Discrete Optimization: Modelling

https://bicmr.pku.edu.cn/~wenzw/bigdata2022.html

Acknowledgement: this slides is based on Prof. Ted Ralphs's lecture notes

Outline

Mixed Integer Linear Programming

- Consider linear programming with additionally constraints

$$X = \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}.$$

- The general form of such a mathematical optimization problem is

$$z_{IP} = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}\},$$

  where for $A \in \mathbb{Q}^{m\times n}, b \in \mathbb{Q}^m, c \in \mathbb{Q}^n$.

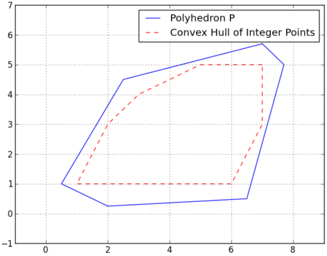
- This type of optimization problem is called a mixed integer linear programming (MILP) problem.
- If $p = n$, then we have a pure integer linear optimization problem.
- Special case: the integer variables are binary, i.e., 0 or 1.

The Geometry of Integer Programming

- Let's consider an integer linear program

$$
\begin{aligned}
\max \quad & c^\top x \\
\text{s.t.} \quad & Ax \le b \\
& x \in \mathbb{Z}^n_+
\end{aligned}
$$

- The feasible region is the integer points inside a polyhedron.



- Why does solving the LP relaxation not necessarily yield a good solution?

How Hard is Integer Programming?

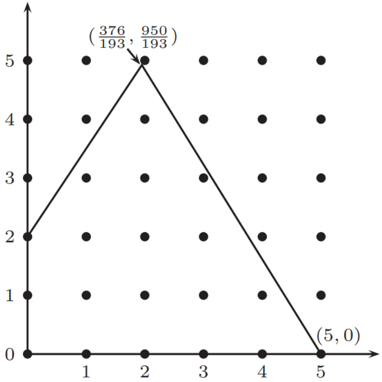
- Solving general integer programs can be much more difficult than solving linear programs.
- There in no known polynomial-time algorithm for solving general MIPs.
- Solving the associated linear programming relaxation results in an upper bound on the optimal solution to the MIP.
- In general, solving the LP relaxation, an LP obtained by dropping the integerality restrictions, does not tell us much.
  - Rounding to a feasible integer solution may be difficult.
  - The optimal solution to the LP relaxation can be arbitrarily far away from the optimal solution to the MIP.
  - Rounding may result in a solution far from optimal.
  - We can bound the difference between the optimal solution to the LP and the optimal solution to the MIP (how?).

How Hard is Integer Programming?

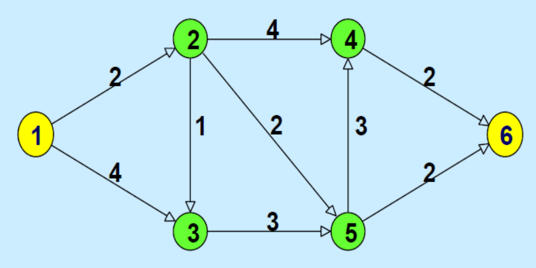
Consider the integer program

$$\begin{aligned} \max \quad & 50x_1 + 32x_2, \\ \text{s.t.} \quad & 50x_1 + 31x_2 \leq 250, \\ & 3x_1 - 2x_2 \geq -4, \\ & x_1, x_2 \geq 0 \text{ and integer.} \end{aligned}$$

The linear programming solution $(376\backslash 193, 950\backslash 193)$ is a long way from the optimal integer solution $(5, 0)$.

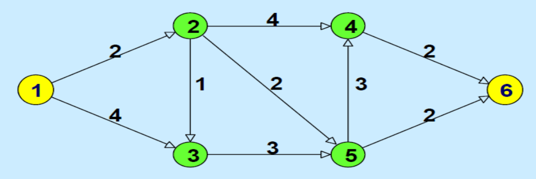


The shortest path problem



- Consider a network G = (N, A) with cost c_{ij} on each edge $(i, j) \in A$. There is an origin node s and a destination node t.
- Standard notation: n = |N|, m = |A|
- cost of of a path: $c(P) = \sum_{(i,j)\in P} c_{ij}$
- What is the shortest path from s to t?

The shortest path problem



$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t. $\sum_j x_{sj} = 1$

$$\sum_j x_{ij} - \sum_j x_{ji} = 0, \text{ for each i } \neq s \text{ or } t$$

$$-\sum_i x_{it} = -1$$

$x_{ij} \in \{0,1\}$ for all (i,j)

Conjunction versus Disjunction

- A more general mathematical view that ties integer programming to logic is to think of integer variables as expressing **disjunction**.
- The constraints of a standard mathematical program are **conjunctive**.
  - All constraints must be satisfied.

$$g_1(x) \leq b_1 \text{ AND } g_2(x) \leq b_1 \text{ AND } \cdots \text{ AND } g_m(x) \leq b_m$$

  - This corresponds to intersection of the regions associated with each constraint.
- Integer variables introduce the possibility to model disjunction.
  - At least one constraint must be satisfied.

$$g_1(x) \leq b_1 \text{ OR } g_2(x) \leq b_1 \text{ OR } \cdots \text{ OR } g_m(x) \leq b_m$$

  - This corresponds to union of the regions associated with each constraint.

Representability Theorem

The connection between integer programming and disjunction is captured most elegantly by the following theorem.

Theorem

A set $\mathcal{F} \subseteq \mathbb{R}^n$ is MIP representable if and only if there exist rational polytopes $\mathcal{P}_1, \cdots, \mathcal{P}_k$ and vectors $r^1, \cdots, r^t \in \mathbb{Z}^n$ such that

$$\mathcal{F} = \bigcup_{i=1}^{n} \mathcal{P}_i + \text{intcone}\{r^1, \cdots, r^t\}.$$

where $\text{intcone}\{r^1, \cdots, r^t\} = \left\{\sum_{i=1}^t \lambda_i r_i \mid \lambda \in \mathbb{Z}_+^t\right\}$

Roughly speaking, we are optimizing over a union of polyhedra, which can be obtained simply by introducing a disjunctive logical operator to the language of linear programming.

Outline

Modeling with Integer Variables

- From a practical standpoint, why do we need integer variables?
- Integer variable essentially allow us to introduce disjunctive logic.
- If the variable is associated with a physical entity that is indivisible, then the value must be integer.
- At its heart, integrality is a kind of disjunctive constraint.
- 0-1 (binary) variables are often used to model more abstract kinds of disjunctions (non-numerical).
  - Modeling yes/no decisions.
  - Enforcing logical conditions.
  - Modeling fixed costs.
  - Modeling piecewise linear functions.

Modeling Binary Choice

- We use binary variables to model yes/no decisions.
- Example: Integer knapsack problem
  - We are given a set of items with associated values and weights.
  - We wish to select a subset of maximum value such that the total weight is less than a constant K.
  - We associate a 0-1 variable with each item indicating whether it is selected or not.

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{m} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{m} w_j x_j \leq K \\
& x \in \{0, 1\}^n
\end{aligned}
$$

Modeling Dependent Decisions

- We can also use binary variables to enforce the condition that a certain action can only be taken if some other action is also taken.
- Suppose x and y are binary variables representing whether or not to take certain actions.
- The constraint $x \leq y$ says "only take action x if action y is also taken"

MIP reformulation of ℓ_0-minimization

- Big-M assumption: $\forall i$, $|x_i| \leq M$

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad \|x\|_0 \leq k, |x_i| \leq M$$

- MIP formulation:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad \sum_{i=1}^{n} y_i \leq k, |x_i| \leq M y_i, y_i \in \{0, 1\}$$

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:

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

Selecting from a Set

- We can use constraints of the form $\sum_{j \in T} x_j \geq 1$ to represent that at least one item should be chosen from a set T.
- Similarly, we can also model that at most one or exactly one item should be chosen.
- Example: Set covering problem
  - A set covering problem is any problem of the form.

$$\min \quad \{c^\top x \mid Ax \geq 1, x_j \in \{0,1\}\}$$

    where A is a 0-1 matrix.
  - Each row of A represents an item from a set S.
  - Each column A_j represents a subset S_j of the items.
  - Each variable x_j represents selecting subset S_j.
  - In other words, each item must appear in at least one selected subset.

Modeling Disjunctive Constraints

- We are given two constraints $a^\top x \geq b$ and $c^\top x \geq d$ with nonnegative coefficients.
- Instead of insisting both constraints be satisfied, we want at least one of the two constraints to be satisfied.
- To model this, we define a binary variable y and impose

$$\begin{aligned} a^\top x &\geq yb, \\ c^\top x &\geq (1-y)d, \\ y &\in \{0,1\}. \end{aligned}$$

- More generally, we can impose that at least k out of m constraints be satisfied with

$$\begin{aligned} a_i^\top x &\geq y_i b_i, \\ \sum_{i=1}^{m} y_i &\geq k, \\ y_i &\in \{0,1\}. \end{aligned}$$

Modeling Disjunctive Constraints (cont'd)

- Consider the disjunctive constraints $a^\top x \geq b$ and $c^\top x \geq d$ where the coefficients are allowed to be negative.
- To model this, we use the Big-M Reformulation. we define a binary variable y and impose

$$
\begin{aligned}
a^\top x &\geq b - My,\\
c^\top x &\geq d - M(1-y),\\
y &\in \{0,1\}.
\end{aligned}
$$

where M is a sufficiently large positive number.

Modeling a Restricted Set of Values

- We may want variable x to only take on values in the set $\{a_1, \cdots, a_m\}$.
- We introduce m binary variables y_j, $j = 1, \cdots, m$ and the constraints

$$x = \sum_{j=1}^{m} a_j y_j,$$

$$\sum_{j=1}^{m} y_j = 1,$$

$$y_j \in \{0, 1\}.$$

Fixed-charge Problems

- In many instances, there is a fixed cost and a variable cost associated with a particular decision.
- Example: Fixed-charge Network Flow Problem
  - We are given a directed graph $G = (N, A)$.
  - There is a fixed cost c_{ij} associated with "opening" arc (i, j) (think of this as the cost to "build" the link).
  - There is also a variable cost d_{ij} associated with each unit of flow along arc (i, j).
  - Consider an instance with a single supply node.
    - Minimizing the fixed cost by itself is a minimum spanning tree problem (easy).
    - Minimizing the variable cost by itself is a minimum cost network flow problem (easy).
    - We want to minimize the sum of these two costs (difficult).

Modeling the Fixed-charge Network Flow Problem

- To model the FCNFP, we associate two variables with each arc.
  - x_{ij} (fixed-charge variable) indicates whether arc (i,j) is open.
  - f_{ij} (flow variable) represents the flow on arc (i,j).
  - Note that we have to ensure that $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} + d_{ij}f_{ij} & \\
\text{s.t.} \quad & \sum_{j\in O(i)} f_{ij} - \sum_{j\in I(i)} f_{ji} = b_i, & \forall i \in N \\
& f_{ij} \leq Cx_{ij}, & \forall (i,j) \in A \\
& f_{ij} \geq 0, & \forall (i,j) \in A \\
& x_{ij} \in \{0,1\}, & \forall (i,j) \in A
\end{aligned}
$$

Alternative Formulations

- A key concept in the rest of the course will be that every mathematical model has many alternative formulations.
- Many of the key methodologies in integer programming are essentially automatic methods of reformulating a given model.
- The goal of the reformulation is to make the model easier to solve.

Simple Example: Knapsack Problem

- We are given a set $N = \{1, \cdots, n\}$ of items and a capacity K.
- There is a profit c_i and a size w_i associated with each item $i \in N$.
- We want to choose the set of items that maximizes profit subject to the constraint that their total size does not exceed the capacity.
- The most straightforward formulation is to introduce a binary variable x_i associated with each item.
- x_i takes value 1 if item i is chosen and 0 otherwise.
- Then the formulation is

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} w_j x_j \leq K, \\
& x_i \in \{0, 1\}, \quad \forall i
\end{aligned}
$$

An Alternative Formulation

- Let us call a set $C \subseteq N$ a cover is $\sum_{i \in C} w_i > K$.
- Further, a cover C is minimal if $\sum_{i \in C \setminus \{j\}} w_i \leq K$ for all $j \in C$.
- Then we claim that the following is also a valid formulation of the original problem.

$$\begin{array}{lll} \max & \sum_{j=1}^{n} c_j x_j, & \\ \text{s.t.} & \sum_{j \in C} x_j \leq |C| - 1, & \text{for all minimal covers } C \\ & x_i \in \{0, 1\}, \quad i \in N & \end{array}$$

- Which formulation is "better"?

Back to the Facility Location Problem

- Here is another formulation for the same problem:

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

- 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 (how do we prove this?).
- Therefore, the second polyhedron will yield a better lower bound.
- The second polyhedron is a better approximation to the convex hull of integer solutions.

Formulation Strength and Ideal Formulations

- Consider two formulations A and B for the same ILP.
- Denote the feasible regions corresponding to their LP relaxations as $\mathcal{P}_A$ and $\mathcal{P}_B$.
- Formulation A is said to be at least as strong as formulation B if $\mathcal{P}_A \subseteq \mathcal{P}_B$
- If the inclusion is strict, then A is stronger than B.
- If $\mathcal{S}$ is the set of all feasible integer solutions for the ILP, then we must have $\mathrm{conv}(\mathcal{S}) \subseteq \mathcal{P}_A$ (why?).
- A is ideal if $\mathrm{conv}(\mathcal{S}) = \mathcal{P}_A$.
- If we know an ideal formulation, we can solve the IP (why?).
- How do our formulations of the knapsack problem compare by this measure?

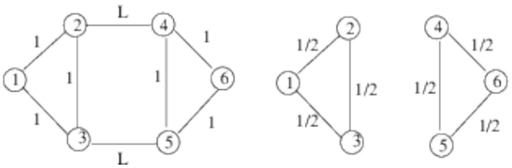
Strengthening Formulations

- Often, a given formulation can be strengthened with additional inequalities satisfied by all feasible integer solutions.

- Example: given a graph G = (V, E), a perfect matching in G is a subset M of edge set E, such that every vertex in V is adjacent to exactly one edge in M.
  - We are given a set of n people that need to paired in teams of two.
  - Let c_{ij} represent the "cost" of the team formed by person i and person j.
  - The nodes represent the people and the edges represent pairings.
  - We have $x_e = 1$ if the endpoints of e are matched, $x_e = 0$ otherwise.

$$
\begin{aligned}
\min \quad & \sum_{e=\{i,j\}\in E} c_e x_e & \\
\text{s.t.} \quad & \sum_{\{j|\{i,j\}\in E\}} x_{ij} = 1, & \forall i \in N \\
& x_e \in \{0,1\}, & \forall e = \{i,j\} \in E
\end{aligned}
$$

Valid Inequalities for Matching



- Consider the graph on the left above.
- The optimal perfect matching has value L + 2.
- The optimal solution to the LP relaxation has value 3.
- This formulation can be extremely weak.
- Add the valid inequality $x_{24} + x_{35} \geq 1$.
- Every perfect matching satisfies this inequality.

The Odd Set Inequalities

- We can generalize the inequality from the last slide.
- Consider the cut S corresponding to any odd set of nodes.
- The cutset corresponding to S is

$$\delta(S) = \{\{i,j\} \in E | i \in S, j \notin S\}.$$

- An odd cutset is any $\delta(S)$ for which the $|S|$ is odd.
- Note that every perfect matching contains at least one edge from every odd cutset.
- Hence, each odd cutset induces a possible valid inequality.

$$\sum_{e \in \delta(S)} x_e \geq 1, S \subset N, |S| \text{ odd.}$$

Using the New Formulation

- If we add all of the odd set inequalities, the new formulation is ideal.
- Hence, we can solve this LP and get a solution to the IP.
- However, the number of inequalities is exponential in size, so this is not really practical.
- Recall that only a small number of these inequalities will be active at the optimal solution.
- Later, we will see how we can efficiently generate these inequalities on the fly to solve the IP

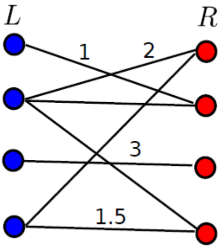
Contrast with Linear Programming

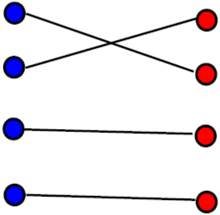
- In linear programming, the same problem can also have multiple formulations.
- In LP, however, conventional wisdom is that bigger formulations take longer to solve.
- In IP, this conventional wisdom does not hold.
- We have already seen two examples where it is not valid.
- Generally speaking, the size of the formulation does not determine how difficult the IP is.

The Max-Weight Bipartite Matching Problem

Given a bipartite graph G = (N, A), with N = L $\cup$ R, and weights w_{ij} on edges (i,j), find a maximum weight matching.

- Matching: a set of edges covering each node at most once
- Let n=|N| and m = |A|.
- Equivalent to maximum weight / minimum cost perfect matching.





The Max-Weight Bipartite Matching

Integer Programming (IP) formulation

$$\begin{aligned}
\max \quad & \sum_{ij} w_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j} x_{ij} \le 1, \forall i \in L \\
& \sum_{i} x_{ij} \le 1, \forall j \in R \\
& x_{ij} \in \{0, 1\}, \forall (i, j) \in A
\end{aligned}$$

- $x_{ij} = 1$ indicate that we include edge (i, j) in the matching
- IP: non-convex feasible set

The Max-Weight Bipartite Matching

Integer program (IP)

$$
\begin{aligned}
\max \quad & \sum_{ij} w_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j} x_{ij} \le 1, \forall i \in L \\
& \sum_{i} x_{ij} \le 1, \forall j \in R \\
& x_{ij} \in \{0, 1\}, \forall (i,j) \in A
\end{aligned}
$$

LP relaxation

$$
\begin{aligned}
\max \quad & \sum_{ij} w_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j} x_{ij} \le 1, \forall i \in L \\
& \sum_{i} x_{ij} \le 1, \forall j \in R \\
& x_{ij} \ge 0, \forall (i,j) \in A
\end{aligned}
$$

- Theorem. The feasible region of the matching LP is the convex hull of indicator vectors of matchings.
- This is the strongest guarantee you could hope for an LP relaxation of a combinatorial problem
- Solving LP is equivalent to solving the combinatorial problem

Primal-Dual Interpretation

Primal LP relaxation

$$
\begin{aligned}
\max \quad & \sum_{ij} w_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, \forall i \in L \\
& \sum_{i} x_{ij} \leq 1, \forall j \in R \\
& x_{ij} \geq 0, \forall (i,j) \in A
\end{aligned}
$$

Dual

$$
\begin{aligned}
\min \quad & \sum_{i} y_i \\
\text{s.t.} \quad & y_i + y_j \geq w_{ij}, \forall (i,j) \in A \\
& y \geq 0
\end{aligned}
$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$

Primal-Dual Interpretation

Suppose edge weights $w_{ij} = 1$, then binary solutions to the dual are node covers.

Dual

$$\begin{aligned} \min \quad & \sum_i y_i \\ \text{s.t. } & y_i + y_j \geq 1, \forall (i,j) \in A \\ & y \geq 0 \end{aligned}$$

Dual Integer Program

$$\begin{aligned} \min \quad & \sum_i y_i \\ \text{s.t. } & y_i + y_j \geq 1, \forall (i,j) \in A \\ & y \in \{0, 1\} \end{aligned}$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$
- Consider IP formulation of the dual, then

$$P^*_{IP} \leq P^*_{LP} = D^*_{LP} \leq D^*_{IP}$$

Total Unimodularity

Defintion: A matrix A is Totally Unimodular if every square submatrix has determinant 0, +1 or -1.

Theorem: If $A \in \mathbb{R}^{m \times n}$ is totally unimodular, and b is an integer vector, then $\{x : Ax \leq b; x \geq 0\}$ has integer vertices.

- Non-zero entries of vertex x are solution of $A'x' = b'$ for some nonsignular square submatrix A' and corresponding sub-vector b'
- Cramer's rule:

$$x_i = \frac{\det(A'_i \mid b')}{\det A'}$$

Claim: The constraint matrix of the bipartite matching LP is totally unimodular.

The Minimum weight vertex cover

- undirected graph G = (N, A) with node weights $w_i \geq 0$
- A vertex cover is a set of nodes S such that each edge has at least one end in S
- The weight of a vertex cover is sum of all weights of nodes in the cover
- Find the vertex cover with minimum weight

Integer Program

$$\min \quad \sum_i w_i y_i$$
$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$
$$y \in \{0,1\}$$

LP Relaxation

$$\min \quad \sum_i w_i y_i$$
$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$
$$y \geq 0$$

LP Relaxation for the Minimum weight vertex cover

- In the LP relaxation, we do not need $y \leq 1$, since the optimal solution y^* of the LP does not change if $y \leq 1$ is added.
  Proof: suppose that there exists an index i such that the optimal solution of the LP y_i^* is strictly larger than one. Then, let y' be a vector which is same as y^* except for $y_i' = 1 < y_i^*$. This y' satisfies all the constraints, and the objective function is smaller.
- The solution of the relaxed LP may not be integer, i.e., $0 < y_i^* < 1$
- rounding technique:

$$y_i' = \begin{cases} 0, & \text{if } y_i^* < 0.5 \\ 1, & \text{if } y_i^* \geq 0.5 \end{cases}$$

- The rounded solution y' is feasible to the original problem

LP Relaxation for the Minimum weight vertex cover

The weight of the vertex cover we get from rounding is at most twice as large as the minimum weight vertex cover.

- Note that $y'_i = \min(\lfloor 2y^*_i \rfloor, 1)$
- Let P^*_{IP} be the optimal solution for IP, and P^*_{LP} be the optimal solution for the LP relaxation
- Since any feasible solution for IP is also feasible in LP, $P^*_{LP} \leq P^*_{IP}$
- The rounded solution y' satisfy

$$\sum_i y'_i w_i = \sum_i \min(\lfloor 2y^*_i \rfloor, 1) w_i \leq \sum_i 2y^*_i w_i = 2P^*_{LP} \leq 2P^*_{IP}$$

Outline

1. Introduction to Integer Programming
2. Integer Programming Modeling and Formulation
3. Constraint Programming (CP)

Difference with mathematical programming

Problem:

- Solving combinatorial optimization problems (Decision Optimization)

Modeling:

- Declarative modeling paradigm
- Logical constraints & global constraints
- Integer, interval & boolean variables (maybe double variables)

Solving:

- Constructive search & domain reduction (propagation)
- Based on computer science (logic programming, graph theory, ...)

Solution:

- Feasible solution & optimal solution

A simple example: n-Queen Problem

The classic queens problem: placing n queens on an nxn checkerboard so that no two queens can attack each other, i.e., no two queens are on the same row, column, or diagonals.

Integer variable params:

- N: number of variables
- 0: minimum value
- N-1: maximum value
- "X": name prefix

constraints:

- all_diff: $x_i \neq x_j, \forall i \neq j$

DOcplex for n-Queen problem

```
# Create model
mdl = CpoModel()

# Create column index of each queen
x = mdl.integer_var_list (N, 0, N - 1, "X")

# One queen per row
mdl.add(mdl.all_diff (x))

# One queen per diagonal xi - xj != i - j
mdl.add(mdl.all_diff (x[i] + i for i in range(N)))

# One queen per diagonal xi - xj != j - i
mdl.add(mdl.all_diff (x[i] - i for i in range(N)))
```

High-level constraints: Global constraints

Global constraint captures complex relationships among multiple variables in a concise and efficient way.

alldifferent

- All variables in a set take distinct values
- Assignment problems
- alldifferent([x, y, z])
- $x \neq y, x \neq z, y \neq z$

table

- Tuple of variables takes values from predefined set
- table([x, y, z], [(1, 2, 3), (4, 5, 6)])
- (x, y, z) = (1, 2, 3) or (4, 5, 6)

circuit

- Sequence of variables forms a Hamitonian cycle
- Routing problems
- circuit(x)
- x=[0, 1, 3, 2, 0] means $0- > 1- > 3- > 2- > 0$

Arithmetic expressions and constraints

CP Optimizer supports integer variables and is possible to contain float-point expressions in constraints or objective function.

- operator +, -, *, /
- Sum
- Diff
- ScalProd
- Div
- Modulo(%)

- StandardDeviation
- Min
- Max
- Count
- CountDifferent
- Abs
- Element

- ==
- !=
- $<$
- $>$
- $<=$
- $>=$

Diff and Element

Diff & operator-

Automatic linearization by slack variables

```
IloNumExpr e1 = x * y;
IloNumExpr e2 = z / w;
IloNumExpr diff =  IloDiff (e1, e2);
model.add(diff == 0);
```

Element: y=array[x]

```
// Create the model
IloModel model(env);

// Define an array
 IloIntArray  array(env, 4);
array[0]  = 10;
array[1]  = 20;
array[2]  = 30;
array[3]  = 40;

// Define variables
 IloIntVar  x(env, 0, 3, "x");
 IloIntVar  y(env, 0, 100, "y");

// Add the element constraint
model.add(y == IloElement(array, x));
```

Logical and compatibility constraints

- &&
- ||
- Not
- IfThen
- AllowedAssignments
- ForbiddenAssignments

IfThen

```
model.add(IloIfThen(x >= 5, y <= 3));

// Nested structure
model.add(IloIfThen(x >= 5, IloIfThen(y <= 3, z == 0)));

// Combined with global constraints
model.add(IloIfThen(x != y,  IloAllDiff (env, x, y, z)));
```

AllowedAssignments

```
IloIntTupleSet allowed(env);
allowed.add(IloIntArray(env, 2, 1, 2));
allowed.add(IloIntArray(env, 2, 2, 3));

model.add(IloAllowedAssignments(vars, allowed));
```

Special constraints on integer variables

Theoretically, these special constraints can be written from arithmetic constraints and expressions, but they can also be designed and implemented to reduce domains efficiently during a search.

- AllDiff
- AllMinDistance
- Pack
- Inverse
- Lexicographic
- Distribute

Pack

```
IloIntVarArray bin(env, 3, 0, 1); // 3 items, 2 bins
IloIntArray size(env, 3); // Size of each item
size[0] = 2;
size[1] = 3;
size[2] = 4;
IloIntVarArray load(env, 2, 0, 5); // Max capacity is 5

model.add(IloPack(bin, size, load));
```

Interval variables

Interval variables

Static

- StartOf
- EndOf
- LengthOf
- SizeOf

Dynamic

- StartEval
- EndEval
- LengthEval
- SizeEval

StartOf

```
IloIntervalVar task(env, 10); // Task with duration 10
IloIntExpr start = IloStartOf(task); // Static start time
model.add(start >= 5); // Task must start after time 5
```

StartEval

```
IloIntervalVar task(env, 10);
IloIntVar condition(env, 0, 1); // 0 or 1

// Dynamic start time: if condition=1, start time increases by 5
IloIntExpr dynamicStart = IloStartEval(task) + condition * 5;

// Constraint depends on runtime value of 'condition'
model.add(dynamicStart <= 20);
```

Special constraints on interval variables

Forbidden constraints

- ForbidStart
- ForbidEnd
- ForbidExtent

Precedence constraints

- End/Start
+ Before/At
+ End/Start
- e.g. EndBeforeStart

Groups of interval variables

- PresenceOf
- Isomorphism
- Span
- Alternative
- Synchronize

Sequence constraints

- First
- Last
- Before
- Prev

Special constraints on interval variables

* CumulFunctionExpr
  - AlwaysIn/AlwaysEqual
  - AlwaysConstant
  - AlwaysNoState
  - operator$<=, >=$
* Pulse
* Step
* StepAtStart
* StepAtEnd
* HeightAtStart
* HeightAtEnd

Resource usage constraints

```
IloIntervalVar  task1(env, 10, "Task1");
IloIntervalVar  task2(env, 5, "Task2");

IloCumulFunctionExpr resourceUsage(env);
resourceUsage += IloPulse(task1, 2);
resourceUsage += IloPulse(task2, 1);
resourceUsage += IloStep(5, 3);
resourceUsage += IloStepAtStart(task1, 1);
resourceUsage += IloStepAtEnd(task2, -1);

model.add(resourceUsage <= 5);
```