Integer linear programming

- Maximizing or minimizing a linear function subject to linear inequality and equality constraints where some or all of the variables are required to be integral.

- Typical application: management and efficient use of scarce resources to increase productivity. Examples: production scheduling, machine sequencing, portfolio analysis, VLSI circuit design, problems in molecular biology, high energy physics and X-ray crystallography...

- Computing an optimal solution of an integer linear program is NP-complete.

- Important: developing structured ILP formulations.

Mathematical Foundations

- Let $S \subseteq \mathbb{R}^n$. A point $x \in \mathbb{R}^n$ is a *convex combination* of points of S if there is a finite set $\{x^i\}_{i=1}^t \in S$ and a $\lambda \in \mathbb{R}^t$ with $\sum_{i=1}^t \lambda_i = 1$, $x = \sum_{i=1}^t \lambda_i x^i$ and $\lambda_i \geq 0$ for all $i = 1, \ldots, t$. If $\lambda_i > 0$ for all $i = 1, \ldots, t$, x is called strict convex combination of S. The convex hull of S, $conv(S)$, is the set of all convex combinations of points in S.

- A set of points $x^1, \ldots, x^k \in \mathbb{R}^n$ is called *linearly independent*, if the unique solution of $\sum_{i=1}^k \lambda_i x^i = 0$, for $\lambda_i \in \mathbb{R}$ is $\lambda_i = 0$ for all $i = 1, \ldots, k$.

- A set of points $x^1, \ldots, x^k \in \mathbb{R}^n$ is called *affinely independent* if the unique solution of $\sum_{i=1}^k \alpha_i x^i = 0$, $\alpha_i \in \mathbb{R}$, $\sum_{i=1}^k \alpha_i = 0$ is $\alpha_i = 0$ for all $i = 1, \ldots, k$.

- A *polyhedron* $P \subseteq \mathbb{R}^n$ is a set of points satisfying a finite number of linear inequalities; $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, where (A, b) is an $m \times (n+1)$-matrix. A polyhedron is called *rational*, if there is an $m' \times (n+1)$-matrix (A', b') with rational coefficients such that $P = \{x \in \mathbb{R}^n \mid A'x \leq b'\}$.

- A point $x \in P$ is called *vertex* of the polyhedron P, if x cannot be represented as a strict convex combination of two different points from P.

- A polyhedron $P \subseteq \mathbb{R}^n$ is *bounded* if there is an $\omega \in \mathbb{R}_+$, such that $P \subseteq \{x \in \mathbb{R}^n \mid -\omega \leq x_j \leq \omega \text{ for } j = 1, \ldots, n\}$. A bounded polyhedron is called a *polytope*.

- A polyhedron P is of *dimension* k, $dim(P) = k$, if the maximum number of affinely independent points in P is $k+1$.

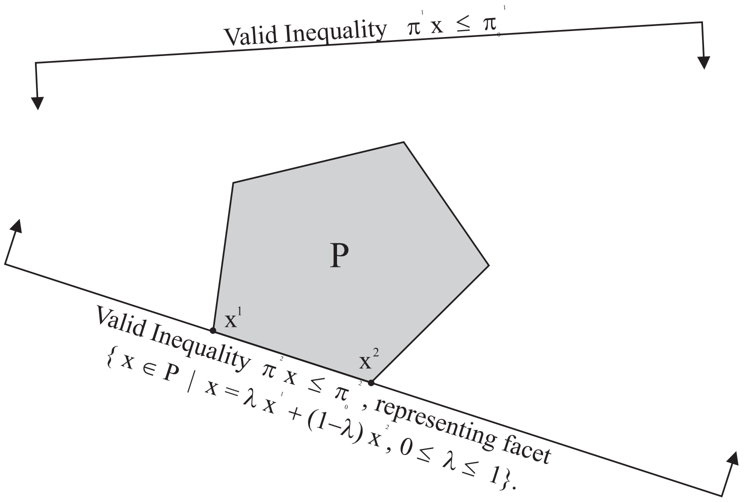

Figure 1: Valid inequalities and faces.

- An inequality $a^i x \leq b_i$ is called *valid* for P, if it is satisfied by all points in P.

- If $a^i x \leq b_i$ is a valid inequality for P and $F = \{x \in P \mid a^i x = b_i\}$, then F is called a *face* of P and we say that $a^i x \leq b_i$ represents the set F. A face F is said to be *proper*, if $F \neq \emptyset$ and $F \neq P$. A face F of P is a *facet* of P if $dim(F) = dim(P) - 1$.
- P can be characterized by inequalities representing its facets. To each polyhedron P there is a finite set of linear inequalities forming a minimal representation of P.

Linear Programming

- Linear programming algorithms are often used as a subroutine in integer programming algorithms to obtain bounds on the optimal value of the integer program.
- Corresponding to any integer programming problem there is a linear programming problem that has the same solution as the integer problem.
- Linear programming: minimize/maximize a linear objective function subject to a finite number of linear inequalities.

$$\begin{aligned} min \quad z_{LP} &= c^T x \\ Ax &\geq b \\ x &\in \mathbb{R}^n_+ \end{aligned} \tag{1}$$

where

$$c \in \mathbb{R}^n,\ b \in \mathbb{R}^m,\ A \in \mathbb{R}^{m \times n}.$$

- $S = \{x \in \mathbb{R}^n_+ \mid Ax \geq b\}$: *feasible region*,
- $\hat{x} \in S$: *feasible solution*.
- The feasible region of a linear program is a convex polyhedron.
- An instance of the problem is said to be *feasible* if $S \neq \emptyset$.
- $z_{LP} = c^T x$: *objective function*
- *optimal solution*: $\hat{x} \in S$ that minimises z_{LP}
- An instance of (1) is *unbounded*, if for all $\omega \in \mathbb{R}$ there is an $\hat{x} \in S$ with $c^T \hat{x} < \omega$.

Algorithms for Linear Programming

- Simplex algorithm (Dantzig, 1951): worst-case complexit exponential but in practice often about linear.
- Ellipsoid method (Khachiyan, 1980): polynomial but computationally impractical.
- Projective algorithm (Karmarkar, 1984): polynomial.

The Simplex Algorithm

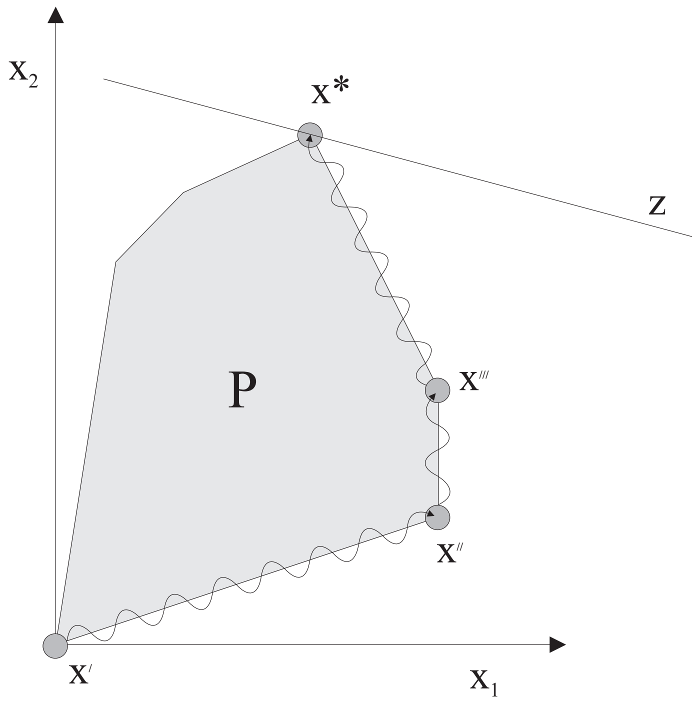

Figure 2: Simplex Algorithm in two-dimensional case.

- Constraints of a linear program define polytope in $\mathbb{R}^n$.
- Simplex Theorem: at least one of the vertices of this polytope constitutes an optimal solution.

- To find such a vertex: choose some vertex as starting point and go to a neighboring vertex if it yields a better objective function value.
- Repeat until a vertex is reached corresponding to an optimal solution.

Theory of Integer Linear Programming

- *Mixed integer linear program* (MIP):

$$\min \{c^T x + h^T y \mid Ax + Gy \geq b;\ x \in \mathbb{Z}^n_+,\ y \in \mathbb{R}^p_+\} \qquad (2)$$

  where

$$c \in \mathbb{R}^n,\ b \in \mathbb{R}^m,\ h \in \mathbb{R}^p,\ A \in \mathbb{R}^{m\times n},\ G \in \mathbb{R}^{m\times p}$$

- Special case without continuous variables: pure integer linear program (ILP).

$$\begin{aligned} \min \quad z_{IP} &= c^T x \\ x &\in P_F \cap \mathbb{Z}^n \end{aligned} \qquad (3)$$

  where

$$P_F = \{x \mid Ax \geq b,\ x \in \mathbb{R}^n_+\},\ c \in \mathbb{R}^n,\ b \in \mathbb{R}^m,\ A \in \mathbb{R}^{m\times n}$$

- The optimal solution of an integer linear program can be computed by solving the following problem:

$$\begin{aligned} \min \quad z_{IP} &= c^T x \\ x &\in P_I \end{aligned} \qquad (4)$$

  where

$$P_I = conv(\{x \mid x \in P_F\ \cap \mathbb{Z}^n\}).$$

- Integer points within $P_F \equiv$ feasible solutions of the ILP.
- The feasible region of (3) only consists of the integer points, the feasible region of (4), P_I, c onsists of the convex hull of the integer points.

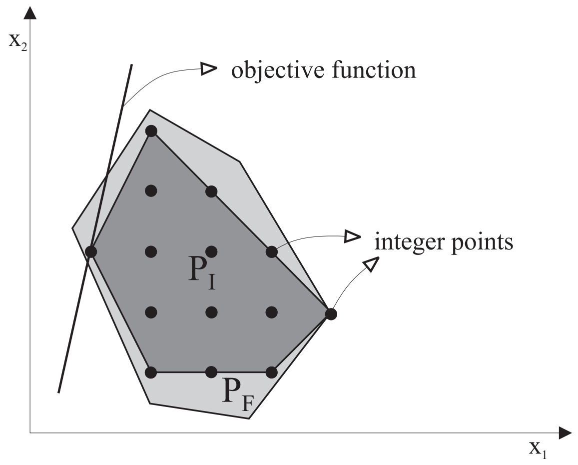

Figure 3: Feasible region.

- Let an optimisation problem Q with feasible region $X(Q)$ be given. An optimisation problem Q^R is called a *relaxation* of Q, if for the feasible region $X(Q^R)$ holds:

$$X(Q) \subset X(Q^R).$$

- Removing the integrality constraints of an ILP $\Rightarrow$ Linear program, called *LP-relaxation* of the ILP.
- LP-relaxation:

$$\begin{aligned} \min \quad z_R &= c^T x \\ x &\in P_F \end{aligned} \tag{5}$$

where

$$P_F = \{x \mid Ax \geq b,\ x \in \mathbb{R}^n_+\},\ c \in \mathbb{R}^n,\ b \in \mathbb{Z}^m,\ A \in \mathbb{Z}^{m \times n}$$

- $P_I \subseteq P_F \Rightarrow z_R \leq z_{IP}$.
- $P_F = P_I \Rightarrow P_F$ is called *integral*. Then $z_R = z_{IP}$ and optimal solution can be calculated in polynomial time by solving the LP-relaxation.

- In general: $P_I \subset P_F \Rightarrow$ LP relaxation yields a lower bound for the objective function.
- The efficiency of many algorithms for solving integer linear programs depends on the quality of this bound. The better P_F approximates the feasible area P_I the tighter is the bound. For efficiently solving an ILP formulation P_F should be close to P_I.

Some Results of Polyhedral Theory

- Coefficient matrix is totally unimodular $\Rightarrow$ corresponding polyhedron is integral.
- A square, integer matrix $B \in \mathbb{Z}^{n \times n}$ is called *unimodular* if for its determinant B holds $|B| = \pm 1$.
- An integer matrix $A \in \mathbb{R}^{m \times n}$ is called *totally unimodular* if every square, nonsingular submatrix of A is unimodular.
- Let A be a totally unimodular matrix and let b be an integral vector. If $P = \{x \in \mathbb{R}^n_+ \mid Ax \geq b\}$ is not empty, then P is an integral polyhedron.
- Let A be a totally unimodular matrix, and let b, b', d and d' be integral vectors. If $P = \{x \in \mathbb{R}^n_+ \mid b' \leq Ax \geq b, d' \leq x \leq d\}$ is not empty, then P is an integral polyhedron.
- Theorem of Hoffman and Kruskal: Let A be an integral matrix. Then A is totally unimodular, if and only if for *each* integral vector b the polyhedron $\{x \in \mathbb{R}^n_+ \mid Ax \leq b\}$ is integral.

Checking Total Unimodularity

- Let $A \in \{0, 1, -1\}^{m \times n}$. If A has no more than two nonzero entries in each column, and if $\sum_i a_{ij} = 0$ if column j contains two nonzero coefficients, then A is totally unimodular.
- The following statements are equivalent:
  - A is totally unimodular.
  - The transpose of A, A^T, is totally unimodular.

- A matrix obtained by multiplying a row (column) of A by -1 is totally unimodular.
- A matrix obtained by permuting rows (columns) of A is totally unimodular.
- A matrix obtained by duplicating rows (columns) of A is totally unimodular.

- Let $A \in \{0, 1, -1\}^{m \times n}$. Then A is totally unimodular, if and only if for every $J \subseteq N = \{1, \ldots, n\}$ there exists a partition J_1, J_2 of J such that

$$\left| \sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij} \right| \leq 1 \quad \text{for } i = 1, \ldots, m.$$

- Let $A \in \{0, 1\}^{m \times n}$. Then A is called an *interval matrix* if in each column the 1's appear consecutively; that is if $a_{ij} = a_{kj} = 1$ and $k > i + 1$, then $a_{ij} = 1$ for all l with $i < l < k$.

- Interval matrices are totally unimodular.

The Branch-And-Bound Algorithm

- Idea: Partition the feasible region S of the optimisation problem IP in a set of disjoint subsets $\{S^i \mid i = 1, \ldots, k\}$ and solve the problem over each of those subsets.
- Let a set S be given. The sets $\{S^i\}_{i=1}^k$ are said to be a *division* of S, if $\bigcup_{i=1}^k S^i = S$. A division is called *partition*, if $S^i \cap S^j = \emptyset$ for $i, j = 1, \ldots, k,\ i \neq j$.
- Let $\{S^i\}_{i=1}^k$ be a division of S and $z_{IP}^i = \min\{c^T x \mid x \in S^i\}$. Then for $z_{IP} = \min\{c^T x \mid x \in S\}$ holds

$$z_{IP} = \min_{i=1,\ldots,k} z_{IP}^i.$$

- $\Rightarrow$ *Divide and conquer* algorithm: A problem is partitioned into subproblems that are recursively solved by the same algorithm. The solution of the entire problem is composed from the solutions of the subproblems.
- In the Branch-And-Bound method for integer linear programming, the computation effort for the subproblems is restricted, e.g. by solving relaxations. This way it is not necessary to compute the optimal solution of an integer linear problem of its own at each node of the enumeration tree.
- Some pruning criteria for pruning the enumeration tree at the node corresponding to S^i:
    1. RP^i is infeasible.
    2. There is an optimal solution x_R^i of RP^i that satisfies $x_R^i \in S^i$ and $z_R^i = c^T x_R^i$.

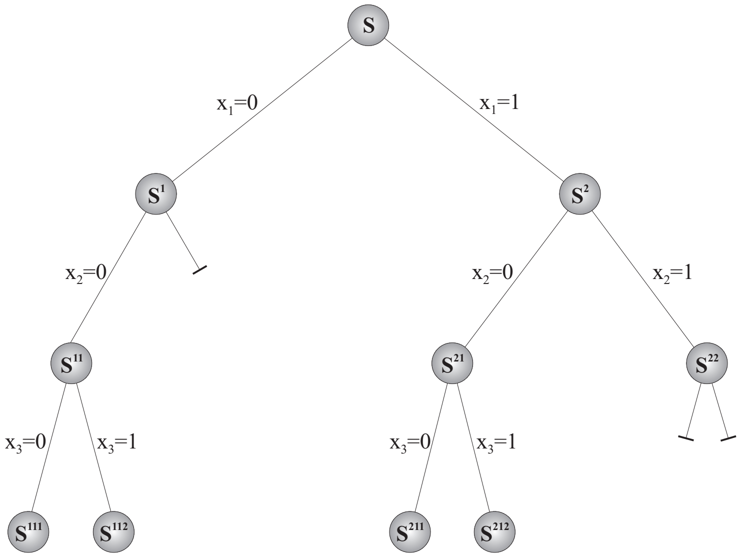

Figure 4: Example of a branch-and-bound tree.

3. $z_R^i \geq \overline{z_{IP}}$, where $\overline{z_{IP}}$ is the value of some feasible solution of IP.

- Let $L = \{IP^i\}$ be a set of integer linear programs of the form $z_{IP}^i = \min\{c^T x \mid x \in S^i\}$ where $S^i \subset S$. Each problem from L is associated with a lower bound $\underline{z^i} \leq z_{IP}^i$.

- Partitioning by *Dakin method*: Select a decision variable x_n and generate the subproblems IP^{i_1} and IP^{i_2} from IP. Let x^R denote an optimal solution of the relaxation of IP. The subproblem IP^{i_1} is built from IP by adding the inequality $x_n \leq \lfloor x_n^R \rfloor$; adding $x_n \geq \lceil x_n^R \rceil$ yields the subproblem IP^{i_2}.

- Let S^t be the feasible region of node t in the enumeration tree. If $\min\{c^T x \mid x \in S_R^t\} < \overline{z_{IP}}$ holds, the node t cannot be pruned.

- $\Rightarrow$ Bounds, i. e. the quality of the relaxations, are the primary factor for the efficiency of a branch-and-bound algorithm. This

$L = \{IP\}$; $S^0 = S$; $\underline{z}^0 = -\infty$; $\overline{z_{IP}} = \infty$;

while $(L \neq \emptyset)$ {

/ Node selection and relaxation */*

Choose a problem $IP^i \in L$ and remove it from L;

Solve the relaxation RP^i;

Let z^i_R be the optimal objective function value of the relaxation and let x^i_R be an optimal solution.

/ Pruning */*

if $(z^i_R \geq \overline{z_{IP}})$ {

Start the next loop iteration. If the relaxation is solved by using a dual algorithm, this can be done as soon as the dual objective function value reaches or exceeds $\overline{z_{IP}}$.

}

if $(x^i_R \in S^i)$ {

if $(c^T x^i_R < \overline{z_{IP}})$ {

Set $\overline{z_{IP}} = c^T x^i_R$;

$x^0 = x^i_R$;

}

Remove from L all problems with $\underline{z}^i \geq \overline{z_{IP}}$;

if $(c^T x^i_R = \overline{z_{IP}})$ {

Start the next loop iteration.

}

}

/ Partitioning */*

Let $\{S^{ij}\}_{j=1}^k$ be a partition of S^i.

Add to L the problems $\{IP^{ij}\}_{j=1}^k$ where $\underline{z}^{ij} = z^i_R$ for $j = 1, \ldots, k$. For each of the new subproblems the objective function value of their parent's relaxation, i.e. of the current relaxation, is used as lower bound.

}

if $(\overline{z_{IP}} < \infty)$ {

The solution x^0, whose objective function value is $\overline{z_{IP}} = c^T x^0$ is optimal.

$x^* = x^0$; $z^* = \overline{z_{IP}}$;

else {

There is no feasible solution: $S = \emptyset$;

}

is the reason why it is essential that the feasible region of the relaxation be close to the feasible region of the integral problem.

- Variable selection: no general robuts methods established yet. Common practice are user-defined *branching priorities*. Example: a binary variable indicating whether a project is to be started or not should be assigned a higher priority than a variable corresponding to detailed decisions within the project.

Node Selection

- Node selection: Input: list L of the active subproblems, i. e. the set of nodes of the decision tree that have not been pruned yet.
- Question: Which node to examine next?
- *A priori rules* determine the development of the tree in advance. Examples: *depth first search with backtracking* and *breadth first search*.
- *Adaptive rules* use information about the status of the active nodes. Examples: *Best bound* and *best estimate*.
- Best bound: choose the node with the smallest lower bound, i. e. whose relaxation has the lowest objective function value.
- Best estimate: choose the node that has the largest probability of representing an optimal solution. If $\hat{z}^i \geq \underline{z}^i$ is an estimate for z^i_{IP}, then that $i \in L$ is selected that minimises $\hat{z}^i$.

Branch And Cut

- The *branch-and-cut* algorithm is a synthesis of the branch-and-bound algorithm and cutting-plane methods.

- *Cut*: valid inequality of the original integer problem *IP* that is not part of the current formulation and that is not satisfied by all feasible points of the current LP relaxation.

- If a solution of the LP relaxation of *IP* does not satisfy all of the constraints the *separation problem* is to find a violated inequality. Since the convex hull of the feasible points is a polyhedron such an inequality must exist.

- Branch-and-cut: At each node of the enumeration tree that is not pruned a separation problem is solved. If one or more violated inequalities are found they are added to the formulation and the LP is solved again. If none are found the algorithm branches.

The OASIC Model

- $asap(i)$ denotes the as soon as possible, $alap(i)$ the as late as possible control step for the starting time of operation i.
- $N(i) = \{asap(i), \ldots, alap(i)\}$ is a superset of all control steps in which i can be started in any feasible schedule.
- $t_i \in N(i)$ denotes the starting time for the execution of an operation i.
- $Q_i^k \in \mathbb{N}$ represents the amount of time required to execute operation i by an instance of functional unit type k.
- $L_j^k \in \mathbb{N}$ represents the latency of the functional unit type k executing operation j, i. e. the minimal time interval between two successive data inputs to the same instance of k.
- The number of available instances of a functional unit type k is denoted R_k.
- E_R contains an edge (i, k) if operation i can be executed by by resource type k.
- $t_j = \sum_{k:(j,k)\in E_R} \sum_{n\in N(j)} n \cdot x_{jn}^k$
- Objective function

$$\min \ M_{steps}$$

- Constraints

  1. Time Constraints

The maximal number of control steps M_{steps}, i. e. the value of the objective function is defined as the starting time of the last operation to be executed.

$$t_j \le M_{steps} \qquad \forall\ j \in N_I$$

2. Precedence Constraints

– True Dependences:

$$\sum_{\substack{k:(i,k)\in E_R:\\ Q_i^k \ge n_j - n_i + 1}} x_{in_i}^k + \sum_{k:(j,k)\in E_R} x_{jn_j}^k \le 1 \qquad \forall\ (i,j,r,t) \in E_D^t$$

$$\forall n_i \in N(i) \quad \forall \quad n_j \in N(j), n_j \le n_i + \max_k Q_i^k - 1$$

– Output Dependences: Let $W_i = \{Q_i^k \mid (i,k) \in E_R\}$ for each operation i; then the constraints are defined as follows:

$$\sum_{\substack{k:(i,k)\in E_R:\\ Q_i^k = w_i}} x_{in_i}^k + \sum_{\substack{k:(j,k)\in E_R:\\ Q_j^k = w_j}} x_{jn_j}^k \le 1 \qquad \forall\ (i,j,r,o) \in E_D^o$$

$$\forall\ \ n_i \in N(i)\ \forall\ n_j \in N(j), n_j \le n_i + w_i - w_j,$$
$$\forall\ \ w_i \in W_i\ \forall\ w_j \in W_j$$

– Anti Dependences:

$$\sum_{k:(i,k)\in E_R} x_{in_i}^k + \sum_{\substack{k:(j,k)\in E_R\\ n_j + Q_j^k \le n_i}} x_{jn_j}^k \le 1 \qquad \forall\ (i,j,r,a) \in E_D^a$$

$$\forall\ \ n_i \in N(i)\ \forall\ n_j \in N(j),\ n_j \le n_i - \min_k Q_j^k$$

3. Execution Constraints

The execution of each operation must start in exactly one control step and is performed by exactly one resource instance.

$$\sum_{\substack{k \in N_R^F: \\ (j,k) \in E_R}} \sum_{n \in N(j)} x_{jn}^k \; = \; 1 \qquad \forall \, j \in N_I$$

4. Resource Constraints

   In no control step there must be more than R_k operations being executed by resource type k, i. e. the number of available instances of each functional unit type must not be exceeded.

$$\sum_{\substack{i \in N_I: \\ (i,k) \in E_R}} \sum_{\substack{n_i = n \\ n_i \in N(i)}}^{n + L_i^k - 1} x_{in_i}^k \leq R_k \qquad \forall \, k \in N_R^F \; \wedge \; 0 \leq n \leq M_{steps}$$

Improving the Precedence Constraints

- Tighter formulations allow more efficient computations.
- Goal: Transform the assignment and precedence polytope into a node packing graph, for which some integral facets are known.
- Change the precedence constraint formulation to take these facets into account.
- Let $G_N = (V_N, E_N)$ be an undirected graph and c a cost vector. Then the *weighted node packing problem* is defined as follows:

$$\begin{array}{rrcll} max & \sum_u c_u x_u & & & \\ & x_u + x_v & \leq & 1 & \forall\, (u,v) \in E_N \\ & x_u & \in & \{0,1\} & \forall\, u \in V_N \end{array}$$

- The constraints have the form $Ax \leq b$, where A is a $(0,1)$-matrix with each row containing two ones and b being a 1-vector.
- Each variable x_{in}^k is mapped to a node $u \in V_N$.
- The edges E_N of the graph are defined as follows: for each row of A (each constraint), the columns (variables) with coefficient 1 define a complete subgraph (clique) in G.
- A clique C of a graph G is a graph $C = (V_C, E_C)$, where for all $u, v \in V_C$: $(u,v) \in E_C$.
- An *odd cycle* C is a cycle consisting of an odd number of nodes.
- It is called *chordless* if no two nodes of C share an edge that does not belong to the cycle.

- Let $\mathcal{K}$ be the set of all cliques of G and $\mathcal{C}$ be the set of all chordless odd cycles in G.
- The following inequalities represent facets of the node packing polytope:

$$\sum_{u \in K} x_u \leq 1 \qquad \forall\, K \in \mathcal{K} \tag{6}$$

$$\sum_{u \in C} x_u \leq \frac{1}{2}(|C| - 1) \qquad \forall\, C \in \mathcal{C} \text{ where } |C| \geq 5 \tag{7}$$

- Example: Let two instructions i and j be given with execution time $w_i = w_j = 1$, and a dependence $i \prec j$. Assume $N(i) = \{1, \ldots, 4\}; N(j) = \{2, \ldots, 5\}$.

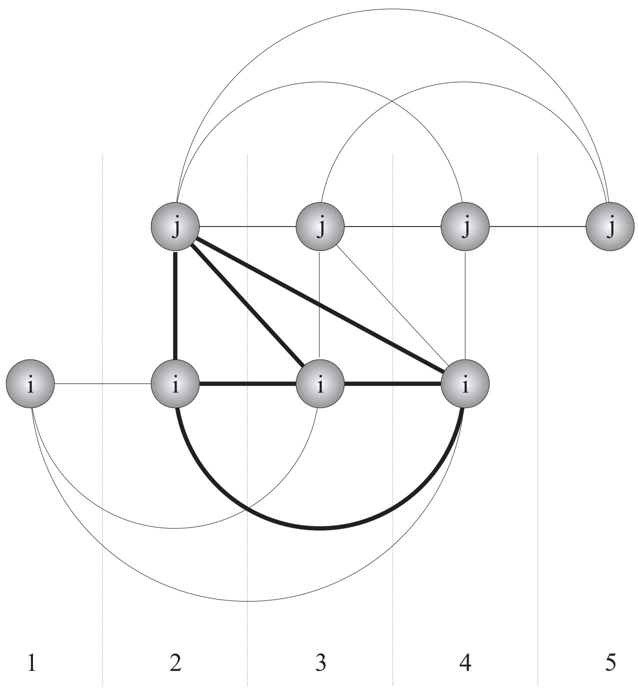

Figure 5: Node Packing.

- The deges of the graph correspond to the precedence inequalities, i.e. all infeasible orderings of i and j are represented by an edge in the graph. A clique facet for control step 2 is represented by the bold edges.

- This yields the following tighter formulation of the precedence constraints:

$$\sum_{k \in R_j(w_j)} \sum_{\substack{n_j \le n \\ n_j \in N(j)}} x^k_{jn_j} + \sum_{k \in R_i(w_i)} \sum_{\substack{n_i \ge n - w_i + 1 \\ n_i \in N(i)}} x^k_{in_i} \;\le\; 1$$
$$\forall\ (i, j, r, t) \in E^t_D, w_i \in W_i, w_j \in W_j$$
$$n \in N^t(i, j) = \{n' + w_i - 1 | n' \in N(i)\} \quad \cap \quad N(j)$$

Here $W_i = \{Q^k_i \mid i \text{ can be executed on } k\}$. The constraints for anti and output dependences are formulated analogeously.