6 $\mathcal{NP}$-Completeness

Initiated in large measure by the seminal papers of S. A. Cook (1971) and R. M. Karp (1972) in the area of discrete optimization.

6.1 General

Definition. An *instance* of an optimization problem consists of a feasible set F and a cost function $c : F \to \Re$. An optimization *problem* is defined as a collection of instances.

For example, "linear programming" is a problem and $\min\{x : x \geq 3\}$ is an "instance".

Definition. The **size** of an instance is defined as the number of bits used to describe the instance, according to a prescribed format.

Definition. An algorithm runs in *polynomial time* if there exists an integer k such that $T(n) = O(n^k)$.

The class $\mathcal{P}$: A combinatorial optimization (CO) problem is in $\mathcal{P}$ if it admits algorithms of polynomial complexity.

The class $\mathcal{NP}$: A combinatorial problem is in $\mathcal{NP}$ if for all"YES" instances, there exists a polynomial length **certificate** that can be used to verify in polynomial time that the answer is indeed yes.

$\mathcal{NP}$: e.g., verify the optimality of a LP solution.

Obviously, $\mathcal{P} \subseteq \mathcal{NP}$. But,

$$\mathcal{P} = \mathcal{NP}?$$

Definition: Suppose that there exists an algorithm for some problem A that consists of a polynomial time computation in addition of polynomial number of subroutine calls to an algorithm for problem B. We then say that problem A **reduces** (in polynomial time) to problem B. For short, $A \overset{\text{R}}{\Longrightarrow} B$

Theorem: If $A \overset{\text{R}}{\Longrightarrow} B$ and $B \in \mathcal{P}$, then $A \in \mathcal{P}$.

The above theorem says that if $A \overset{\text{R}}{\Longrightarrow} B$, then problem A is not much more difficult than problem B.

6.2 Three forms of a CO problem

A CO problem: F is the feasible solution set and $c : F \to \Re$ is a cost function,

$$\begin{array}{ll} \min & c(f) \\ \text{s.t.} & f \in F. \end{array}$$

Consider the following combinatorial optimization problem, called the *maximum clique problem*:

Given a graph $G = (V, E)$ find the largest subset $C \subseteq V$ such that for all distinct $u, v \in C$, $(v, u) \in E$.

The maximum clique problem is in $\mathcal{NP}$ or in short, Clique $\in \mathcal{NP}$.

a) Optimization version: Find the optimal solution.

b) The evaluation version: Find the optimal value of $c(f)$, $f \in F$.

c) The recognitive version: Given an integer L, is there a feasible solution $f \in F$ such that $c(f) \leq L$?.

Let us consider the maximum clique problem and assume that we have a procedure *cliquesize* which, given any graph G, will evaluate the size of the maximum clique of G. In other words cliquesize solves the evaluation version of the maximum clique problem. We can then make efficient use of this routine in order to solve the optimization version.

Step 0 . $X = \emptyset$.

Step 1. Find $v \in V$ such that cliquesize$(G(v))$ = cliquesize(G), where $G(v)$ is the subgraph of G consisting of v and all its adjacent nodes.

Step 2. $X = X + v$. $G = G(v)\backslash v$. If $G = \emptyset$, stop; otherwise, go to Step 1.

6.3 $\mathcal{NPC}$

The class co-$\mathcal{NP}$: A combinatorial problem is in co-$\mathcal{NP}$ if for all "NO" instances, there exists a polynomial length **certificate** that can be used to verify in polynomial time that the answer is indeed no.

Obviously, $\mathcal{P} \subseteq$ co$-\mathcal{NP}$. But,

$$\mathcal{P} = \text{co}-\mathcal{NP}?$$

The class $\mathcal{NP}$-hard: A problem A is $\mathcal{NP}-$hard if for any problem $B \in \mathcal{NP}$, $B \overset{\text{R}}{\Longrightarrow} A$.

Define a set of Boolean variables $\{x_1, x_2, \ldots, x_n\}$ and let the complement of any of these variables x_i be denoted by $\bar{x}_i$. In the language of logic, these variables are referred to as *literals*. To each literal we assign a label of *true* or *false* such that x_i is *true* if and only if $\bar{x}_i$ is *false*.

Let the symbol $\vee$ denote *or* and the symbol $\wedge$ denote *and*. We then can write any Boolean expression in which is referred to as *conjunctive normal form*, i.e., as a finite

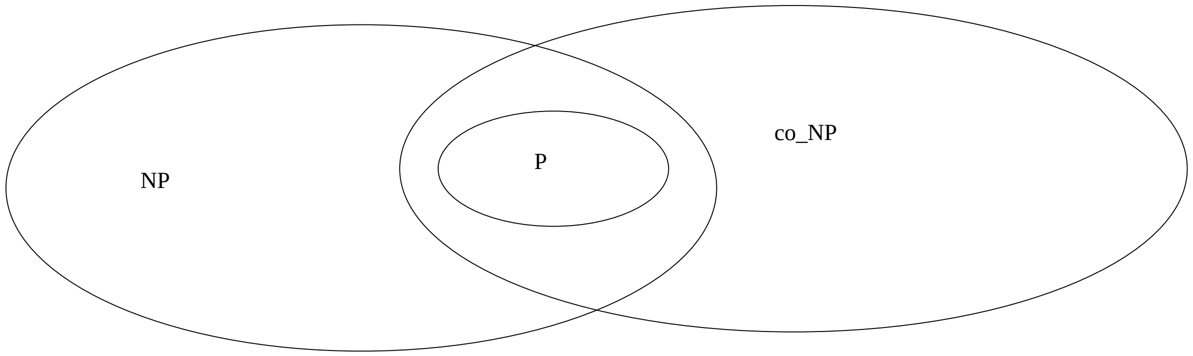

Figure 1: Relationships among $\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$

conjunction of disjunctions using each literal once at most. For example, with the set of variables $\{x_1, x_2, x_3, x_4\}$ one might encounter the following conjunctive normal form expression

$$(x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_4).$$

Each disjunctive grouping in parenthesis is referred to as a *clause*. The *satisfiability problem* is

Given a set of literals and a conjunction of clauses defined over the literals, is there an assignment of values to the literals for which the Boolean expression is *true*?

If so, then the expression is said to be satisfiable. The Boolean expression above is satisfiable via the following assignment: $x_1 = x_2 = x_3 = true$ and $x_4 = false$. Let SAT denote the satisfiability problem and Q be any member of $\mathcal{NP}$.

Theorem. (Cook (1971)) Every problem $Q \in \mathcal{NP}$ polynomially reduces to SAT.

Karp (1972) showed that SAT polynomially reduces to many combinatorial problems.

The class $\mathcal{NPC}$: A recognition problem A is $\in \mathcal{NPC}$ if i) $A \in \mathcal{NP}$ and ii) for any problem $B \in \mathcal{NP}$, $B \overset{\text{R}}{\Longrightarrow} A$.

Cook's Theorem shows SAT $\in \mathcal{NPC}$ because SAT $\in \mathcal{NP}$.

Examples of $\mathcal{NPC}$ problems: ILP, Clique, Vertex Packing, TSP, 3-Index Assignment, Knapsack, etc.

For a good guide to the theory of $\mathcal{NPC}$, see

1979, M. R. Garey and D. S. Johnson, "Computers and Intractability: a Guide to the Theory of $\mathcal{NP}$-Completeness".

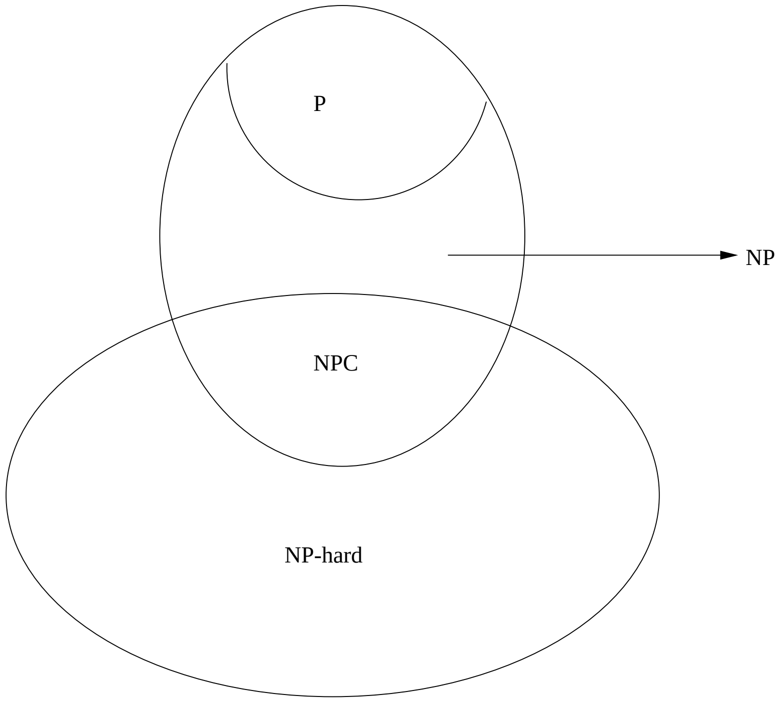

Figure 2: Relationships among $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NPC}$, and $\mathcal{NP}$-hard

6.4 Algorithms for $\mathcal{NPC}$

1. Exact methods: Branch and bound, cutting plane methods, branch and cut, dynamic programming method, etc.

2. Special classes: Plane graphs, perfect graphs.

3. Local search.

4. Heuristics.

5. Approximation methods: Semidefinite programming (SDP) approach.

6. Polyhedral methods.

7. Probabilistic methods.

6.5 Cutting plane methods

1. Simplex method revisited

Consider the standard linear programming problem

$$(P) \qquad \begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b, \\ & x \geq 0, \end{array} \tag{6.1}$$

where $A \in \Re^{m\times n}$ $(m \leq n)$ is of full row rank, and its dual form

$$(D) \qquad \begin{array}{ll} \max & b^T y \\ \text{s.t.} & A^T y \leq c. \end{array} \tag{6.2}$$

Let x be a basic feasible solution to the standard form problem, let $B(1), \ldots, B(m)$ be the indices of the basic variables, and let $B = [A_{B(1)} \ldots A_{B(m)}]$ the corresponding basis matrix. In particular, we have $x_i = 0$ for every nonbasic variable, while the vector $x_B = (x_{B(1)}, \ldots, x_{B(m)})^T$ of basic variables is given by

$$x_B = B^{-1}b \geq 0.$$

The full tableau implementation of the simplex method at the very beginning is:

0	c_1	$\ldots$	c_n
b_1	$\vert$		$\vert$
$\vdots$	A_1	$\ldots$	A_n
b_m	$\vert$		$\vert$

Let $\bar{c}^T = c^T - c_B^T B^{-1} A$. By using elementary row operations to change $c_{B(1)}, \ldots, c_{B(m)}$ to be zero, we have the following tableau:

$-c_B^T x_B$	$\bar{c}_1$	$\ldots$	$\bar{c}_n$
$x_{B(1)}$	$\vert$		$\vert$
$\vdots$	$B^{-1}A_1$	$\ldots$	$B^{-1}A_n$
$x_{B(m)}$	$\vert$		$\vert$

Anticycling rules: Lexicography and Bland's rule.

Finding an initial basic feasible solution: The artificial variables method and the big$-M$ method.

For the **dual simplex method**, we have

0	c_1	$\ldots$	c_n
b_1	$\vert$		$\vert$
$\vdots$	A_1	$\ldots$	A_n
b_m	$\vert$		$\vert$

and

$$\begin{array}{|c|ccc|}
\hline
-c_B^T x_B & \bar{c}_1 & \dots & \bar{c}_n \\
\hline
x_{B(1)} & | & & | \\
\vdots & B^{-1}A_1 & \dots & B^{-1}A_n \\
x_{B(m)} & | & & | \\
\hline
\end{array}$$

We do not require $B^{-1}b$ to be nonnegative, which means that we have a basic, but not necessarily feasible solution to the primal problem. However, we assume that $\bar{c} \geq 0$; equivalently, the vector $y^T = c_B^T B^{-1}$ satisfies $y^T A \leq c^T$, and we have a feasible solution to the dual problem. The cost of this dual feasible solution is $y^T b = c_B^T B^{-1} b = c_B^T x_B$, which is the negative of the entry at the upper left corner of the tableau.

2. A generic cutting plan algorithm

We consider the integer linear programming (ILP) problem

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b, \\ & x \geq 0, \ \text{integer} \end{array} \tag{6.3}$$

and its linear programming (LP) relaxation

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b, \\ & x \geq 0. \end{array} \tag{6.4}$$

The main idea of cutting plane methods is to solve the integer linear programming problem (6.3) by solving a sequence of linear programming problems, as follows.

A generic cutting plane algorithm

step 1. Solve the linear programming relaxation (6.4). Let x^* be an optimal solution.

Step 2. If x^* is integer stop; x^* is an optimal solution to (6.3).

Step 3. If not, add a linear inequality constraint to (6.4) that all integer solutions to (6.3) satisfy, but x^* does not; go to Step 1.

3. The Gomory cutting plane algorithm

Let x^* be an optimal basic feasible solution to (6.4) and let B be an associated optimal basis. We partition x into a subvector x_B of basic variables and a subvector x_N of nonbasic variables. Assume for simplicity that the first m variables are basic, so that $x_{B(i)} = x_i$, for $i = 1, \cdots, m$. Recall that a tableau provides us with the coefficients of the equation $B^{-1}Ax = B^{-1}b$. Let N be the set of nonbasic variables. Let A_N be the submatrix of

A with columns A_i, $i \in N$. From the optimal tableau, we obtain the coefficients of the constraints

$$x_B + B^{-1}A_N x_N = B^{-1}b.$$

Let $\bar{a}_{ij} = (B^{-1}A_j)_i$ and $\bar{a}_{i0} = (B^{-1}b)_i$. We consider one equality from the optimal tableau, in which $\bar{a}_{i0}$ is fractional:

$$x_i + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{a}_{i0}. \tag{6.5}$$

Since $x_j \geq 0$ for all j, we have

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq x_i + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{a}_{i0}.$$

Since x_j should be integer, we obtain

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{a}_{i0} \rfloor. \tag{6.6}$$

The above inequality is valid for all integer solutions, but it is not satisfied by x^*. The reason is that $x_i^* = \bar{a}_{i0}$, $x_j^* = 0$ for all nonbasic $j \in N$, and $\lfloor \bar{a}_{i0} \rfloor < \bar{a}_{i0}$ since $\bar{a}_{i0}$ is assumed fractional.

By combining (6.5) and (6.6), we have

$$\sum_{j \in N} (\lfloor \bar{a}_{ij} \rfloor - \bar{a}_{ij}) x_j \leq \lfloor \bar{a}_{i0} \rfloor - \bar{a}_{i0}. \tag{6.7}$$

Note that only nonbasic variables appear in (6.7), which is particularly useful in using the dual simplex method for the augmented linear programming relaxation.

It has been shown that by systematically adding these cuts and using the dual simplex method with appropriate anticycling rules, we obtain a finitely terminating algorithm for solving general integer linear programming problems.

Example 6.1 (Illustration of the Gomory cutting plane algorithm)

Consider the integer linear programming problem

$$\begin{array}{ll} \min & (-x_1 - x_2) \\ \text{s.t.} & 7x_1 + x_2 \leq 15 \\ & -x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0, \text{ integer.} \end{array}$$

We transform this problem in standard form

$$\begin{array}{ll} \min & (-x_1 - x_2) \\ \text{s.t.} & 7x_1 + x_2 + x_3 \qquad = 15 \\ & -x_1 + x_2 \qquad + x_4 = 1 \\ & x_1, x_2, x_3, x_4 \geq 0, \text{ integer.} \end{array}$$

We solve the linear programming relaxation by the simplex method.

		x_1	x_2	x_3	x_4
	0	-1	-1	0	0
$x_3 =$	15	7^*	1	1	0
$x_4 =$	1	-1	1	0	1

The final tableau is

		x_1	x_2	x_3	x_4
	9/2	0	0	1/4	3/4
$x_1 =$	7/4	1	0	1/8	−1/8
$x_2 =$	11/4	0	1	1/8	7/8

The optimal solution (in terms of the original variables) is $x^1 = (7/4, 11/4)^T$. One of the equations in the optimal tableau is

$$x_1 + \frac{1}{8}x_3 - \frac{1}{8}x_4 = \frac{7}{4}.$$

We apply the Gomory cutting plane algorithm, and find the cut

$$x_1 - x_4 \leq 1.$$

Using $-x_1 + x_2 + x_4 = 1$, we find that $x_1 - x_4 \leq 1$ is equivalent to:

$$x_2 \leq 2.$$

We can augment the linear programming relaxation by adding the constraint $x_2 \leq 2$. However, actually we do not add $x_2 \leq 2$ to the original ILP problem, but its equivalent in terms of the current **nonbasic variables** x_3 and x_4; namely:

$$-\frac{1}{8}x_3 - \frac{7}{8}x_4 \leq -\frac{3}{4}.$$

There is a simple way to obtain this constraint directly from the source constraint. To that end the source-constraint is written as:

$$x_1 + (0 + \frac{1}{8})x_3 + (-1 + \frac{7}{8})x_4 = 1 + \frac{3}{4},$$

so that

$$-\frac{1}{8}x_3 - \frac{7}{8}x_4 \leq -\frac{3}{4}.$$

Introducing the slack variable x_5 for this cut constraint, the new linear programming relaxation becomes

$$\begin{aligned}
\min \quad & (-\frac{9}{2} + \frac{1}{4}x_3 + \frac{3}{4}x_4) \\
\text{s.t.} \quad & x_1 + \frac{1}{8}x_3 - \frac{1}{8}x_4 = \frac{7}{4} \\
& x_2 + \frac{1}{8}x_3 + \frac{7}{8}x_4 = \frac{11}{4} \\
& -\frac{1}{8}x_3 - \frac{7}{8}x_4 + x_5 = -\frac{3}{4} \\
& x_1, x_2, x_3, x_4, x_5 \geq 0.
\end{aligned}$$

The basic solution $x_1 = \frac{7}{4}$, $x_2 = \frac{11}{4}$, $x_5 = -\frac{3}{4}$ is not feasible. On the other hand all objective coefficients are nonnegative, so that we may apply the **dual simplex method**.

		x_1	x_2	x_3	x_4	x_5
	$9/2$	0	0	$1/4$	$3/4$	0
$x_1 =$	$7/4$	1	0	$1/8$	$-1/8$	0
$x_2 =$	$11/4$	0	1	$1/8$	$7/8$	0
$x_5 =$	$-3/4$	0	0	$-1/8$	$-7/8^*$	1

The final tableau is

		x_1	x_2	x_3	x_4	x_5
	$27/7$	0	0	$1/7$	0	$6/7$
$x_1 =$	$13/7$	1	0	$1/7$	0	$-1/7$
$x_2 =$	2	0	1	0	0	1
$x_4 =$	$6/7$	0	0	$1/7$	1	$-8/7$

The optimal solution (in terms of the original variables) is $x^2 = (13/7, 2)^T$. One of the equations in the optimal tableau is

$$x_1 + \frac{1}{7}x_3 - \frac{1}{7}x_5 = \frac{13}{7}.$$

The corresponding cut constraint in terms of the nonbasic variables x_3 and x_5 is

$$-\frac{1}{7}x_3 - \frac{6}{7}x_5 \leq -\frac{6}{7}.$$

One can easily check that this cut constraint is equivalent to

$$x_1 - x_5 \leq 1,$$

which is equivalent to

$$x_1 + x_2 \leq 3.$$

After introducing the slack variable x_6 for this cut constraint, the new linear programming relaxation becomes

$$\begin{array}{lllllll}
\min & (-\frac{27}{7} + \frac{1}{7}x_3 + \frac{6}{7}x_5) & & & & & \\
\text{s.t.} & x_1 & & + \frac{1}{7}x_3 & - \frac{1}{7}x_5 & & = \frac{13}{7} \\
 & & x_2 & & + x_5 & & = 2 \\
 & & & \frac{1}{7}x_3 + x_4 & - \frac{8}{7}x_5 & & = \frac{6}{7} \\
 & & & -\frac{1}{7}x_3 & - \frac{6}{7}x_5 & + x_6 & = -\frac{6}{7} \\
 & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0. & & & & &
\end{array}$$

Again, we may apply the **dual simplex method**. One can easily check that the optimal solution of the new linear programming relaxation is $x_1 = 1, x_2 = 2$ ($x_3 = 6, x_4 = x_5 = x_6 = 0$). Thus $x^3 = (1,2)^T$ is integer and in fact an optimal solution of the original ILP problem. □.

6.6 Branch and bound

1. Branch and bound uses a "divide and conquer" approach to explore the set of feasible integer solutions. However, instead of exploring the entire feasible set, it uses bounds on optimal cost to avoid exploring certain parts of the set of feasible integer solutions.

Let F be the set of feasible solutions to the problem

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & x \in F. \end{array}$$

We partition the set F into a finite collection of subsets $F_1, \ldots, F_k$, and solve separately each one of the subproblems

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & x \in F_i, \quad i = 1, \ldots, k. \end{array}$$

We then compare the optimal solutions to the subproblems, and choose the best one. Each subproblem may be almost as difficult as the original problem and this suggests trying to solve each subproblem by means of the same method; that is, by splitting it into further subproblems, etc. This is the **branching part of the method** and leads to a tree of subproblems; see Figure 3.

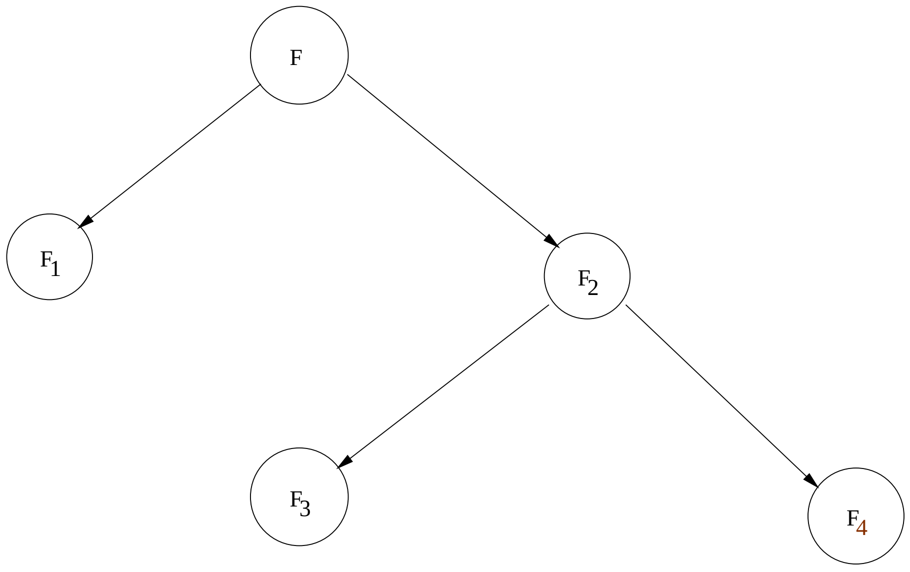

Figure 3: A tree of subproblems

We also assume that there is a fairly efficient algorithm which can compute a *lower bound* $b(F_i)$ to the optimal cost of the corresponding subproblem; that is

$$b(F_i) \le \min_{x \in F_i} c^T x.$$

The basic idea is that while the optimal cost in a subproblem may be difficult to compute exactly, a lower bound might be easier to obtain. A popular method is to use the optimal cost of the linear programming relaxation.

In the course of the algorithm, we will also occasionally solve certain subproblems to optimality, or simply evaluate the cost of certain feasible solutions. This allows us to maintain an upper bound U on the optimal cost, which could be the cost of the best feasible solution encountered thus far. If the lower bound $b(F_i)$ corresponding to a particular subproblem satisfies $b(F_i) \geq U$, then this subproblem need not be considered further, since the optimal solution to the subproblem is no better than the best feasible solution encountered thus far.

A generic branch and bound algorithm

Step 0. Initially, U is set either to ∞ or to the cost of some feasible solution, if one happens to be available.

Step 1. Select an active subproblem F_i.

Step 2. If the subproblem is infeasible, delete it; otherwise, compute $b(F_i)$ for the corresponding subproblem.

Step 3. If $b(F_i) \geq U$, delete the subproblem.

Step 4. If $b(F_i) < U$, either obtain an optimal solution to the subproblem, or break the corresponding subproblem into further subproblems, which are added to the list of active subproblems.

For LP based branch and bound algorithm:

• Compute the lower bound by solving the LP relaxation.

• Form the LP solution x^*, if there is a component x_i^* which is fractional, one can create two subproblems by adding either one of the constraints

$$x_i \leq \lfloor x_i^* \rfloor, \quad \text{or } x_i \geq \lceil x_i^* \rceil.$$

Note that both constraints are violated by x^*.

• Select the active subproblem using either depth-first or breadth-first search strategies.

Example 6.2 (Illustration of the branch and bound algorithm)

We solve the ILP of Example 6.1 by the branch and bound algorithm. Initially, $U = \infty$. We solve the linear programming relaxation and the optimal solution $x^1 = (7/4, 11/4)^T$. Then, $b(F)$ is the optimal cost of the relaxation, i.e., $b(F) = -9/2$. We create two subproblems, by adding the constraints $x_2 \geq 3$ (the subproblem F_1), or $x_2 \leq 2$ (subproblem F_2). The active list of subproblems is $\{F_1, F_2\}$. The linear programming relaxation of subproblem F_1 is infeasible (in this case let $b(F_1) = \infty$) and therefore, we can delete this subproblem from the active list. Next we consider the optimal solution to the linear programming relaxation of subproblem F_2 (LPF_2). As in the Gomory cutting plane method, instead of adding $x_2 \leq 2$ directly to the LPF_2, we add its equivalent in terms of the current nonbasic variables x_3 and x_4; namely:

$$-\frac{1}{8}x_3 - \frac{7}{8}x_4 \le -\frac{3}{4}.$$

By using the **dual simplex method**, we find that $x^2 = (13/7, 2)^T$ is an optimal solution to the LPF_2, and thus $b(F_2) = -27/7$. We further decompose subproblem F_2 into two subproblems, by adding either $x_1 \ge 2$ (subproblem F_3), or $x_1 \le 1$ (subproblem F_4). The active list of subproblems is now $\{F_3, F_4\}$. The linear relaxation of subproblem F_3 is infeasible and thus $b(F_3) = \infty$. Therefore, we can delete this subproblem from the active list.

Next we consider the optimal solution to the linear programming relaxation of subproblem F_4 (LPF_4). As in the Gomory cutting plane method, instead of adding $x_1 \le 1$ directly to the LPF_4, we add its equivalent in terms of the current nonbasic variables x_3 and x_5; namely:

$$-\frac{1}{7}x_3 - \frac{1}{8}x_4 \le -\frac{6}{7}.$$

After introducing the slack variable x_6 for this constraint, the linear programming relaxation of the subproblem F_4 becomes

$$\begin{array}{lllllll}
\min & (-\frac{27}{7} + \frac{1}{7}x_3 + \frac{6}{7}x_5) & & & & & \\
\text{s.t.} & x_1 & & +\frac{1}{7}x_3 & -\frac{1}{7}x_5 & & = \frac{13}{7} \\
 & & x_2 & & +x_5 & & = 2 \\
 & & & \frac{1}{7}x_3 + x_4 & -\frac{8}{7}x_5 & & = \frac{6}{7} \\
 & & & -\frac{1}{7}x_3 & -\frac{1}{8}x_5 + x_6 & & = -\frac{6}{7} \\
 & x_1, x_2, x_3, x_4, x_5, x_6 \ge 0. & & & & &
\end{array}$$

By applying the dual simplex method, one can easily check the optimal solution to the above linear programming relaxation is $x_1 = 1, x_2 = 2$ $(x_3 = 6, x_4 = x_5 = x_6 = 0)$. Thus $x^3 = (1, 2)^T$ is an integer and $U = -3$. Since the list of subproblems is empty, we terminate. The optimal solution is $x^3 = (1, 2)^T$.

2. Branch and cut: utilize cuts when solving subproblems. If we can add "deep" cuts, we can accelerate branch and bound considerably. However, finding such cuts are nontrivial.

3. Mixed integer linear programming (MLIP) problems.

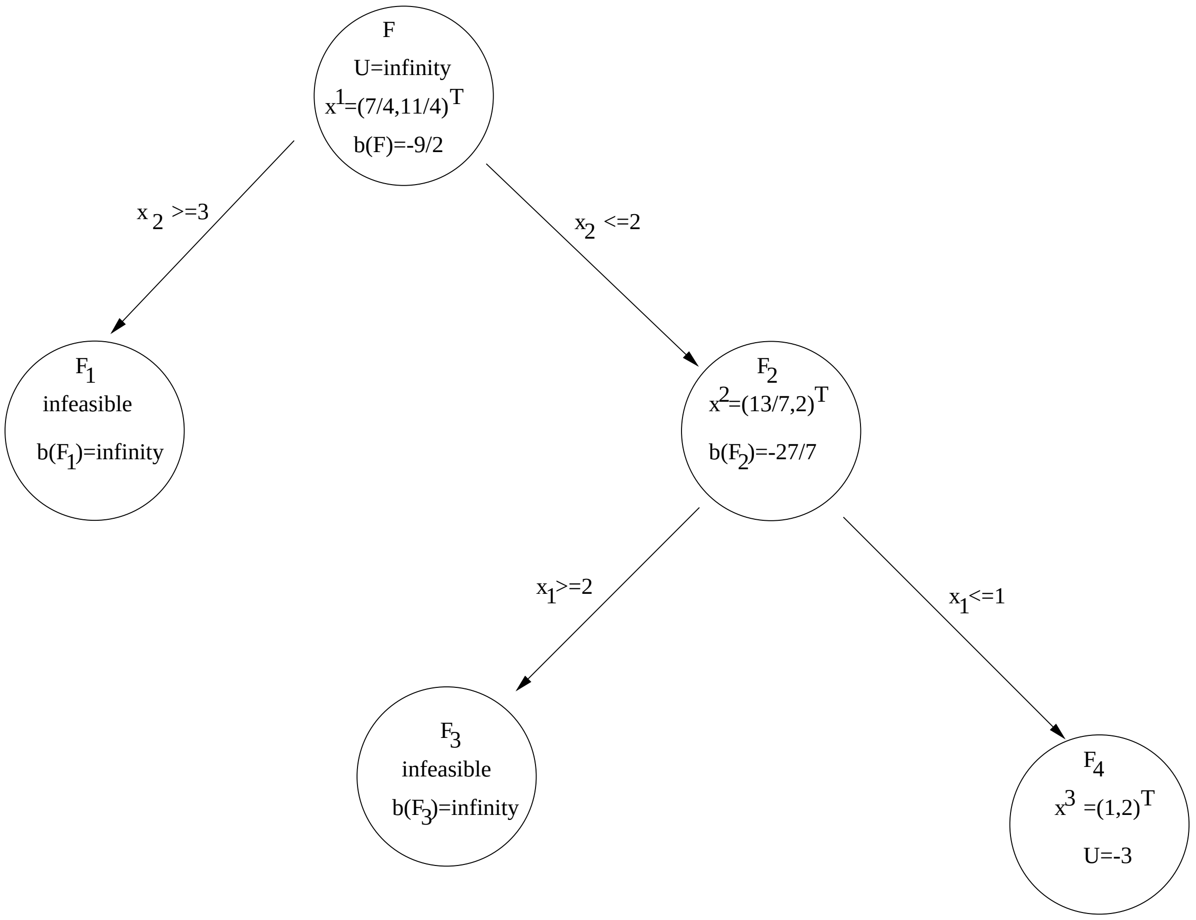

Figure 4: Illustration of the branch and bound algorithm for Example 6.2.

Consider

$$\begin{array}{ll} \min & (c^T x + d^T y) \\ \text{s.t.} & A_1 x + A_2 y \le b, \\ & x \ge 0, \\ & y \ge 0, \ \text{integer}. \end{array}$$

4. Application to a flowshop scheduling problem.

We are given a set of n jobs, J_i, $i = 1, \cdots, n$. Each job has two tasks, each to be performed on one of two machines. Job J_i requires a processing time τ_{ij} on machine i, and each task must complete processing on machine 1 before starting on machine 2. Let t_{ji} be the time at which job i finishes on machine j. The *sum finishing time* is defined to be the sum of times that all the jobs finish processing on machine 2:

$$f = \sum_{i=1}^{n} c_{2i}.$$

The sum-finishing-time problem (SFTP) is the problem of determining the order in which to assign the tasks to machines so f is minimum.

5. Dynamic programming.

Dynamic programming is related to branch and bound in the sense that it performs an intelligent enumeration of all the feasible points of a problem, but it does so in a different way. The idea is to work backwards from the last decision to the earlier decisions.

Suppose we need to make a sequence of n decisions to solve a combinatorial optimization problem, say $D_1, D_2, \ldots, D_n$. Then if the sequence is optimal, the last k decisions, $D_{n-k+1}, D_{n-k+2}, \ldots, D_n$ must be optimal. That is, the completion of an optimal sequence of decisions must be optimal. This is referred to as the *principle of optimality* as we have already known.

6.7 Local search

Given F, c. Find the optimal (min) solution to the problem

$$\begin{array}{ll} \min & c(f) \\ \text{s.t.} & f \in F. \end{array}$$

For any $f \in F$, define its neighborhood $N(f) \subset F$.

Step 0. $f = f_0$.
Step 1.

$$c(f_{k+1}) = \begin{array}{ll} \min & c(f) \\ \text{s.t.} & f \in N(f_k). \end{array}$$

If $c(f_{k+1}) = c(f_k)$, stop; otherwise, repeat Step 1.

Drawback: Local minimum found.

6.8 ε-approximation algorithms

Given a CO problem, with $A = \{F, c\}$,

$$c(x^*) = \begin{array}{ll} \min & c(x) \\ \text{s.t.} & x \in F. \end{array}$$

An approximation algorithm finds $\bar{x}$ such that

$$\left| \frac{\bar{c}(\bar{x}) - c(x^*)}{c(x^*)} \right| \leq \varepsilon.$$

Then this algorithm is called an ε-approximation algorithm.

Traveling Salesman Problem (TSP).

Theorem. If $\mathcal{NP} \neq \mathcal{P}$, then TSP has no polynomial time $\varepsilon-$ approximation algorithm.

Symmetric TSP (ΔTSP):

$$\begin{aligned}
&i)\ d_{ii} = 0;\\
&ii)\ d_{ij} = d_{ji}.\\
&iii)\ \forall i, j, k,\ d_{ij} + d_{jk} \geq d_{ik}.
\end{aligned}$$

Theorem. ΔTSP $\in \mathcal{NPC}$.

Consider the following algorithm for the ΔTSP:

Step 0. $G = (V, E)$ and $D = (d_{ij})$.

Step 1. Find a minimum spanning tree T^*.

Step 2. Duplicate T^*. Find an Eulerian walk.

Step 3. Find an embedded TSP tour τ by skipping any intermediate node that has already been visited.

Theorem. The algorithm is a 1-approximation algorithm.
Proof. Suppose the optimal TSP tour is τ^*. Then

$$c(\tau^*) \geq c(T^*)$$

and

$$c(\tau^*) \leq c(\tau) \leq 2c(T^*).$$

Then,

$$0 \leq c(\tau) - c(\tau^*) \leq c(\tau^*).$$

□