Chapter 2
Integer Programming

Paragraph 3
Advanced Methods

Search and Inference

- Different search strategies and branching constraint selections can tailor the search part of our quest to find and prove an optimal solution.
- Considering a relaxation, we have made a first attempt to reduce the search burden by inferring information about a problem.
- Pushing more optimization burden into polynomial inference procedures can dramatically speed up optimization.

Reduced Cost Filtering

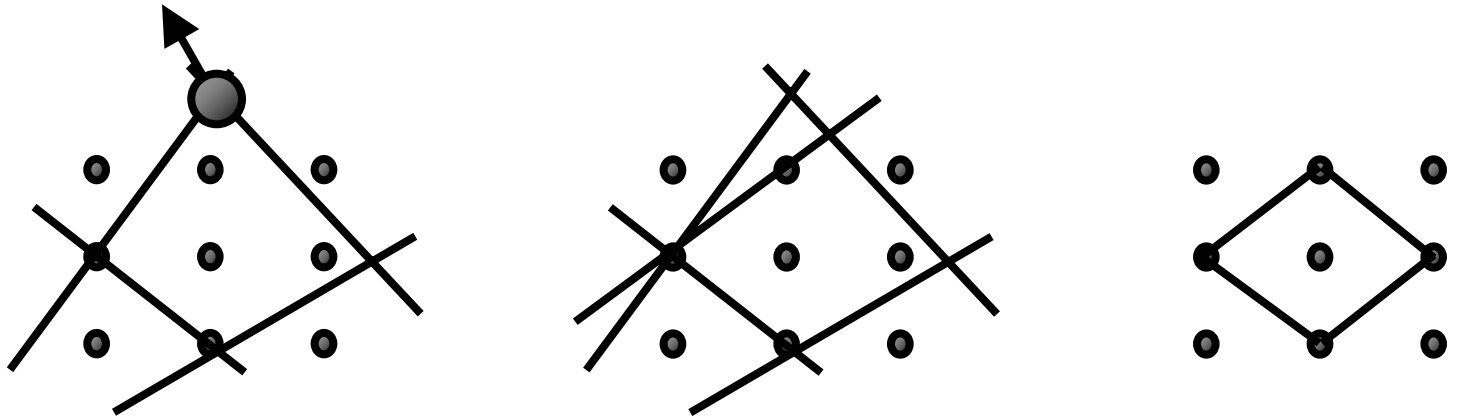
- Assume a certain non-basic variable x is 0 in the current relaxed LP solution, and the reduced costs of x are given by c_x.
- When enforcing a lower bound on x, $x \geq k$, then the dual simplex algorithm shows that the optimal relaxation value will increase (we assume minimization) by at least k c_x.
- If that increase is greater than the current gap between upper and lower bound, x must be lower or equal k-1 in any improving solution.

Cutting Planes

- Recall that Simplex returns the optimal solution to an IP when all corners are integer.
- Consequently, if we could find linear inequalities that give us the convex hull of the integer feasible region, we would be in good shape.
- The idea of cutting planes tries to infer so-called valid inequalities that preserve all integer feasible solutions, but cut off some purely fractional region of the LP polytope.

Cutting Planes

- Assume we find a fractional solution to our LP relaxation.
- A cutting plane can be derived that renders the current relaxed solution infeasible and that preserves all integer feasible solutions.



Gomory Cuts

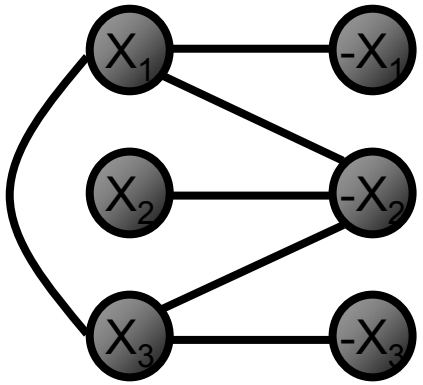
- Gomory cuts are one of the most famous examples of cutting planes.
- Given a constraint $x_{B(i)} + a_N^T x_N = b_i$ where b_i is fractional (the basic solution x^0 sets $x^0_{B(i)} = b_i$).
- Denote with $f_j = a_j - \lfloor a_i \rfloor$ the fractional part of a, and denote with $g_i = b_i - \lfloor b_i \rfloor$ the fractional part of b_i.
- Then, $x_{B(i)} + \lfloor a_N \rfloor^T x_N \leq b_i$. Since the left hand side is integer, we even have $x_{B(i)} + \lfloor a_N \rfloor^T x_N \leq \lfloor b_i \rfloor$. Subtracting this inequality from $x_{B(i)} + a_N^T x_N = b_i$ yields: $f_N^T x_N \geq g_i$.
- It can be shown that these cuts alone are sufficient to solve an IP without branching in finitely many steps!

Knapsack Cuts

- For binary IPs, some of the most effective cuts are based on considerations about the Knapsack problem.
- Assume we have that one constraint of our problem is $w^T x \leq C$ ($x \in \{0,1\}^n$).
- Assume also that we found some set $I \subseteq \{1,..,n\}$ such that $\Sigma_{i \in I} w_i > C$. Then, we can infer that it must hold: $\Sigma_{i \in I} x_i \leq |I| - 1$.
- Using sets I with smaller cardinalities gives stronger cuts. We can further improve a cut by considering $J = \{ j \mid j \notin I, w_j \geq \max_{i \in I} w_i\}$ and enforcing: $\Sigma_{i \in I \cup J} x_i \leq |I| - 1$.
- These cuts are also referred to as cover cuts.

Clique Cuts

- Again, for binary IPs we can consider what is called a conflict graph.



$X_1 + X_3 \leq 1$

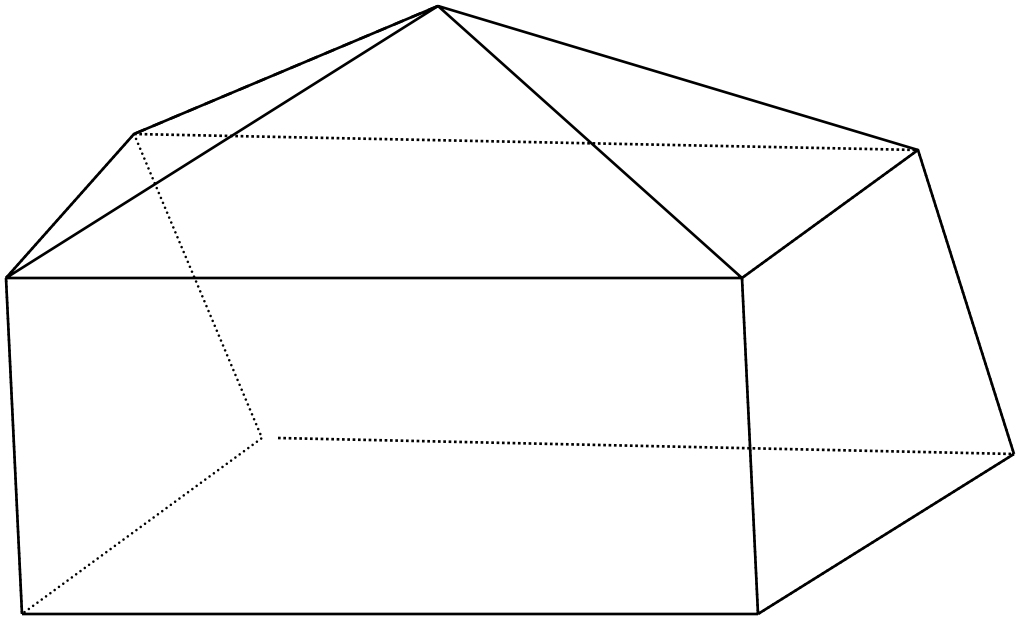
$X_1 - X_2 \leq 0$

$-X_2 + X_3 \leq 0$

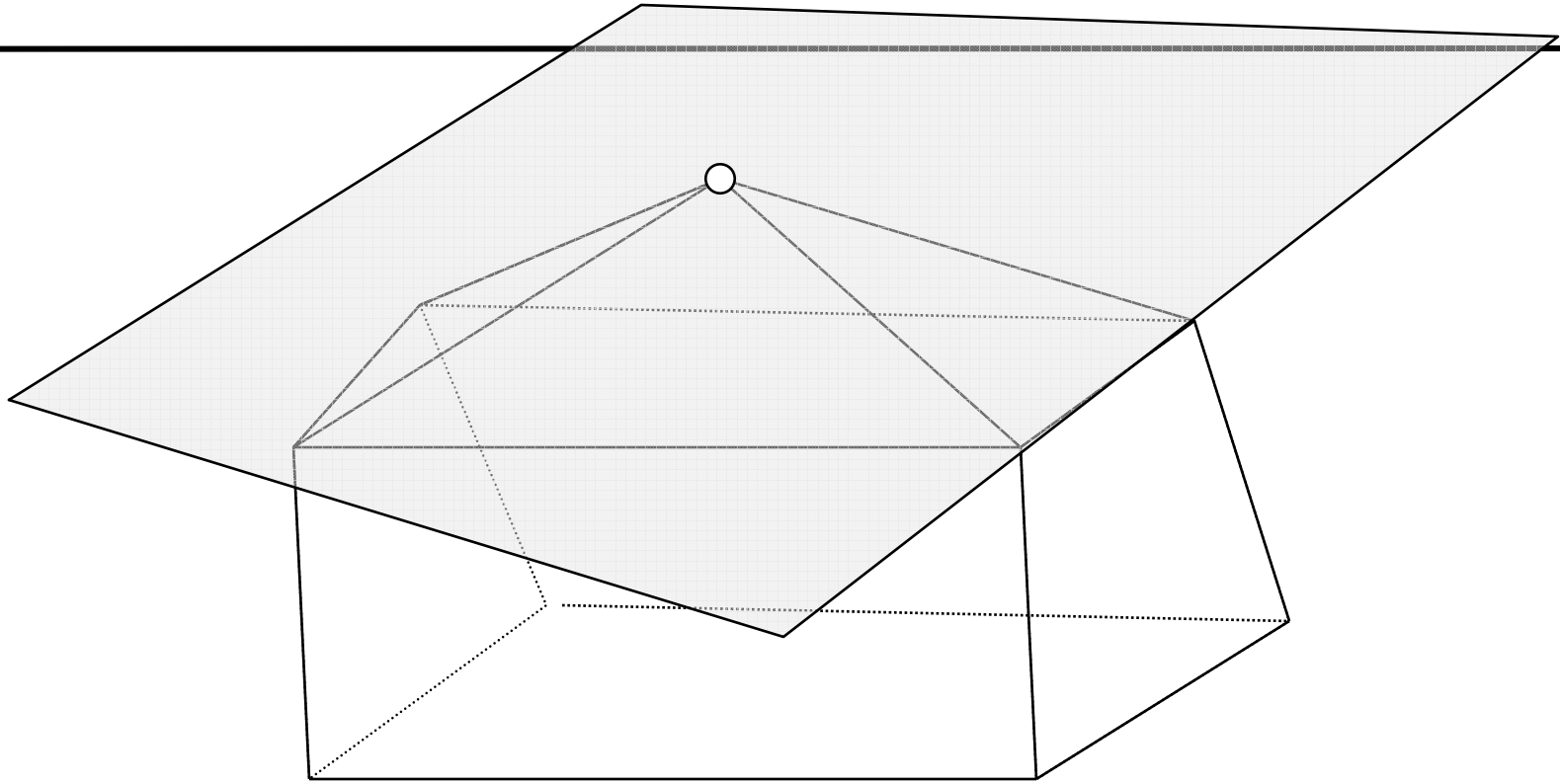
$X_1 - X_2 + X_3 \leq 0$

- Generally, cliques in the conflict graph give us so-called clique cuts that can be very powerful.

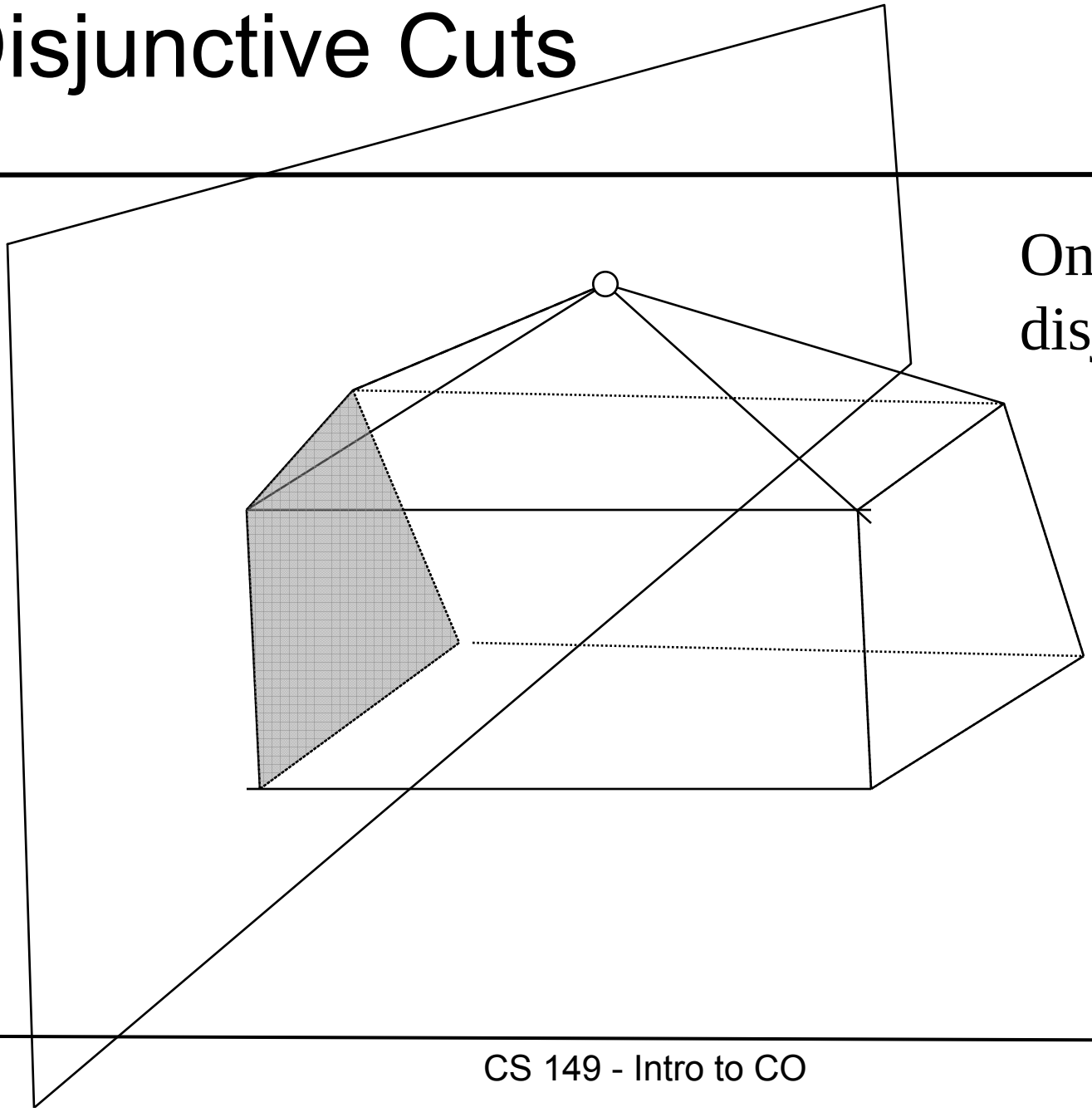
Disjunctive Cuts



Disjunctive Cuts



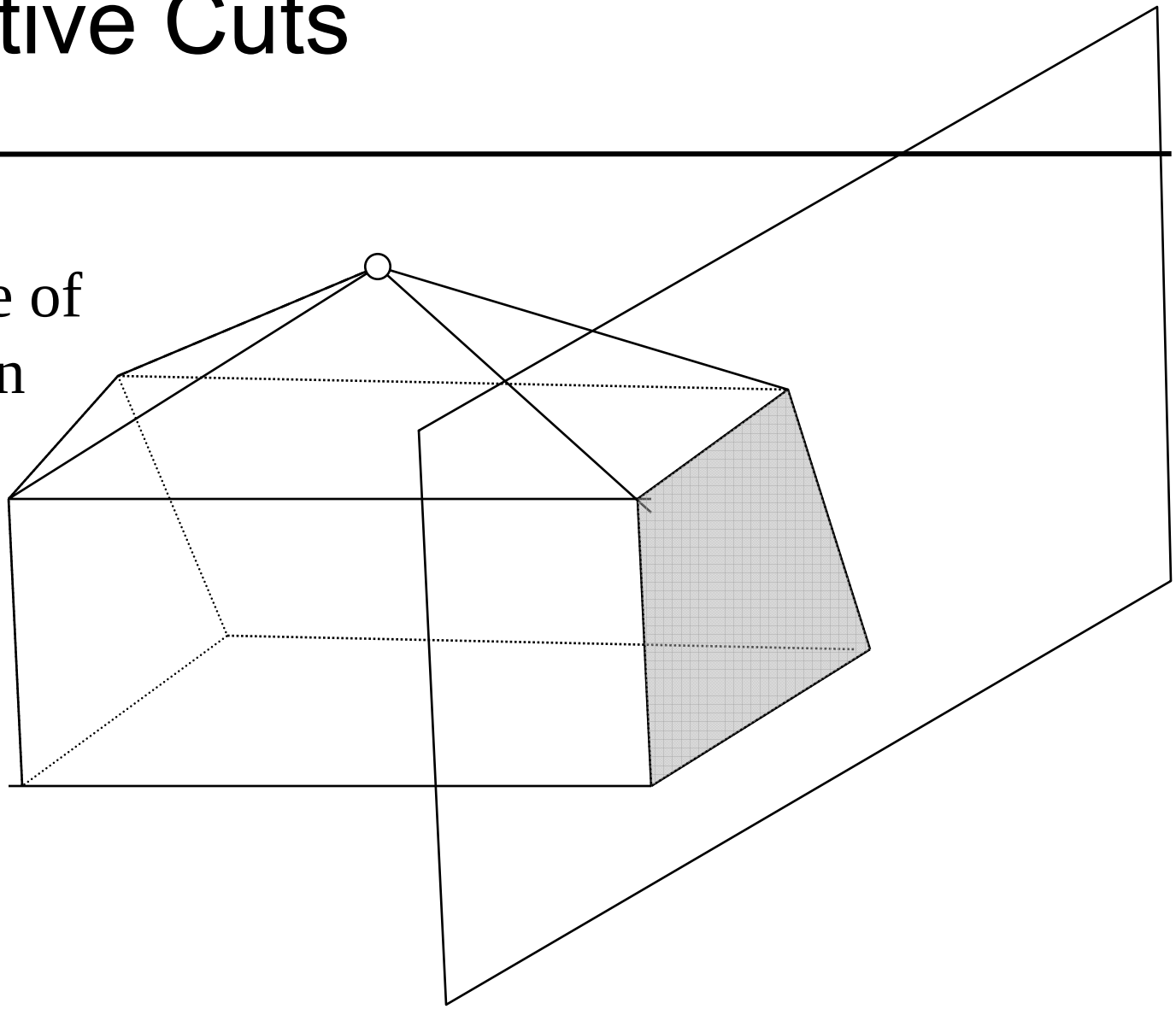
Disjunctive Cuts



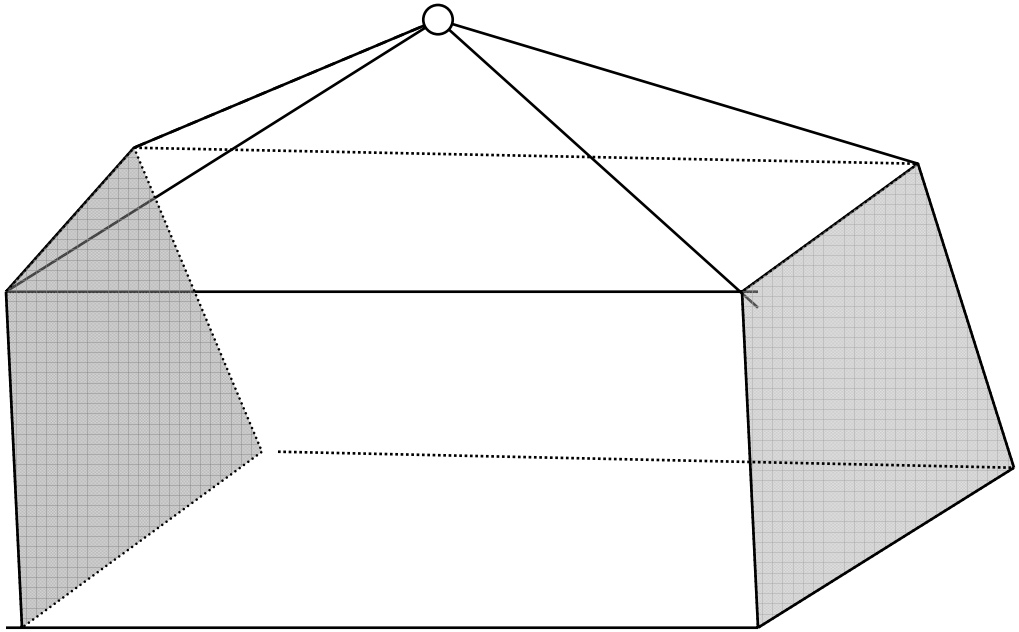
One side of the disjunction

Disjunctive Cuts

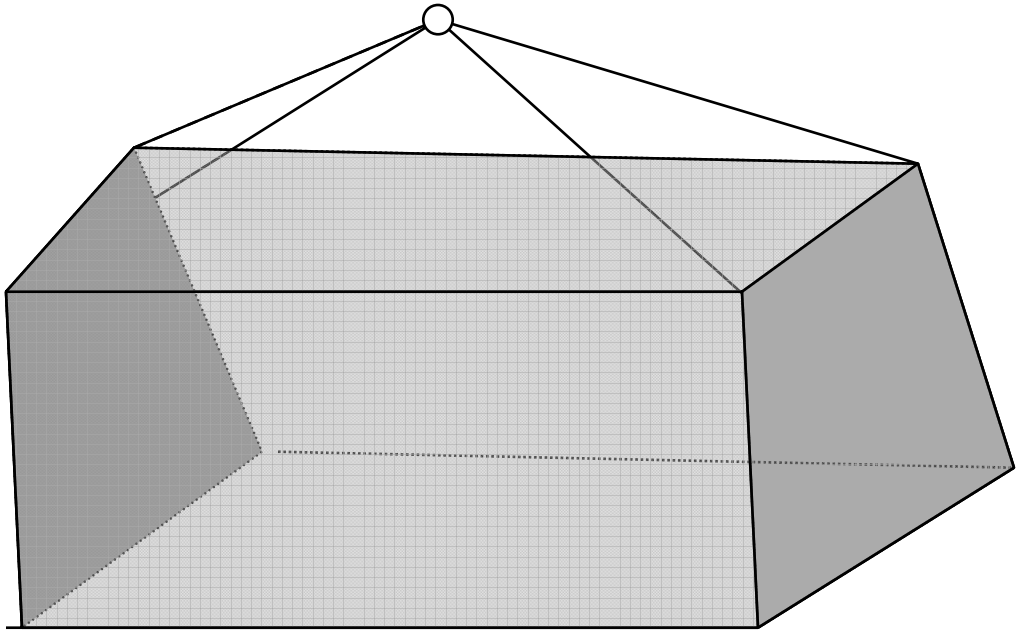
The other side of the disjunction



Disjunctive Cuts

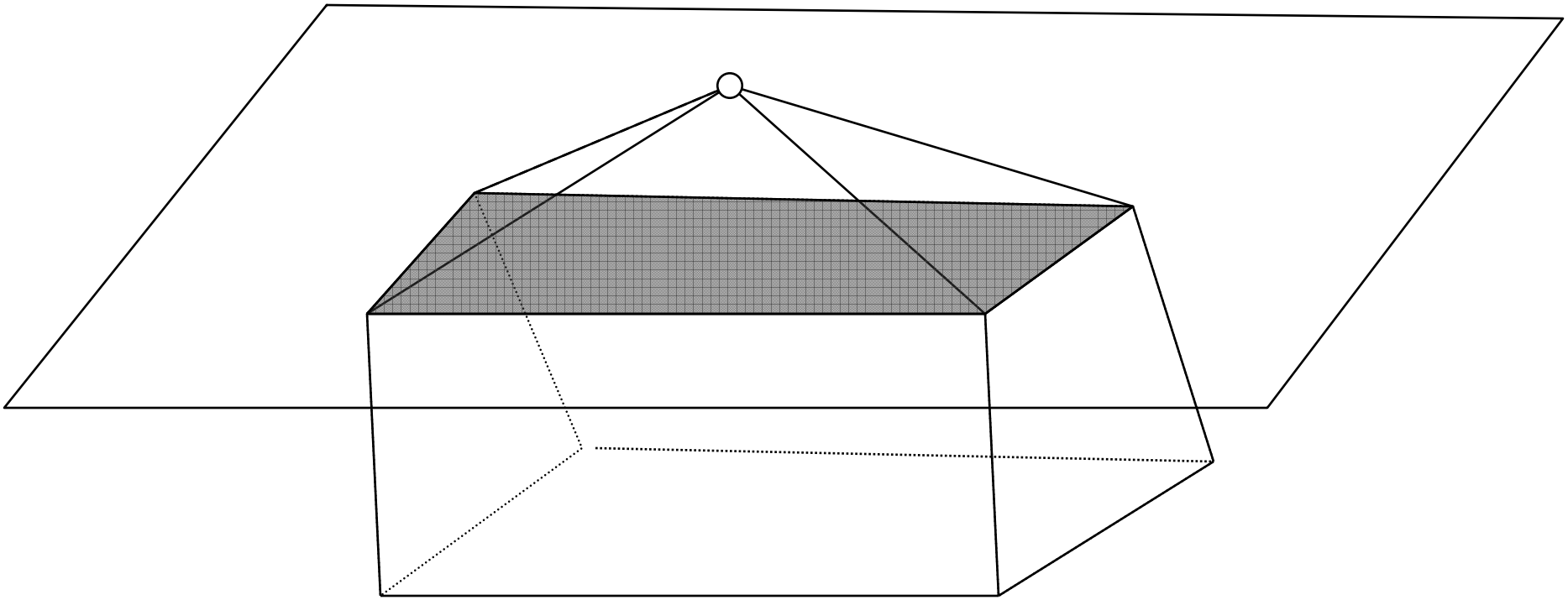


Disjunctive Cuts



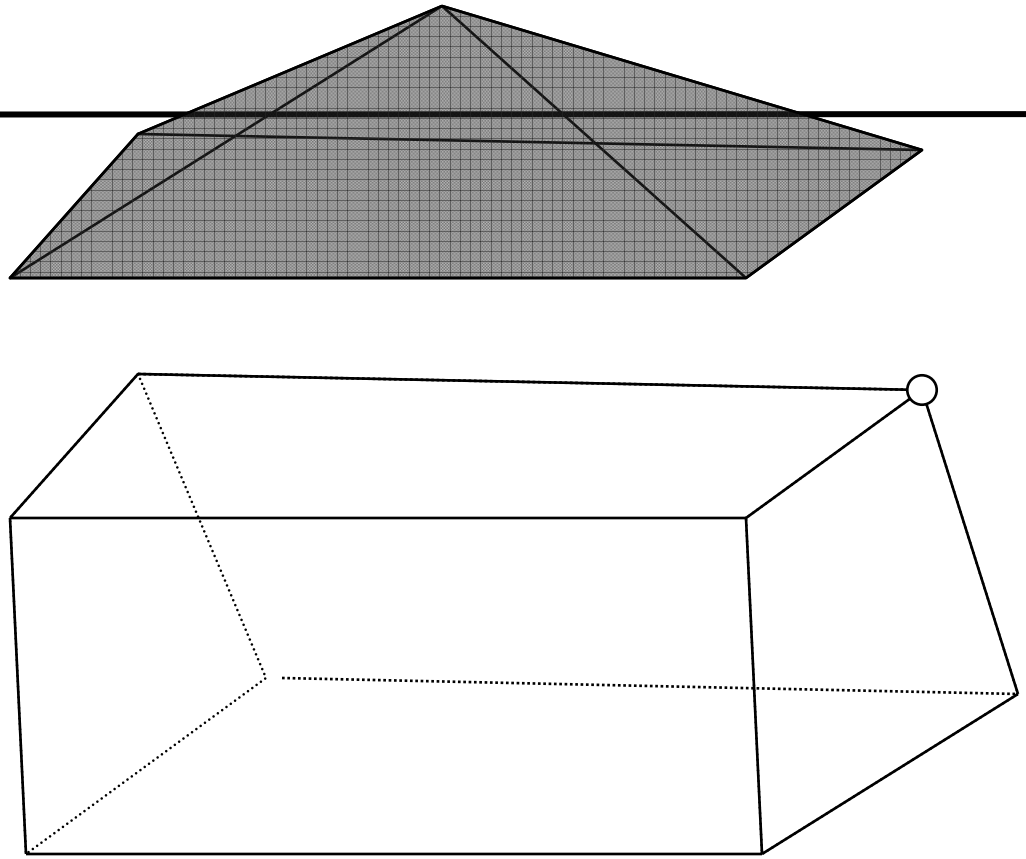
The convex-hull of the
union of the disjunctive sets

Disjunctive Cuts



One facet of the convex-hull
but it is also a cut!

Disjunctive Cuts



The new “feasible” solution!

Disjunctive Cuts

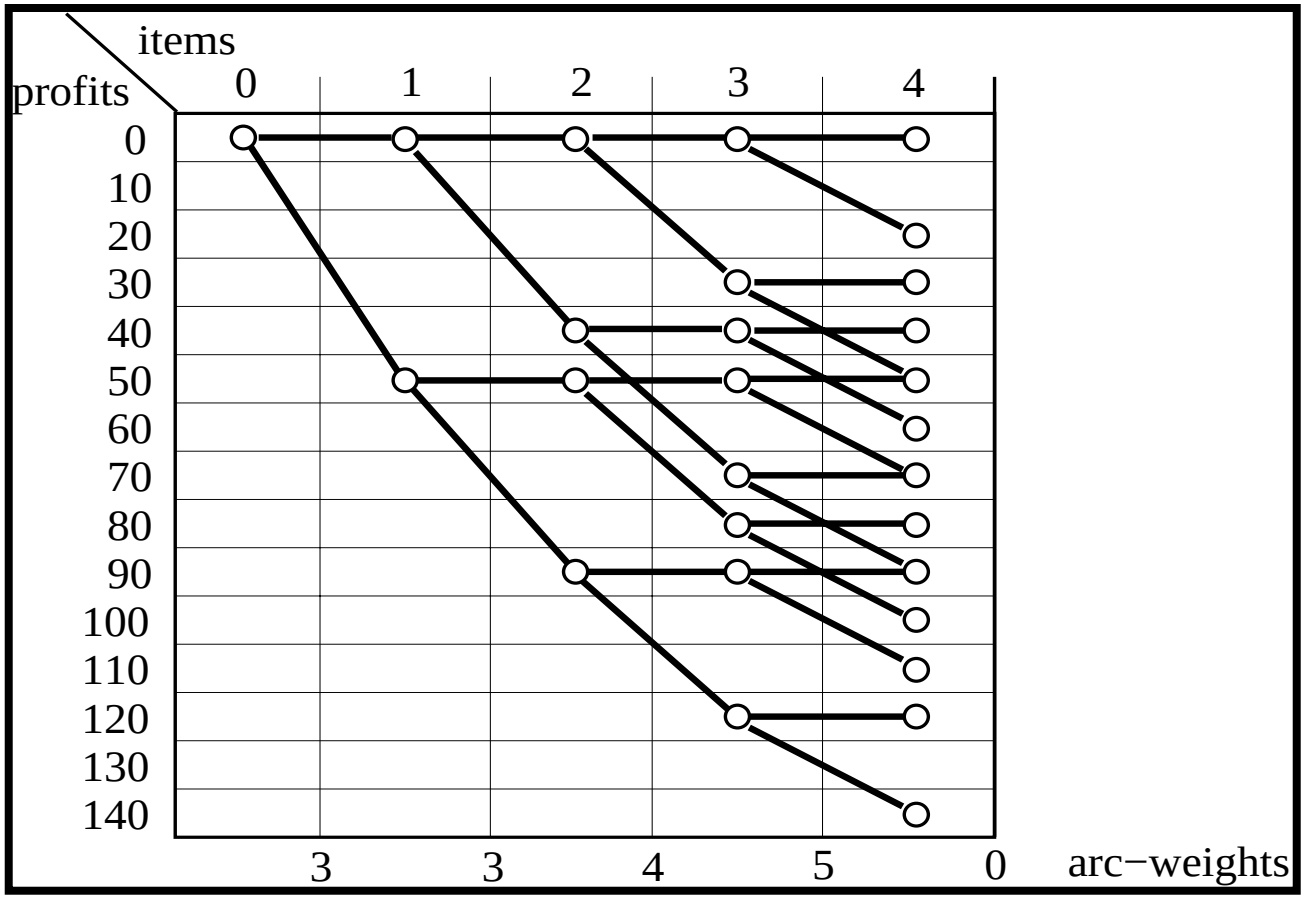
- In practice, we can generate disjunctive cuts by solving some linear program.
- Consequently, LPs cannot only be used to compute a bound on the objective, they can even be used to improve this bound by adding feasible cuts!

Dynamic Programming

- Assume we wanted to compute Fibonacci numbers: $F_{n+1} = F_n + F_{n-1}$, $F_0=1$, $F_1=1$.
- What is stupid about using a recursive algorithm?
- Now assume we want to solve the Knapsack problem, and the maximum item weight is 6. How should we solve this problem?
- Now assume the maximum item profit is 4. How should we solve the problem now?

Dynamic Programming

$$W(i,P) = \min\{W(i-1,P),\ W(i-1,P-p_i)+w_i\}$$



Approximation

- We can adapt a dynamic program to find a near-optimal solution in polynomial time.
- The core idea consists in scaling the profits.

$$\boxed{p_i = \left\lfloor \frac{p_i}{K} \right\rfloor}$$

- How should we choose K?
  - What is the runtime of the scaled program?
  - What is the error that we make?

Approximation

- A very simple 2-approximation can be derived from the linear programming solution.
- Wlog, we may assume that all items have weight lower or equal C.
- Out of the following two, take the solution that achieves maximum profit:
  - LP solution without the fractional item.
  - Take only the item with maximum profit.
- Can we use this 2-approximation to speed up our approximation scheme?

Thank you!