Pseudo-Boolean Optimization

IST, ULisboa

Outline

Outline

Vaccination Facility Location Problem

Suppose that you are in charge of deciding where to install new vaccination facilities from n potential locations in order to be able to vaccinate m people.

Let c_i denote the cost for opening a vaccination facility at location i and let d_{ij} denote the cost estimation of vaccinating person j from location i.

Provide a formulation that allows you to decide where to open the vaccination facilities such that the overall costs (installation and operation) are minimized.

Motivation

Facility Location Problem

- Problem variables
  - x_i : denotes if a vaccination facility is to be open at location i
  - y_{ij} : denotes if person j is vaccinated at location i

$$
\begin{array}{lll}
\textbf{Minimize} & \sum_{i=1}^{n} c_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{m} d_{ij} y_{ij} & \\
\textbf{Subject to} & \sum_{i=1}^{n} y_{ij} = 1 & \forall j \in \{1 \ldots m\} \\
 & x_i - y_{ij} \geq 0 & \forall i \in \{1 \ldots n\}, j \in \{1 \ldots m\} \\
 & x_i \in \{0,1\}, y_{ij} \in \{0,1\} &
\end{array}
$$

Outline

Pseudo-Boolean Optimization (PBO)

Formulation with n variables and m constraints

$$
\begin{array}{lllll}
\textbf{Minimize} & \sum_{j=1}^{n} c_j x_j & & & \\
\textbf{Subject to} & & & & \\
& \sum_{j=1}^{n} a_{ij} x_j & \{\geq, =, \leq\} & b_i & \forall i \in \{1, \ldots, m\} \\
& x_j \in \{0, 1\} & & & \forall j \in \{1, 2, \ldots, n\}
\end{array}
$$

- 0-1 Integer Linear Programming (0-1 ILP)

Pseudo-Boolean Optimization (PBO)

Translation to MaxSAT

$$\begin{array}{lllll} \textbf{Minimize} & \displaystyle\sum_{j=1}^{n} c_j x_j & & & \\ \textbf{Subject to} & & & & \\ & \displaystyle\sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \forall i \in \{1, \ldots, m\} \\ & x_j \in \{0, 1\} & & & \forall j \in \{1, 2, \ldots, n\} \end{array}$$

Pseudo-Boolean Optimization (PBO)

Translation to MaxSAT

$$
\begin{array}{llll}
\textbf{Minimize} & \sum_{j=1}^{n} c_j x_j & & \\
\textbf{Subject to} & & & \\
& \sum_{j=1}^{n} a_{ij} x_j \leq & b_i & \forall i \in \{1, \ldots, m\} \\
& x_j \in \{0, 1\} & & \forall j \in \{1, 2, \ldots, n\}
\end{array}
$$

- Encode each pseudo-Boolean constraint into CNF. All clauses used in the encoding are hard

Pseudo-Boolean Optimization (PBO)

Translation to MaxSAT

$$
\begin{array}{llll}
\textbf{Minimize} & \displaystyle\sum_{j=1}^{n} c_j x_j & & \\
\textbf{Subject to} & & & \\
& \displaystyle\sum_{j=1}^{n} a_{ij} x_j \quad \leq & b_i & \forall i \in \{1, \ldots, m\} \\
& x_j \in \{0, 1\} & & \forall j \in \{1, 2, \ldots, n\}
\end{array}
$$

- Encode each pseudo-Boolean constraint into CNF. All clauses used in the encoding are hard
- For each term $c_j x_j$ in the objective function, add a soft clause $(\neg x_j)$ with weight c_j

Pseudo-Boolean Optimization (PBO)

Algorithmic Solutions

- Translate into MaxSAT and use a Weighted MaxSAT algorithm

Pseudo-Boolean Optimization (PBO)

Algorithmic Solutions

- Translate into MaxSAT and use a Weighted MaxSAT algorithm
- Iterative Pseudo-Boolean solving
  - Linear search on the upper bound (or lower bound) using a pseudo-Boolean satisfiability solver
  - Binary search
  - ...

Pseudo-Boolean Optimization (PBO)

Algorithmic Solutions

- Translate into MaxSAT and use a Weighted MaxSAT algorithm
- Iterative Pseudo-Boolean solving
  - Linear search on the upper bound (or lower bound) using a pseudo-Boolean satisfiability solver
  - Binary search
  - ...
- Core-guided Pseudo-Boolean solving
  - Similar to MaxSAT core-guided using a pseudo-Boolean satisfiability solver enhanced with the feature of finding unsatisfiable subformulas
  - Terms in the objective function are treated as soft clauses

Pseudo-Boolean Optimization (PBO)

Algorithmic Solutions

- Translate into MaxSAT and use a Weighted MaxSAT algorithm
- Iterative Pseudo-Boolean solving
  - Linear search on the upper bound (or lower bound) using a pseudo-Boolean satisfiability solver
  - Binary search
  - ...
- Core-guided Pseudo-Boolean solving
  - Similar to MaxSAT core-guided using a pseudo-Boolean satisfiability solver enhanced with the feature of finding unsatisfiable subformulas
  - Terms in the objective function are treated as soft clauses
- Branch-and-Bound Search

Outline

Pseudo-Boolean Optimization (PBO)

Branch-and-Bound Search

- Search is organized using a tree
- Each node of the tree branches into two new nodes
  - Each branch corresponds to assigning value 0 or 1 to an unassigned variable
- An upper bound (UB) is maintained during the search
  - Initially, $UB = +\infty$
  - Updated whenever a new better solution is found
- At each node, a lower bound estimation procedure is applied. Search is bounded when the lower bound (LB) is higher or equal than the upper bound (UB)

Linear Programming Relaxation

Relaxation of a PBO/ILP problem

- Given a PBO problem instance, where variables have an integer domain, the Linear Programming Relaxation is the corresponding linear program where the variable's integer constraints are relaxed

Linear Programming Relaxation

Relaxation of a PBO/ILP problem

- Given a PBO problem instance, where variables have an integer domain, the Linear Programming Relaxation is the corresponding linear program where the variable's integer constraints are relaxed

PBO:

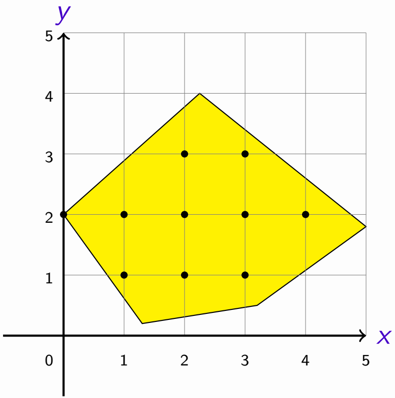
$$
\begin{array}{lll}
\textbf{Minimize} & \sum_{j=1}^{n} c_j x_j & \\
\textbf{Subject to} & \sum_{j=1}^{n} a_{ij} x_j & \leq \quad b_i \\
 & x_j \in \{0,1\} &
\end{array}
$$

LPR:

$$
\begin{array}{lll}
\textbf{Minimize} & \sum_{j=1}^{n} c_j x_j & \\
 & \sum_{j=1}^{n} a_{ij} x_j & \leq \quad b_i \\
 & x_j \in [0,1] &
\end{array}
$$

Linear Programming Relaxation (LPR)

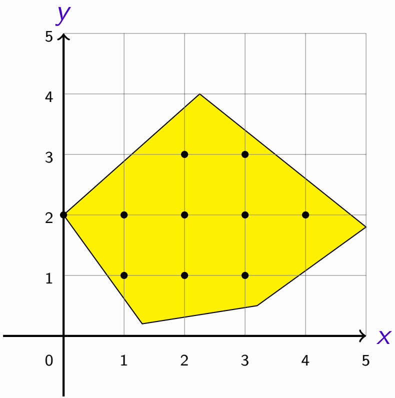
Example in Integer Linear Programming (ILP)



- ILP: 10 feasible points
- LPR: yellow region is feasible region

Linear Programming Relaxation (LPR)

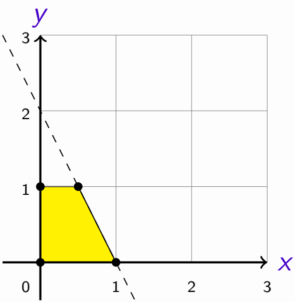
Example in Integer Linear Programming (ILP)



- If the optimal solution of the linear programming relaxation (LPR) is $x = 0, y = 2$, then it is also the ILP optimal solution

Linear Programming Relaxation (LPR)

Example in Pseudo-Boolean Optimization

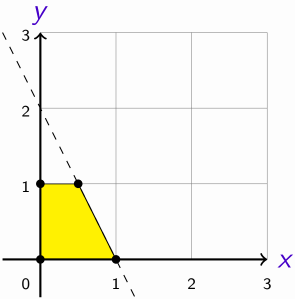


$$\begin{array}{ll} \textbf{Minimize} & -3y - x \\ \textbf{Subject to} & \\ & y + 2x \quad \leq \quad 2 \\ & x, y \in \{0, 1\} \end{array}$$

- Optimal solution of the LPR is $x = 0.5, y = 1$.

Linear Programming Relaxation (LPR)

Example in Pseudo-Boolean Optimization



Minimize $-3y - x$

Subject to

$$\begin{array}{lcl} y + 2x & \leq & 2 \\ x, y \in \{0, 1\} & & \end{array}$$

- Optimal solution of the LPR is $x = 0.5, y = 1$.
- If the objective function was $-3y + x$, the optimal solution of the LPR would be $x = 0, y = 1$. The LPR solution would be integer
  - When this happens, it is also the solution of the PBO instance

Linear Programming Relaxation (LPR)

Relevance of Linear Programming Relaxation

- The linear program relaxation can be solved quickly (i.e., in polynomial time)

Linear Programming Relaxation (LPR)

Relevance of Linear Programming Relaxation

- The linear program relaxation can be solved quickly (i.e., in polynomial time)
- If the relaxed linear program returns an optimal solution where all variables have integer value, then the solution of the relaxed linear program is also the optimal solution of the PBO problem

Linear Programming Relaxation (LPR)

Relevance of Linear Programming Relaxation

- The linear program relaxation can be solved quickly (i.e., in polynomial time)
- If the relaxed linear program returns an optimal solution where all variables have integer value, then the solution of the relaxed linear program is also the optimal solution of the PBO problem
- Otherwise, if the solution of the relaxed linear program is not integer for some variable, it still provides a lower bound on the optimal value of the PBO optimal solution

Linear Programming Relaxation (LPR)

Relevance of Linear Programming Relaxation

- The linear program relaxation can be solved quickly (i.e., in polynomial time)
- If the relaxed linear program returns an optimal solution where all variables have integer value, then the solution of the relaxed linear program is also the optimal solution of the PBO problem
- Otherwise, if the solution of the relaxed linear program is not integer for some variable, it still provides a lower bound on the optimal value of the PBO optimal solution
- There are other uses of linear programming relaxation, namely in directing the search process

Branch and Bound Algorithm

General Description

- Search by recursively dividing into smaller subproblems
- Continuously use linear programming on relaxation to:
  1. get lower bounds (in case of minimization)
  2. get candidate solutions

Branch and Bound Algorithm

Description of the Algorithm

1. Init: Initialize $UB = +\infty$
2. Init: Start with the original problem as the only node and mark it as active

Branch and Bound Algorithm

Description of the Algorithm

1. Init: Initialize $UB = +\infty$
2. Init: Start with the original problem as the only node and mark it as active
3. LPR Solve: Select an active node k. Let z denote the value of the objective function for the optimal solution of the Linear Programming Relaxation (LPR) at node k

Branch and Bound Algorithm

Description of the Algorithm

1. Init: Initialize $UB = +\infty$
2. Init: Start with the original problem as the only node and mark it as active
3. LPR Solve: Select an active node k. Let z denote the value of the objective function for the optimal solution of the Linear Programming Relaxation (LPR) at node k
4. Improve Upper Bound: If the solution of the LPR is integer and $z < UB$, then let $UB = z$ and save solution

Branch and Bound Algorithm

Description of the Algorithm

1. Init: Initialize $UB = +\infty$
2. Init: Start with the original problem as the only node and mark it as active
3. LPR Solve: Select an active node k. Let z denote the value of the objective function for the optimal solution of the Linear Programming Relaxation (LPR) at node k
4. Improve Upper Bound: If the solution of the LPR is integer and $z < UB$, then let $UB = z$ and save solution
5. Split: If the LPR is feasible but optimal solution is not integer and $z < UB$, then use branching procedure to generate two new nodes and mark them as active

Branch and Bound Algorithm

Description of the Algorithm

1. Init: Initialize $UB = +\infty$
2. Init: Start with the original problem as the only node and mark it as active
3. LPR Solve: Select an active node k. Let z denote the value of the objective function for the optimal solution of the Linear Programming Relaxation (LPR) at node k
4. Improve Upper Bound: If the solution of the LPR is integer and $z < UB$, then let $UB = z$ and save solution
5. Split: If the LPR is feasible but optimal solution is not integer and $z < UB$, then use branching procedure to generate two new nodes and mark them as active
6. Deque: Mark node k as inactive

Branch and Bound Algorithm

Description of the Algorithm

1. Init: Initialize $UB = +\infty$
2. Init: Start with the original problem as the only node and mark it as active
3. LPR Solve: Select an active node k. Let z denote the value of the objective function for the optimal solution of the Linear Programming Relaxation (LPR) at node k
4. Improve Upper Bound: If the solution of the LPR is integer and $z < UB$, then let $UB = z$ and save solution
5. Split: If the LPR is feasible but optimal solution is not integer and $z < UB$, then use branching procedure to generate two new nodes and mark them as active
6. Deque: Mark node k as inactive
7. Repeat: If there are active nodes, go back to **3**. Otherwise, the algorithm ends and the optimal solution is the last one saved

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees
  - The usual method is to use information from the LPR solution

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees
  - The usual method is to use information from the LPR solution
  - Let x_i^* denote the value of x_i in the LPR solution

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees
  - The usual method is to use information from the LPR solution
  - Let x_i^* denote the value of x_i in the LPR solution
  - Branch on variable x_i that minimizes $|x_i^* - 0.5|$

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees
  - The usual method is to use information from the LPR solution
  - Let x_i^* denote the value of x_i in the LPR solution
  - Branch on variable x_i that minimizes $|x_i^* - 0.5|$
  - One branch where $x_i = 0$ and the other where $x_i = 1$

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees
  - The usual method is to use information from the LPR solution
  - Let x_i^* denote the value of x_i in the LPR solution
  - Branch on variable x_i that minimizes $|x_i^* - 0.5|$
  - One branch where $x_i = 0$ and the other where $x_i = 1$
  - Note that the solution of the LPR is excluded from both nodes that are generated

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees
  - The usual method is to use information from the LPR solution
  - Let x_i^* denote the value of x_i in the LPR solution
  - Branch on variable x_i that minimizes $|x_i^* - 0.5|$
  - One branch where $x_i = 0$ and the other where $x_i = 1$
  - Note that the solution of the LPR is excluded from both nodes that are generated

Branch and Bound Algorithm

Algorithm Options

- How to branch at a given node k?
  - Heuristic decision. Different branching procedures lead to different search trees
  - The usual method is to use information from the LPR solution
  - Let x_i^* denote the value of x_i in the LPR solution
  - Branch on variable x_i that minimizes $|x_i^* - 0.5|$
  - One branch where $x_i = 0$ and the other where $x_i = 1$
  - Note that the solution of the LPR is excluded from both nodes that are generated

- **Remark:** if not 0-1 domains, split on $\lfloor x_i^* \rfloor$ and $\lceil x_i^* \rceil$

Branch and Bound Algorithm

Algorithm Options

- How to select the next active node?
  - Leads to different ways of exploring the search space

Algorithm Options

- How to select the next active node?
  - Leads to different ways of exploring the search space
  - The usual method is to make a best-first search based on the optimal solution of the LPR

Branch and Bound Algorithm

Algorithm Options

- How to select the next active node?
  - Leads to different ways of exploring the search space
  - The usual method is to make a best-first search based on the optimal solution of the LPR
  - At each step, choose the active node with the smallest optimal value of the LPR (minimization problem)

Branch and Bound Algorithm

Algorithm Options

- How to select the next active node?
  - Leads to different ways of exploring the search space
  - The usual method is to make a best-first search based on the optimal solution of the LPR
  - At each step, choose the active node with the smallest optimal value of the LPR (minimization problem)
  - Goal: try to quickly obtain a low value solution in order to prune the search

Branch and Bound Algorithm

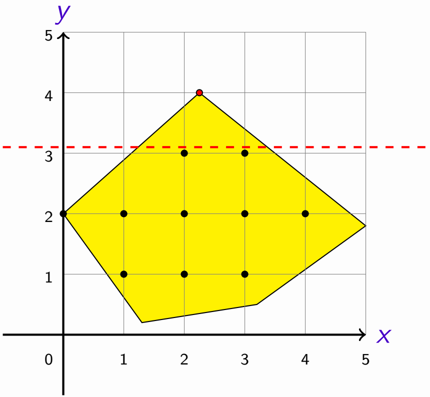
Other Options

- Use other lower bound estimation procedures instead of linear programming relaxation to bound the search (e.g. Lagrangian relaxation)
- Use inference methods (e.g. Boolean propagation, cutting planes)

Outline

Motivation

To further prune the space,
cut, out the non-integer solution, by adding a new constraint



Cutting Planes

Gomory Cuts [Gomory, 58]

- Takes advantage of the domain of variables to be integer
- Rounding operation
- Apply rounding in order to exclude the solution of the LPR, but keep all integer solutions
- Several versions exist that produce cuts of different strength

Cutting Planes

Combination of two constraints

$$\frac{\begin{array}{c}\delta(\sum_{j=1}^{n} a_j x_j \leq b) \\ \delta'(\sum_{j=1}^{n} a'_j x_j \leq b')\end{array}}{\delta \sum_{j=1}^{n} a_j x_j + \delta' \sum_{j=1}^{n} a'_j x_j \leq \delta b + \delta' b'}$$

Cutting Planes

Example

$$\frac{\begin{array}{r} 1(x_4 + 3x_5 + 2x_3 \quad \leq 3) \\ 2(x_1 + x_2 + \neg x_3 \quad \leq 1) \end{array}}{2x_1 + 2x_2 + x_4 + 3x_5 \leq 3}$$

- $\neg x_3$ is replaced with $1 - x_3$
- Notice that x_3 does not occur in the new constraint
- The cutting plane operation in Pseudo-Boolean solving corresponds to the CNF clause resolution

Cutting Planes

Rounding can also be applied

$$\frac{\sum_{j=1}^{n} a_j x_j \leq b}{\sum_{j=1}^{n} \lfloor a_j \rfloor x_j \leq \lfloor b \rfloor}$$

- The correctness of the rounding operation follows from $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor$
- Hence, δ coefficients in cutting plane operations do not need to be integer. Rounding can be safely applied afterwards

Rounding Example

$$\frac{0.5(3x_1 + 2x_2 + x_3 + 2x_4 + x_5 \quad \leq 5)}{1.5x_1 + x_2 + 0.5x_3 + x_4 + 0.5x_5 \leq 2.5}$$

After rounding: $x_1 + x_2 + x_4 \leq 2$

Cutting Planes

Use of Cutting Planes

- Used in branch and bound algorithms for PBO
  - And in the more general case of Integer Linear Programming (ILP)
- Very common at preprocessing (i.e., at the root node of the search tree)
- Algorithms that use cutting plane techniques during the search process are also known as branch and cut algorithms
- Other types of cutting planes exist (e.g., clique cuts)

Cutting Planes

Backtrack search with Cutting Plane learning

- DPLL-like algorithms for PBO can perform cutting plane learning instead of clause learning
- Replace clause resolution with cutting planes in implication graph analysis
- Important note: It is not guaranteed that the new constraint will be assertive

Cutting Planes

Backtrack search with Cutting Plane learning

Consider the following constraints:

$$\begin{aligned} 3x_1 + x_2 + x_7 + 2\neg x_8 &\leq 5 \\ 3\neg x_1 + x_3 + 2x_7 + x_9 &\leq 3 \\ \neg x_2 + \neg x_3 + x_6 &\leq 1 \end{aligned}$$

- Suppose you start with assignment $x_8 = 0$ at first decision level

Cutting Planes

Backtrack search with Cutting Plane learning

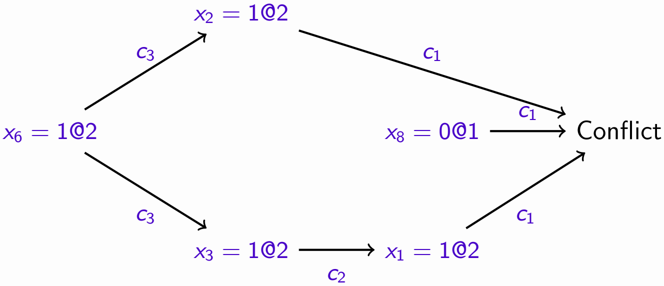
Consider the following constraints:

$$\begin{aligned} 3x_1 + x_2 + x_7 + 2\neg x_8 &\leq 5 \\ 3\neg x_1 + x_3 + 2x_7 + x_9 &\leq 3 \\ \neg x_2 + \neg x_3 + x_6 &\leq 1 \end{aligned}$$

- Suppose you start with assignment $x_8 = 0$ at first decision level
- Next, you decide to assign $x_6 = 1$. What happens?

Backtrack search with Cutting Plane learning

$$
\begin{aligned}
c_1 : \quad & 3x_1 + x_2 + x_7 + 2\neg x_8 && \leq 5 \\
c_2 : \quad & 3\neg x_1 + x_3 + 2x_7 + x_9 && \leq 3 \\
c_3 : \quad & \neg x_2 + \neg x_3 + x_6 && \leq 1
\end{aligned}
$$



Conflict in constraint c_1
Start backward traversal of graph

$$\begin{array}{lll} c_1: & 3x_1 + x_2 + x_7 + 2\neg x_8 & \leq 5 \\ c_2: & 3\neg x_1 + x_3 + 2x_7 + x_9 & \leq 3 \\ c_3: & \neg x_2 + \neg x_3 + x_6 & \leq 1 \end{array}$$

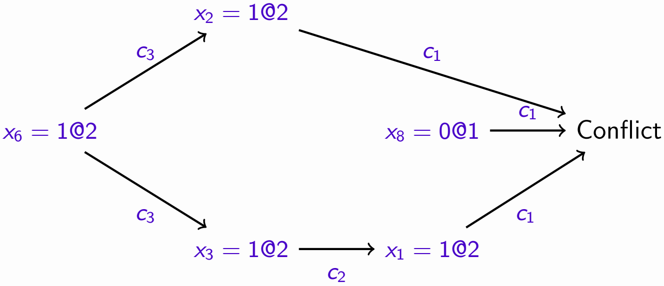
$$\begin{tikzcd}
 & x_2 = 1@2 \arrow[drr, "c_1"] & & \\
x_6 = 1@2 \arrow[ur, "c_3"] \arrow[dr, "c_3"'] & & x_8 = 0@1 \arrow[r, "c_1"] & \text{Conflict} \\
 & x_3 = 1@2 \arrow[r, "c_2"'] & x_1 = 1@2 \arrow[ur, "c_1"'] &
\end{tikzcd}$$

Cutting plane between c_1 and c_2 to remove x_1

$$\begin{array}{l} 1(3x_1 + x_2 + x_7 + 2\neg x_8 \quad \leq 5) \\ 1(3\neg x_1 + x_3 + 2x_7 + x_9 \quad \leq 3) \\ \hline x_2 + x_3 + 3x_7 + 2\neg x_8 + x_9 \leq 5 \end{array}$$

Backtrack search with Cutting Plane learning

$$
\begin{aligned}
c_1 &: \quad 3x_1 + x_2 + x_7 + 2\neg x_8 &&\leq 5 \\
c_2 &: \quad 3\neg x_1 + x_3 + 2x_7 + x_9 &&\leq 3 \\
c_3 &: \quad \neg x_2 + \neg x_3 + x_6 &&\leq 1
\end{aligned}
$$

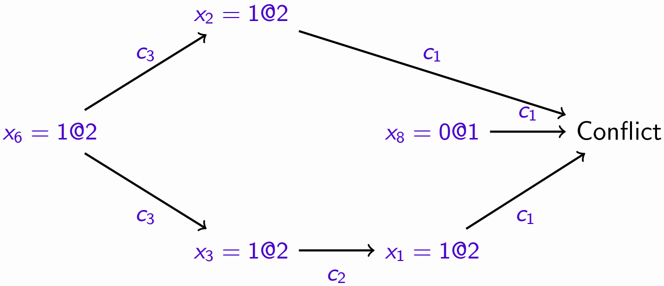


Cutting plane with c_3 to remove x_3

$$
\begin{array}{lr}
1(x_2 + x_3 + 3x_7 + 2\neg x_8 + x_9 & \leq 5) \\
1(\neg x_2 + \neg x_3 + x_6 & \leq 1) \\
\hline
\multicolumn{2}{c}{x_6 + 3x_7 + 2\neg x_8 + x_9 \leq 4}
\end{array}
$$

Backtrack search with Cutting Plane learning

$$
\begin{array}{llr}
c_1: & 3x_1 + x_2 + x_7 + 2\neg x_8 & \leq 5 \\
c_2: & 3\neg x_1 + x_3 + 2x_7 + x_9 & \leq 3 \\
c_3: & \neg x_2 + \neg x_3 + x_6 & \leq 1
\end{array}
$$



Backward traversal to the decision variable x_6

Learned constraint: $x_6 + 3x_7 + 2\neg x_8 + x_9 \leq 4$

Backtrack to level 1 and imply $x_7 = 0$

Summary

Pseudo-Boolean Optimization - Summary

- Pseudo-Boolean Optimization formulations
- Algorithmic strategies to solve PBO
- Branch and Bound algorithms
  - Usage of linear programming relaxations to cut the search tree
- Cutting plane generation
  - Branch and cut
  - Constraint learning in backtrack search