Computational Optimization
ISE 407

Lecture 26

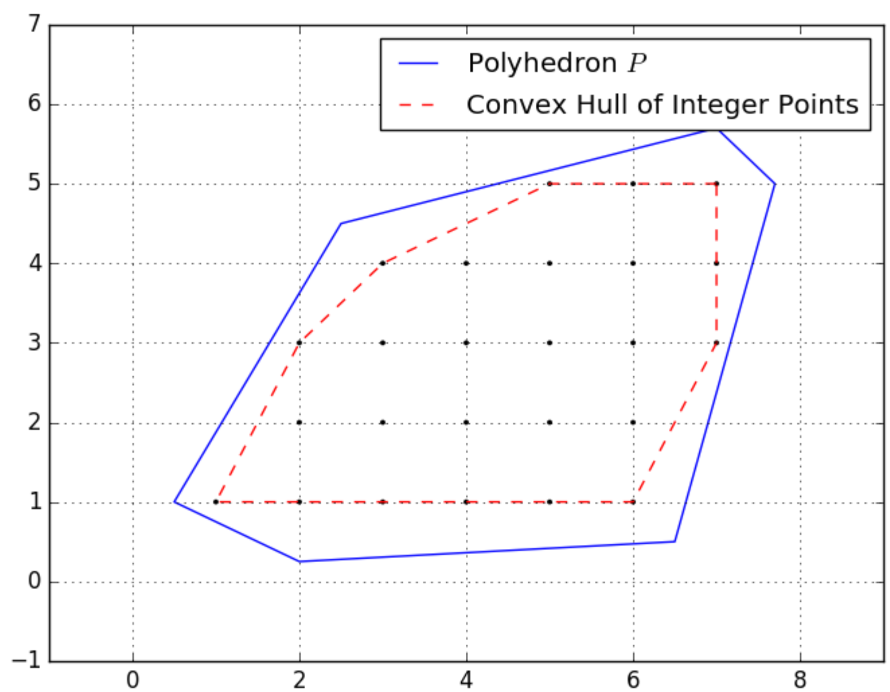
Dr. Ted Ralphs

Discrete Optimization

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

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

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



Special Case: Combinatorial Optimization

A *Combinatorial Optimization Problem* $CP = (E, \mathcal{F})$ consists of

- A *ground set* E,
- A set $\mathcal{F} \subseteq 2^E$ of *feasible solutions*, and
- A *cost function* $c \in \mathbb{Z}^E$ (optional).

The *cost* of $S \in \mathcal{F}$ is $c(S) = \sum_{e \in S} c_e$. The problem is to find a least cost member of $\mathcal{F}$.

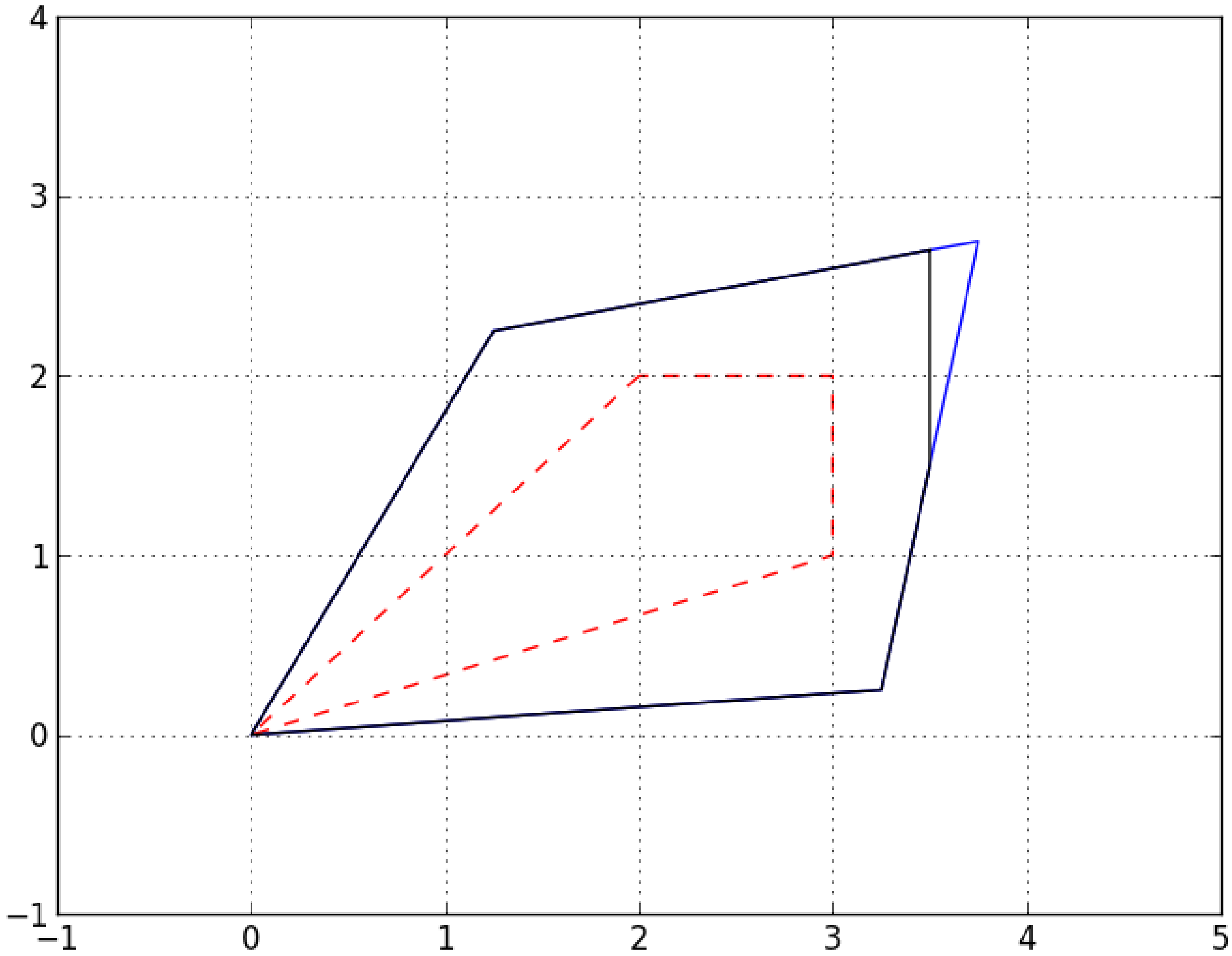
Solving Discrete Optimization Problems

- In general, *convex optimization problems* are "easy" to solve.
- In essence, this is because convex problems have only one local minimum—the global minimum.
- Discrete optimization problems are particularly challenging because
  - 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
  - constructing an explicit description of the convex hull of feasible solutions (convexify) $\Rightarrow$ Cutting plane methods.
  - using a set of logical disjunctions to represent the feasible region as a union of polyhedra (divide and conquer) $\Rightarrow$ Branch and bound

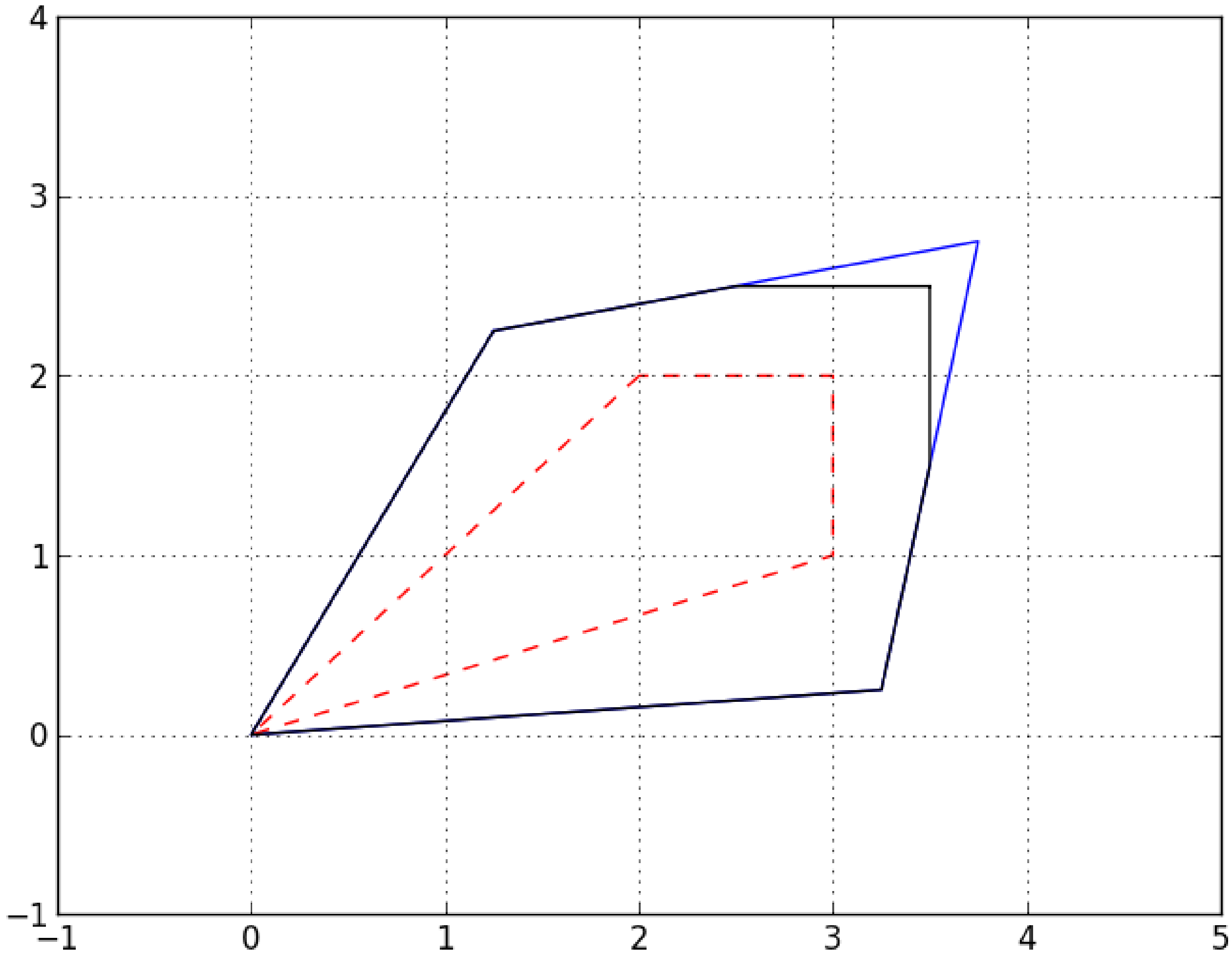
Computational Challenges

- In general, both of these approaches lead to descriptions of exponential size (bad).
- Fortunately, we typically only need a small part of the description to derive a proof of optimality.
- Modern state-of-the-art algorithms effectively combine these two techniques.
- One of the biggest challenges one faces in practice is dealing with the *numerics*.

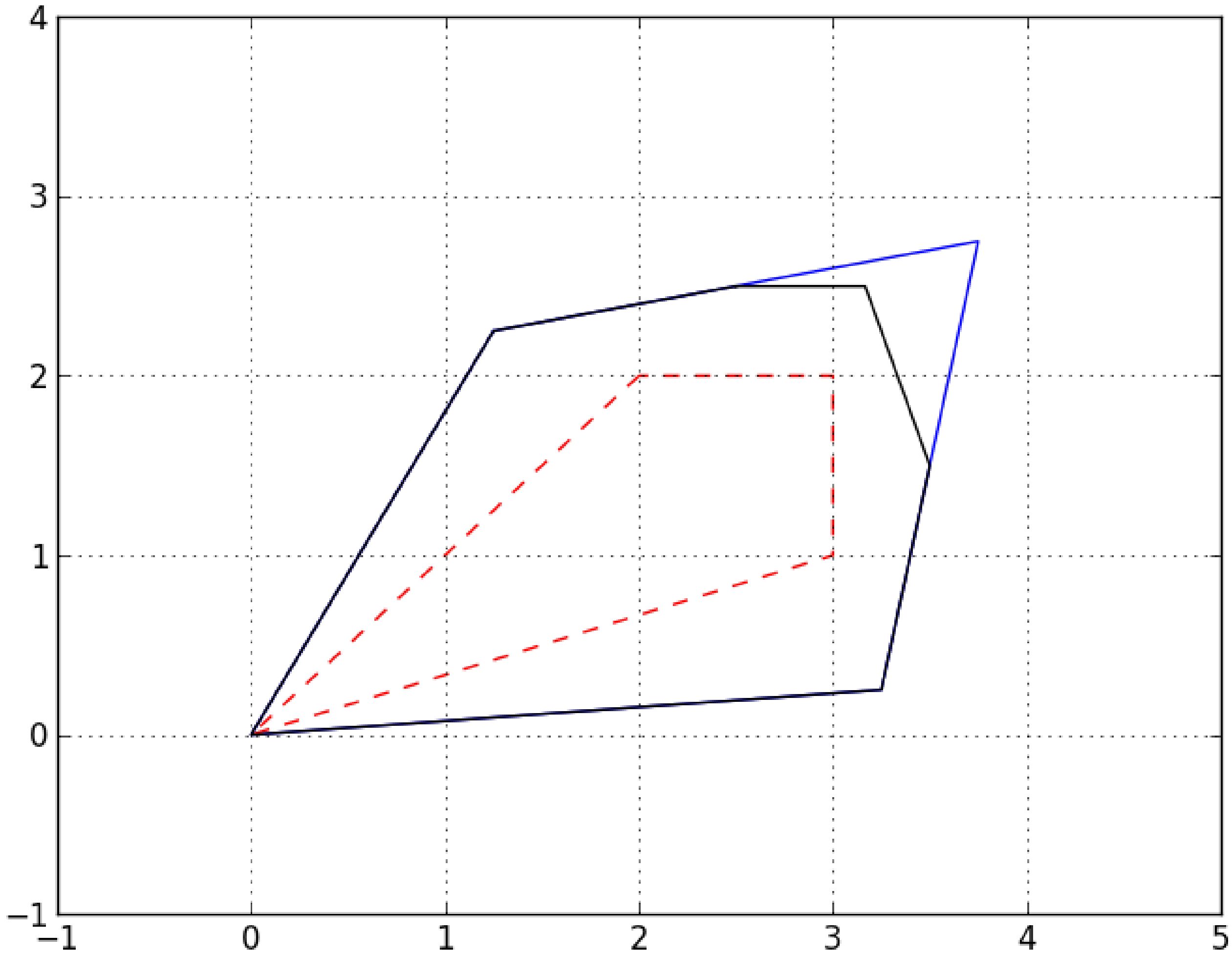
Example: Cutting Plane Algorithm



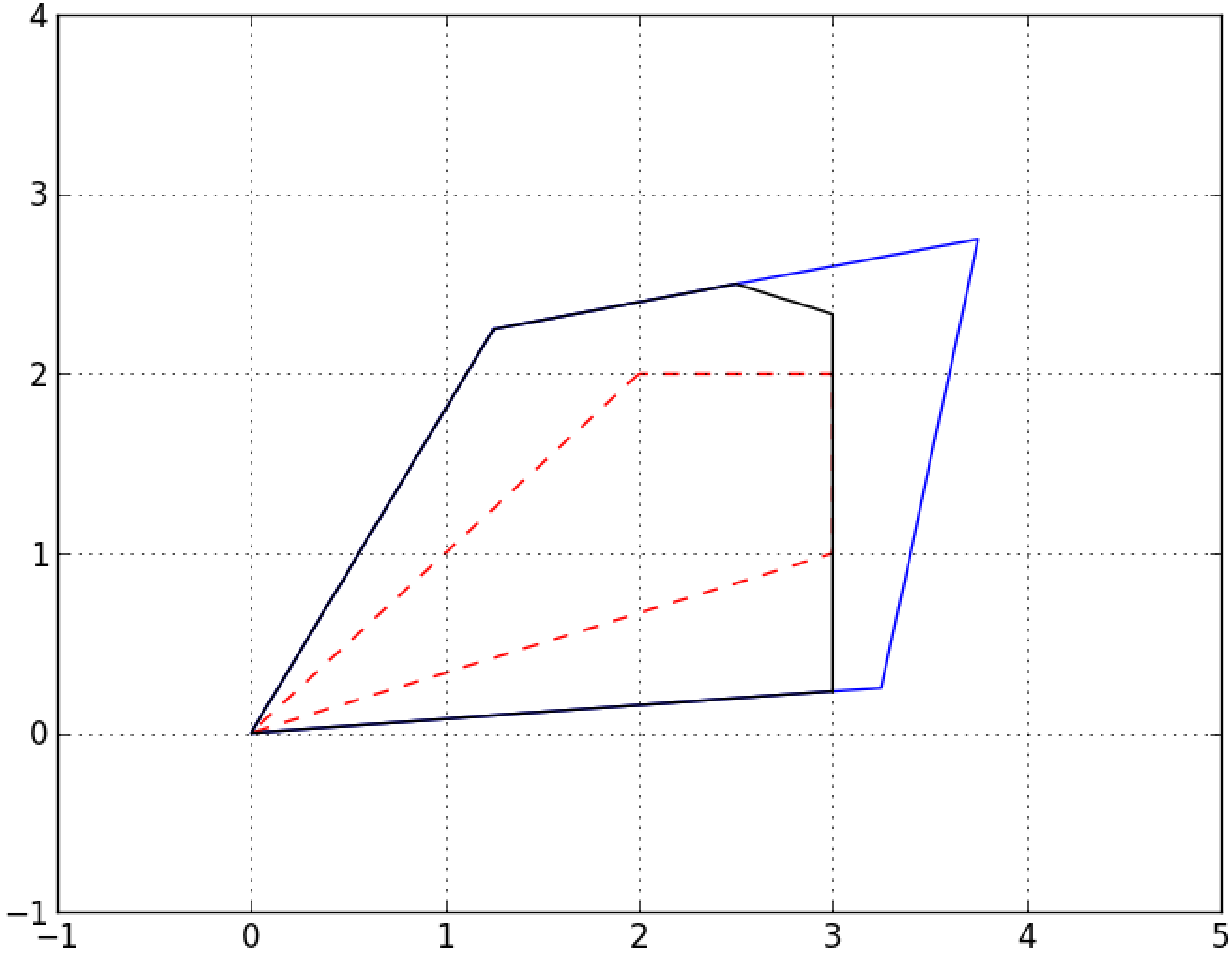
Example: Cutting Plane Algorithm



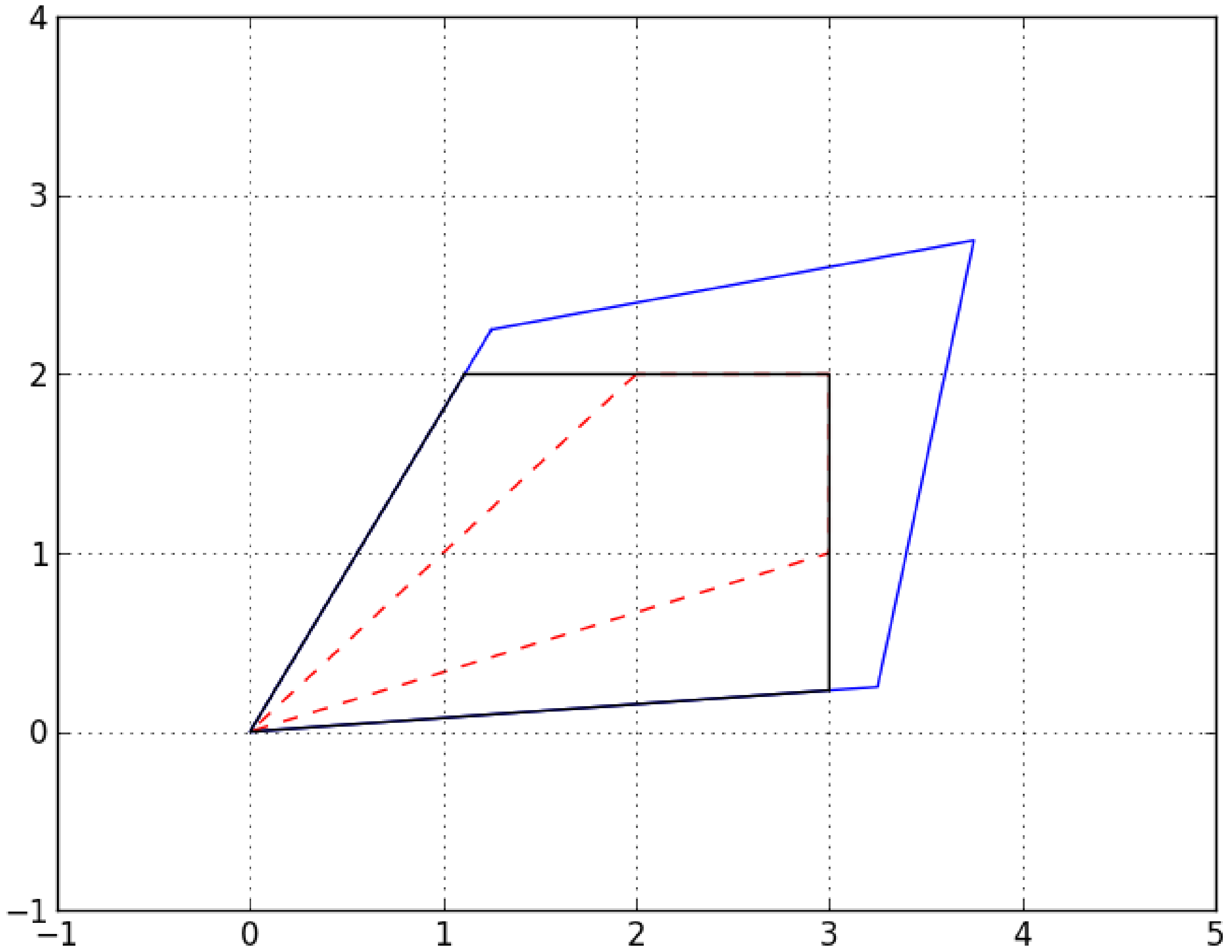
Example: Cutting Plane Algorithm



Example: Cutting Plane Algorithm



Example: Cutting Plane Algorithm



The Chvátal-Gomory Procedure

- Let $A = (a_1, a_2, \ldots, a_n)$ and $N = \{1, \ldots, n\}$.
  1. Choose a weight vector u.
  2. Obtain the valid inequality $\sum_{j \in N}(ua_j)x \leq ub$.
  3. Round the coefficients down to obtain $\sum_{j \in N}(\lfloor ua_j \rfloor)x \leq ub$. Why can we do this?
  4. Finally, round the right hand side down to obtain the valid inequality

$$\sum_{j \in N}(\lfloor ua_j \rfloor)x \leq \lfloor ub \rfloor$$

- This procedure is called the *Chvátal-Gomory* rounding procedure, or simply the *C-G procedure*.
- Surprisingly, any pur integer program can be solved by a finite number of iterations of this procedure!

Deriving Valid Inequalities from the Tableau

- Note that each row of the tableau is a nonnegative linear combination of the original equations.
- Suppose we choose a row in which the value of the basic variable is not an integer.
- Applying the procedure from the last slide, the resulting inequality will only involve nonbasic variables and will be of the form

$$\sum_{j \in NB} f_j x_j \geq f_0$$

  where $0 \leq f_j < 1$ and $0 < f_0 < 1$.
- We can conclude that the generated inequality will be violated by the current LP solution.
- Under mild assumptions on the algorithm used to solve the LP, this yields a general algorithm for solving integer programs.
- However, its convergence can be very slow and the *numerics are a challenge!*.

Divide and Conquer

- *Implicit enumeration* methods 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 a given MILP. We wish to solve $\min_{x \in F} c^\top x$.
- Divide and Conquer: We consider a partition of F into subsets $F_1, \ldots F_k$. 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 get bounds on the value of the original integer program.
- Types of Relaxations
  - Continuous relaxation
  - Combinatorial relaxation
  - Lagrangian relaxations

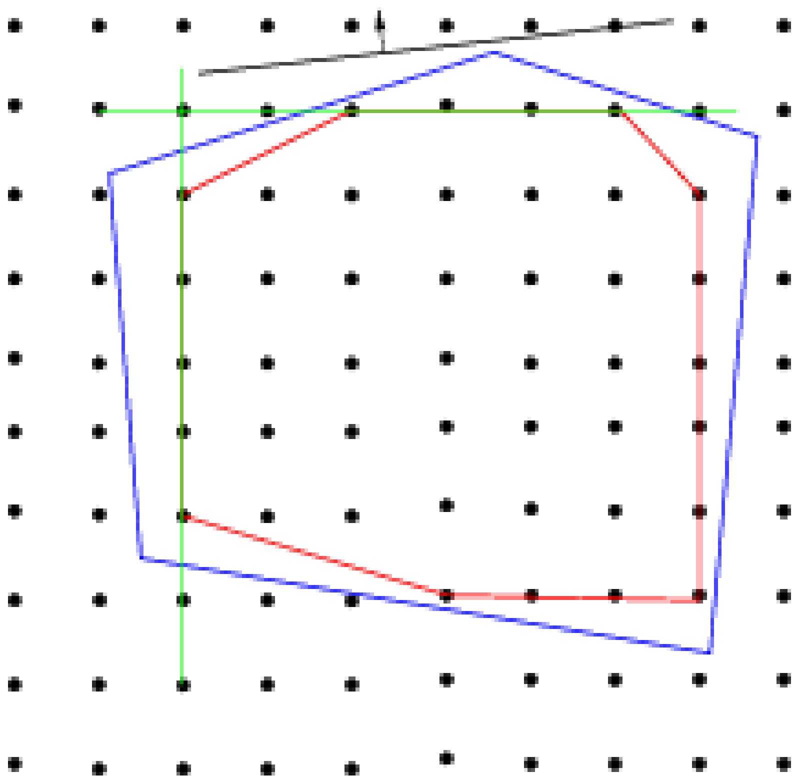
Branch and Bound Algorithm

Initialize the queue with 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.

Ingredient One: Bounding

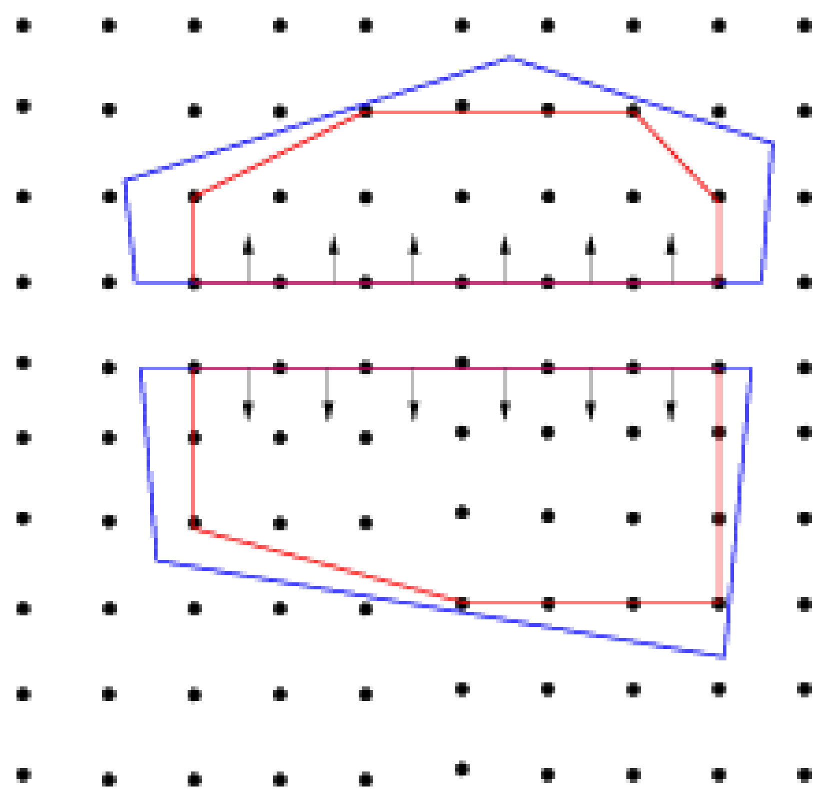
- The method by which bounds are derived in branch and bound is perhaps the most crucial element of an effective algorithm.
- The most common method of bounding is to develop an outer approximation of the convex hull of feasible solutions.
- More sophisticated methods based on decomposition are also possible.



Ingredient Two: 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.



Terminology

- If we picture the subproblems graphically, they form a *search tree*.
- Each subproblem is linked to its *parent* and eventually to its *children*.
- Eliminating a problem from further consideration is called *pruning*.
- The act of bounding and then branching is called *processing*.
- A subproblem that has not yet been considered is called a *candidate* for processing.
- The set of candidates for processing is called the *candidate list*.

Branch and Bound Tree

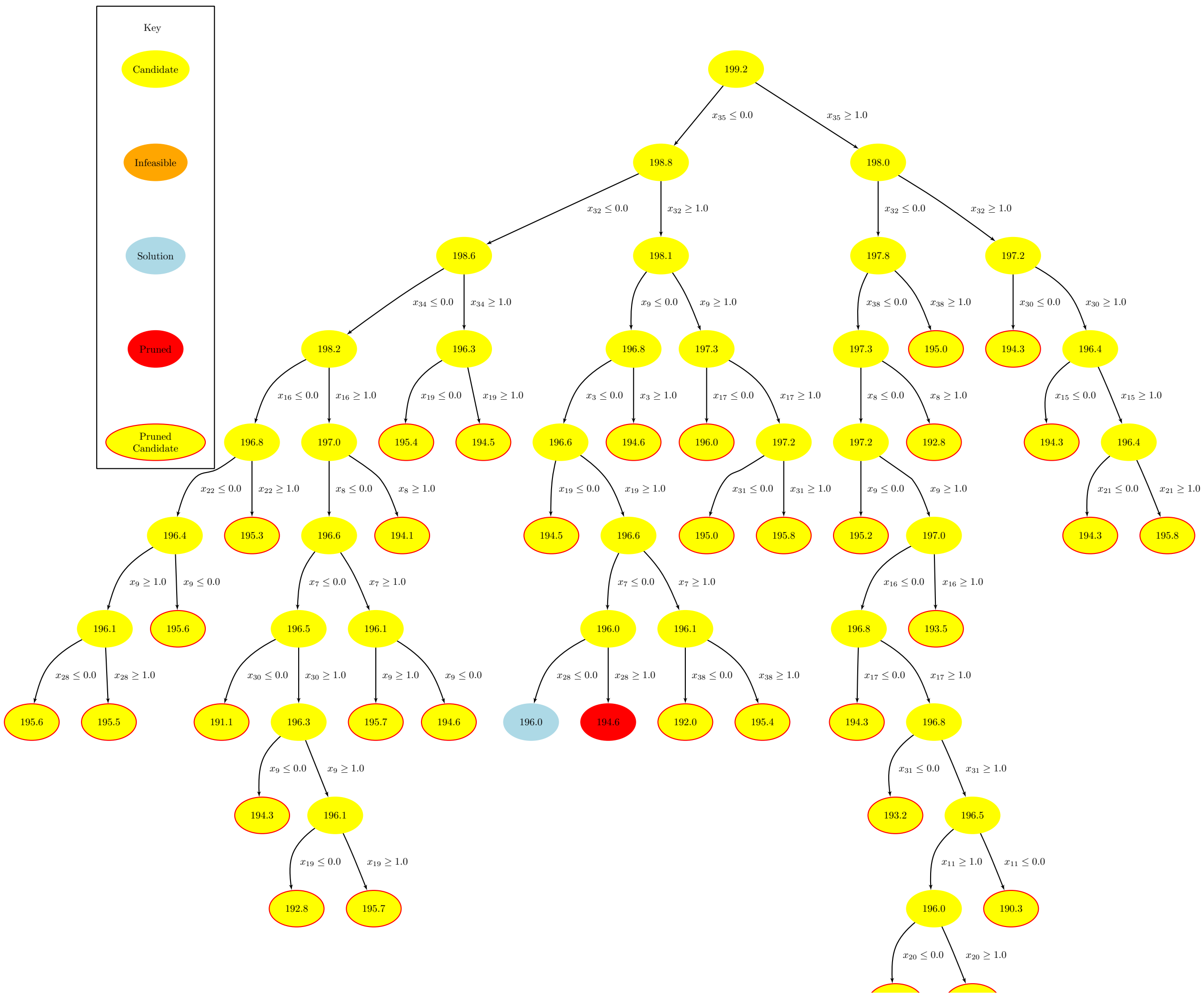


Figure 1: Final tree

A Thousand Words

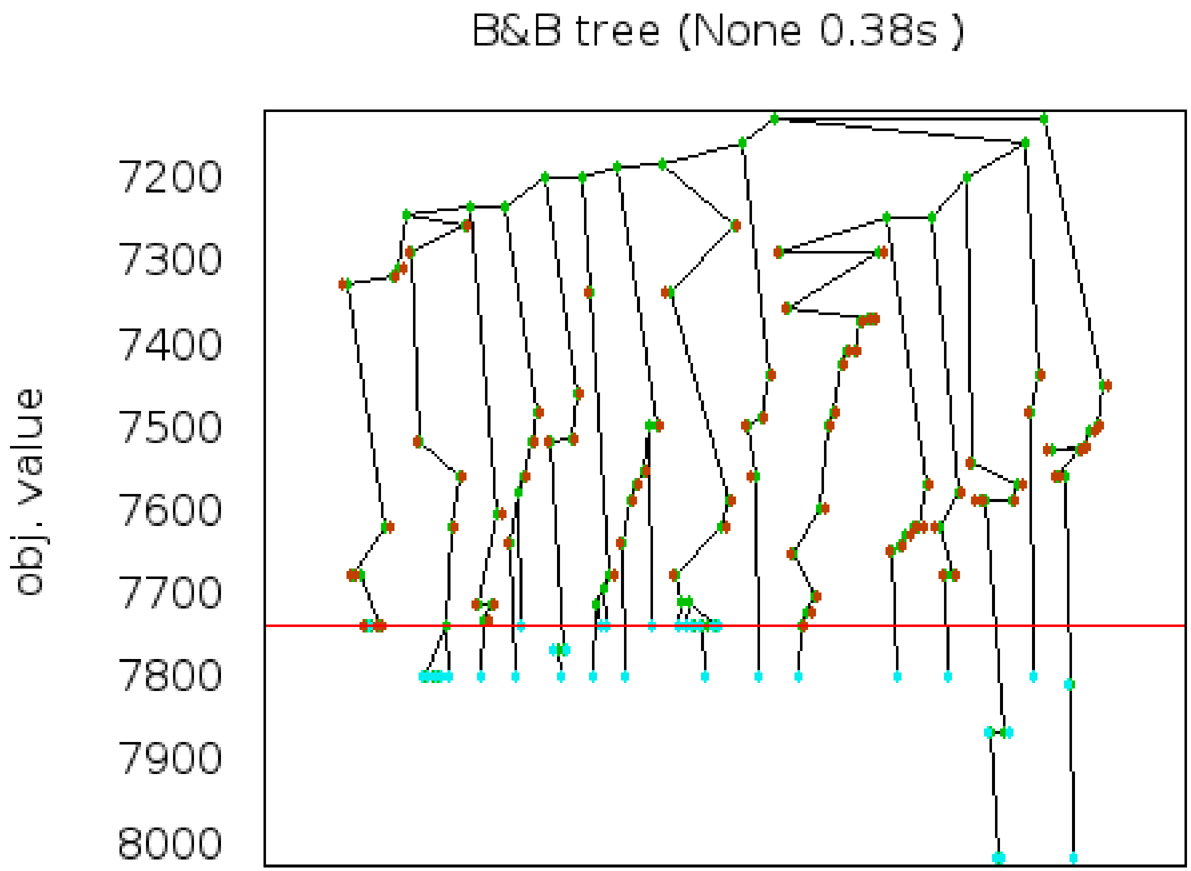


Figure 2: Tree after 400 nodes

A Thousand Words

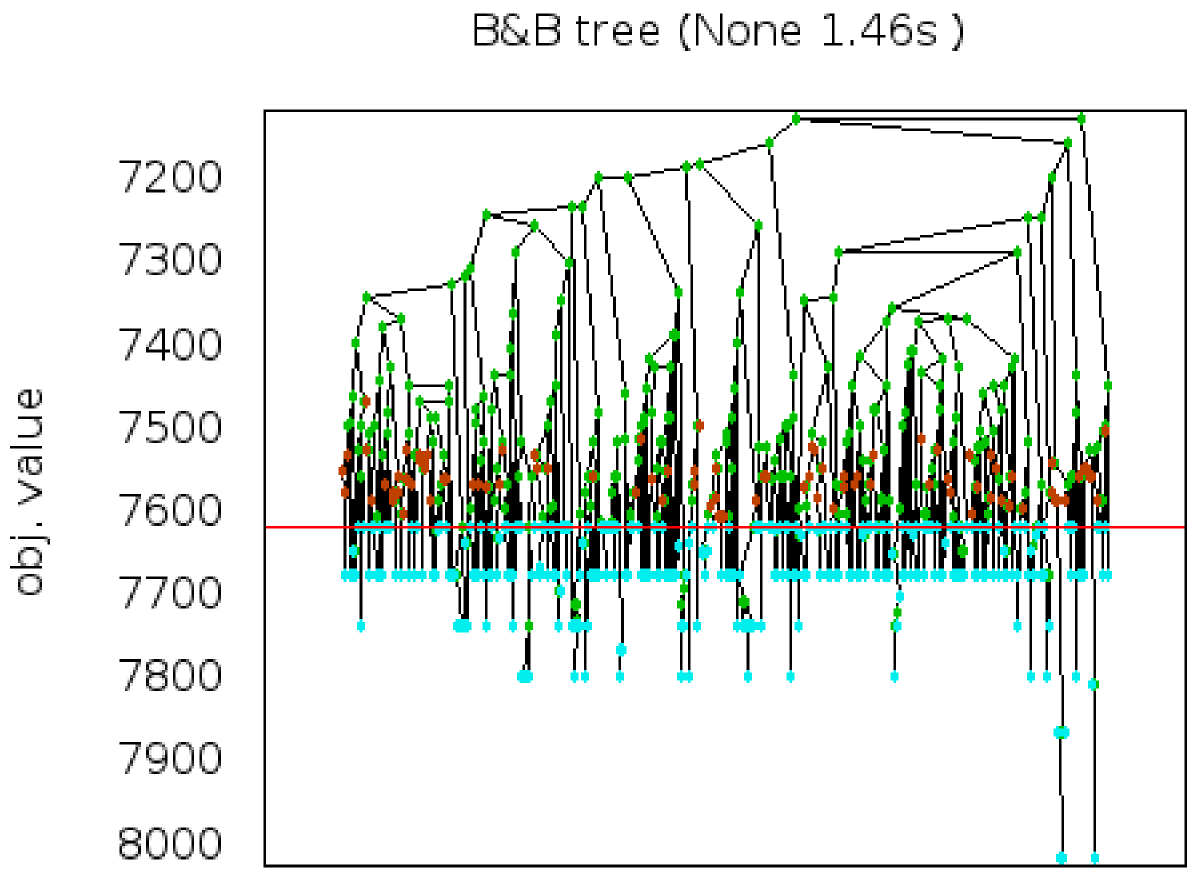


Figure 3: Tree after 1200 nodes

A Thousand Words

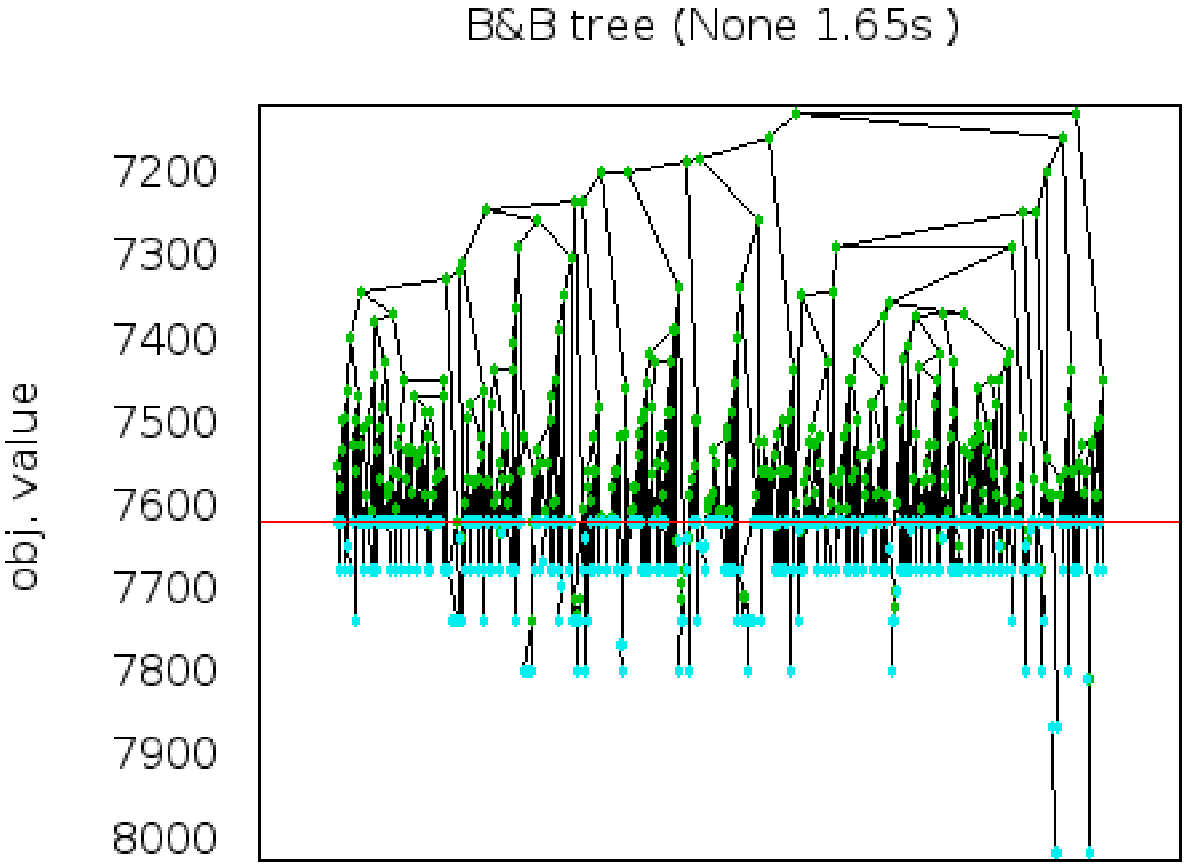


Figure 4: Final tree

Branch and Cut

- In practice, branching and cutting are usually integrated into a single algorithm.
- In principle, the same bound improvement can be obtained by either branching or cutting using the same disjunction, which creates a tradeoff.
  - Cutting does not create additional subproblems, but the conditioning of the matrix degrades when adding cuts.
  - Branching creates additional subproblems, but does not tend to degrade conditioning as much.
- The reasons that cutting generally degrades the conditioning can be understood geometrically.
- Because cuts are obtained as combinations of existing inequalities, new ones tend to be increasingly parallel to old ones.
- Eventually, this becomes such an issue that making further progres is impossible.