the efficiency of feasibility search is limited by our ability to solve these choice problems.

Example: Bilevel Structure of the Branching Problem

- A typical criteria for selecting a branching disjunction is to maximize the bound increase resulting from imposing the disjunction.
- The problem of selecting the disjunction whose imposition results in the largest bound improvement has a natural *bilevel structure*.
  - The upper-level variables can be used to model the choice of disjunction (we'll see an example shortly).
  - The lower-level problem models the bound computation after the disjunction has been imposed.
- In strong branching, we are solving this problem essentially by enumeration.
- The bilevel branching paradigm is to select the branching disjunction directly by solving a bilevel program.

Example: Interdiction Branching

The following is a bilevel programming formulation for the problem of finding a smallest branching set in interdiction branching:

$$
\begin{aligned}
&\text{(BBP)} \quad \max \textstyle\sum c^\top x \\
&\text{s.t.} \\
&\qquad c^\top x \leq \bar{z} \\
&\qquad y \in \mathbb{B}^n \\
&\qquad x \in \arg\max_x c^\top x \\
&\qquad \text{s.t.} \\
&\qquad\qquad x_i + y_i \leq 1, \quad i \in N^a \\
&\qquad\qquad x \in \mathcal{F}^a
\end{aligned}
$$

where $\mathcal{F}$ is the feasible region of a given relaxation of the original problem used for computing the bound.

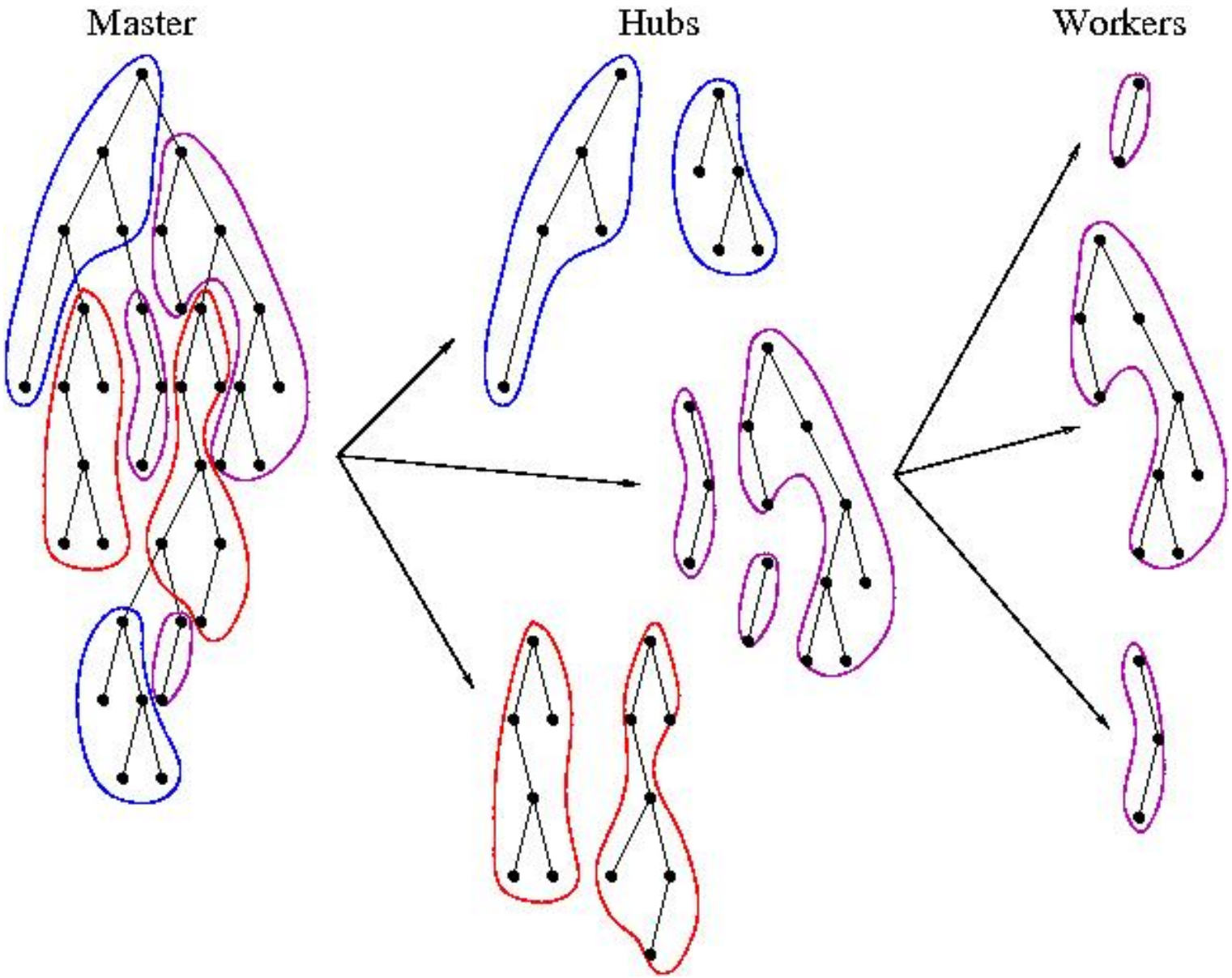
Outline

Why do Parallel Algorithms Arise Naturally?

Parallel algorithms are very natural in this setting for a number of reasons.

- The possible execution paths of a DTM can be thought of as specifying a tree (execution involves searching this tree).
- Problems in NP are those in which exploration of an exponential number of paths is unavoidable (in the worst case).
- Another way of thinking of problems in NP is as problems that can be solved in polynomial time given an exponential number of processors.
- Problems higher in the hierarchy require even more enumeration and thus present even more potential for parallelization.
- Alternatively, problems lower in the hierarchy are in some sense the most difficult to parallelize, since they present the greatest potential for wasted effort.

Task Partitioning in Search Algorithms



Why Isn't Parallel Computing a Panacea?

- Practical algorithms use heuristics to avoid enumeration as much as possible.
- We do not know ahead of time what execution paths will be necessary to the computation.
- This makes it very difficult to distribute the computation.
- In essence, practical algorithms are *designed not to be parallelizable*.

The Case of Branch and Bound

- The execution of branch and bound can be thought of as exploring a particular search tree.
- This tree is essentially the one arising from execution of the corresponding DTM.
- Solvers typically endeavor to make this tree as small as possible.
- The decision problem at each node is to determine which disjunction to branch on in order to minimize the resulting subtree.
- Thus, the solution process can be viewed as a kind of multilevel game in itself.
- As mentioned previously, minimizing the size of the tree actually reduces the potential for parallelization.

Where Do We Go From Here?

- Effective parallelization of feasibility search seems to be an extremely difficult problem to solve.
- The key may be in parallelizing solution of the choice problems themselves.
- These are optimization problems and thus are, in some sense, more parallelizable.
- This is blind conjecture at this point, however.

Conclusions

- This has been a presentation of some half-baked ideas about the complexity of search and the connections to multilevel optimization.
- There is much work to be done and many opportunities.
- Our aim is not just to develop the theory, but also to put it into practice.

Questions?

References I

Garey, M. and D. Johnson 1979. *Computers and Intractability: A Guide to the Thoery of NP-Completeness*. W.H. Freeman and Company.