the last row). We then compute $\xi_4 = 1/1 = 1$, $\xi_5 = 2/1 = 2$, $\xi_2 = 2/0 = \infty$, and $\xi_7 = 6/0 = \infty$. This identifies x_4 as the new leaving variable, as shown below.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
$\leftarrow x_4$	1	0	0	1	0	-1	0	1
x_5	1	0	0	0	1	0	0	2
x_2	0	1	0	0	0	1	0	2
x_7	0	0	1	0	0	0	1	6
	-5	0	-1	0	0	4	0	8
	$\uparrow$							

Since the (x_4, x_1)-th entry is already 1, we skip the x_4-th row by 1. We subtract this row from the row headed by x_5, and add 5-times this row to the objective row. This changes the tableau as follows. We replace the heading of the first row from x_4 to x_1.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
x_1	1	0	0	1	0	-1	0	1
x_5	0	0	0	-1	1	1	0	1
x_2	0	1	0	0	0	1	0	2
x_7	0	0	1	0	0	0	1	6
	0	0	-1	5	0	-1	0	13

The reader is urged to follow this procedure for the remaining two iterations.

It is clear that simplex tableaux do not introduce any new concept. They only provide a procedurally convenient way of carrying out the algebra introduced in Section 26.2.2.

26.3 Integer linear programming

The variables appearing in LP instances are non-negative real-valued. LP solvers compute real-valued optimal solutions. If we restrict the variables to take values from the set of non-negative integers, the optimization problem is called *integer linear programming* (*ILP*). In some cases, we restrict the variables to take only the values $0, 1$. In these cases, we talk about $0,1$-*valued linear programming* or *binary linear programming*. Unlike LP, ILP (or $0,1$-valued LP) does not have known polynomial-time algorithms. Indeed, no such algorithms can exist unless P = NP, because (decision versions of) ILP and $0,1$-valued LP are known to be NP-Complete.

Many optimization problems can be phrased as ILP problems. Solving these instances of ILP is, in general, infeasible. However, we can exploit the polynomial-time (real-valued) LP solvers to achieve graceful algorithms for solving ILP instances. In this section, we concentrate on the formulation of many common optimization problems as ILP problems. Solving these ILP instances will be dealt with in the next section.

It is worthwhile to note here that when converting an instance I of the original optimization problem to an instance J of an ILP problem, we desire to have the size $|J|$ to be a polynomial expression in the size $|I|$. Such an ILP formulation is useful in practice, because if $|J|$ is an exponential function of $|I|$, even a polynomial-time ILP solver does not lead to an efficient algorithm for solving the original optimization problem (on input I).

26.3.1 Knapsack problem

The knapsack problem has a very natural rendering as an ILP problem. There are n objects of weights $w_1, w_2, \ldots, w_n$ and costs $c_1, c_2, \ldots, c_n$. There is a knapsack of weight capacity B. The task is to pack objects in the knapsack such that the total weight of the packed objects is no larger than B, and the total cost of the packed objects is as large as possible. Let $x_1, x_2, \ldots, x_n$ be $0,1$-valued variables with the interpretation that

$$x_i = \begin{cases} 1 & \text{if the } i\text{-th object is packed in the knapsack,} \\ 0 & \text{otherwise.} \end{cases}$$

We then have to

$$\text{maximize} \quad \sum_{i=1}^{n} c_i x_i$$

subject to the constraint that

$$\sum_{i=1}^{n} w_i x_i \leqslant B.$$

26.3.2 Cover problems

Let $G = (V, E)$ be an undirected graph. A vertex cover of G is a subset $C \subseteq V$ such that every edge in E is incident upon at least one vertex of C. The MIN_VERTEX_COVER problem deals with the determination of a vertex cover C of G with the smallest possible size. In order to formulate this problem as an ILP problem, we introduce variables x_u for each $u \in V$. This variable has the following interpretation that

$$x_u = \begin{cases} 1 & \text{if } u \text{ is included in the cover } C, \\ 0 & \text{otherwise.} \end{cases}$$

Since we want to compute a smallest vertex cover, our goal is to

$$\text{minimize} \sum_{u \in V} x_u.$$

We must ensure that the set C to be constructed is indeed a vertex cover of G. Let $e = (u, v)$ be any edge in G. In order to insure that e is covered, we must have either $x_u = 1$ or $x_v = 1$ (or both). This can be written as the following linear constraint:

$$x_u + x_v \geqslant 1 \quad \text{for each } e = (u, v) \in E.$$

Now, suppose that each vertex u has a positive weight w_u. The WEIGHTED_VERTEX_COVER problem deals with the computation of a vertex cover C of G such that the weighted sum $\sum_{u \in C} w_u$ is as small as possible. The variables x_u, their interpretation, and the linear constraints remain the same as above. Only the objective function changes to

$$\text{minimize} \sum_{u \in V} w_u x_u.$$

The vertex cover problem is a special case of the set cover problem. Let X be a finite set of size n. We are given a family $S_1, S_2, \ldots, S_m$ of subsets of X with the property that $\cup_{i=1}^{m} S_i = X$. The MIN_SET_COVER problem deals with the computation of a smallest sub-collection C of the given family, which continues to cover X (that is, the union of the sets in C is X). We introduce a variable x_i for each S_i with the interpretation that

$$x_i = \begin{cases} 1 & \text{if } S_i \text{ is included in the cover } C, \\ 0 & \text{otherwise.} \end{cases}$$

The objective is to

$$\text{minimize} \sum_{i=1}^{m} x_i.$$

In order to insure that the sub-collection C covers X, take any element $a \in X$. Denote $\text{Idx}(a) = \{i \in \{1,2,\ldots,m\} \mid a \in S_i\}$. In order that a is covered by C, we require C to contain S_i for at least one $i \in \text{Idx}(a)$. Thus, we have the following constraints:

$$\sum_{i \in \text{Idx}(a)} x_i \geqslant 1 \quad \text{for each } a \in X.$$

It is easy to see that for all of the above three problems, the ILP instances produced are of size polynomial (in fact, linear) in the size of the instances of the original optimization problems.

26.3.3 Matching problem

Let $G = (V,E)$ be an undirected graph. Two edges $e_1, e_2 \in E$ are called independent if they do not share an endpoint. A *matching* M in G is a set of mutually independent edges. By MAXIMUM_MATCHING, we denote the problem of finding a matching in G of size as large as possible. For each $e \in E$, introduce a variable x_e with the interpretation that

$$x_e = \begin{cases} 1 & \text{if } e \text{ is included in the matching } M, \\ 0 & \text{otherwise.} \end{cases}$$

In order to force M to be a matching, we take any vertex u of G, and let $Nbr(u)$ denote the set of neighbors of u in G. We then require

$$\sum_{v \in Nbr(u)} x_{(u,v)} \leqslant 1 \quad \text{for each } u \in V.$$

26.3.4 Satisfiability problem

Let ϕ be a Boolean formula in the conjunctive normal form (CNF). Let n denote the number of variables $x_1, x_2, \ldots, x_n$ in ϕ, and m the number of clauses in ϕ. We have $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each C_j is a clause. MAX_SAT refers to the problem of finding an assignment of the variables such that the maximum possible number of clauses are satisfied.

We use the variables $x_1, x_2, \ldots, x_n$ as $0,1$-valued ILP variables. We introduce m additional variables $y_1, y_2, \ldots, y_m$ with the interpretation that

$$y_j = \begin{cases} 1 & \text{if the clause } C_j \text{ is satisfied,} \\ 0 & \text{otherwise.} \end{cases}$$

Our goal is to

$$\text{maximize} \quad \sum_{j=1}^{m} y_j.$$

In order to relate the variables x_i to the variables y_j, we introduce the notations:

$$\begin{aligned} \text{Idx}_j^+ &= \{i \in \{1,2,\ldots,n\} \mid \text{the variable } x_i \text{ appears in uncomplemented form in } C_j\}, \\ \text{Idx}_j^- &= \{i \in \{1,2,\ldots,n\} \mid \text{the variable } x_i \text{ appears in complemented form in } C_j.\} \end{aligned}$$

The number of literals satisfied in Clause C_j is $\sum_{i\in \mathrm{Idx}_j^+} x_i + \sum_{i\in \mathrm{Idx}_j^-}(1-x_i)$. Moreover, C_j is satisfied if and only if this count is at least one, so we require

$$\sum_{i\in \mathrm{Idx}_j^+} x_i + \sum_{i\in \mathrm{Idx}_j^-}(1-x_i) \geqslant y_j \quad \text{for all } j = 1,2,\ldots,m.$$

26.3.5 Traveling salesperson problem

Consider n sites numbered $1,2,3,\ldots,n$ with a cost of travel c_{ij} from Site i to Site j for $i \neq j$. We may or may not have $c_{ij} = c_{ji}$ for all i, j $(i \neq j)$, but for the ILP formulation, this is not an important issue. The traveling salesperson problem (TSP) deals with the computation of a closed tour through the n sites with each site visited only once. The object is to minimize the cost of the tour. In order to formulate the TSP as an ILP problem, let us introduce variables x_{ij} for $i \neq j$, having the interpretation that

$$x_{ij} = \begin{cases} 1 & \text{if the tour visits Site } j \text{ immediately after Site } i, \\ 0 & \text{otherwise.} \end{cases}$$

The task is to

$$\text{minimize} \quad \sum_{i=1}^{n}\sum_{\substack{j=1\\ j\neq i}}^{n} c_{ij}x_{ij}.$$

We need constraints to insure that we have constructed a Hamiltonian cycle. First, we take care of that every site j is reached from exactly another site. This is captured by the linear equality

$$\sum_{\substack{i=1\\ i\neq j}}^{n} x_{ij} = 1 \quad \text{for each } j \in \{1,2,3,\ldots,n\}.$$

Likewise, we reach exactly another site immediately after leaving every site i. This demands the equality constraint

$$\sum_{\substack{j=1\\ j\neq i}}^{n} x_{ij} = 1 \quad \text{for each } i \in \{1,2,3,\ldots,n\}.$$

These two sets of constraints alone do not suffice to make the tour a Hamiltonian cycle. For example, two disjoint cycles covering all the sites pass all of these constraints. To disallow all sub-tours, we take any non-empty proper subset $S \subsetneqq \{1,2,3,\ldots,n\}$ and denote $\overline{S} = \{1,2,3,\ldots,n\} \setminus S$. We then require

$$\sum_{i\in S}\sum_{j\in \overline{S}} x_{ij} \geqslant 1 \quad \text{for all } S.$$

The count of such subsets S is $2^n - 2$, that is, there are exponentially many constraints. So this is not a good ILP formulation for the TSP.

In order to have a polynomial-sized formulation, we leave the comfort zone of $0,1$-valued LP, and introduce integer-valued variables $u_2, u_3, \ldots, u_n$. We impose the constraints

$$u_i - u_j + nx_{ij} \leqslant n-1 \quad \text{for all } i,j \in \{2,3,\ldots,n\}, i \neq j.$$

First, let us see that these constraints eliminate sub-tours. If the solution contains multiple sub-tours, take any one of these not containing Site 1. Let T denote the set of sites visited by the sub-tour, and $t = |T|$. There are exactly t pairs i, j with $x_{ij} = 1$ for $i, j \in T$, $i \neq j$. If the above set of inequalities is

satisfied, we add them up for these i, j pairs. The u_i values cancel each other, leaving the inequality $nt \leqslant (n-1)t$, which is absurd for any $t > 0$.

Next, we see that these constraints allow Hamiltonian cycles. Assume that the tour starts (and ends) at Site 1. Let u_i denote the serial number of visiting Site i. If $(1, i_2, i_3, \ldots, i_n, 1)$ is a Hamiltonian cycle, we take $u_{i_k} = k$ for $k = 2, 3, \ldots, n$. We have $u_i \leqslant n$ and $u_j \geqslant 2$, so $u_i - u_j \leqslant n-2$ for all $i, j \in \{2, 3, \ldots, n\}$, $i \neq j$. Therefore if $x_{ij} = 0$, the inequality $u_i - u_j + nx_{ij} \leqslant n-1$ is satisfied (since $n-2 \leqslant n-1$). On the contrary, if $x_{ij} = 1$, then Site j is visited immediately after Site i, that is, $u_j = u_i + 1$, and so $u_i - u_j + nx_{ij} = n-1$.

26.3.6 Network flow problem

Let $G = (V, E)$ be a (directed) network with source s, sink t, and weight capacity $c_e \geqslant 0$ for each $e \in E$. The task is to compute the maximum flow in the network. For each $u \in V$, let us denote $\mathrm{Out}(u) = \{v \in V \mid (u, v) \in E\}$ and $\mathrm{In}(u) = \{v \in V \mid (v, u) \in E\}$. We assume that $\mathrm{In}(s) = \emptyset$ and $\mathrm{Out}(t) = \emptyset$. The flow along edge e is treated as a variable f_e. The objective is to

$$\text{maximize} \quad \sum_{v \in \mathrm{Out}(s)} f_{(s,v)}.$$

The edge capacity constrains are specified as

$$f_e \leqslant c_e \quad \text{for all } e \in E.$$

Finally, we need to ascertain that for any node u other than s and t, the total flow in equals the total flow out, that is,

$$\sum_{v \in \mathrm{In}(u)} f_{(v,u)} - \sum_{v \in \mathrm{Out}(u)} f_{(u,v)} = 0 \quad \text{for all } u \in V \setminus \{s, t\}.$$

If the edge capacities c_e are real values, this is an instance of (real-valued) LP. If c_e are integral, then the edge flows can be restricted to non-negative integer values, and we have an instance of ILP.

26.4 Relaxation and rounding

Let U be a discrete optimization problem, and I an input instance for U. We formulate U as an ILP problem, and convert I to an instance J for the ILP. We assume that $|J|$ is a polynomial in $|I|$. The ILP problem may as such be difficult to solve. We treat the the variables $x_1, x_2, \ldots, x_n$ as continuous variables, and the ILP problem now becomes an LP problem. This process is called *relaxation*. For example, if x_i is a $0, 1$-valued variable, we replace the requirement $x_i \in \{0, 1\}$ by the requirement

$$0 \leqslant x_i \leqslant 1.$$

Similarly, if $x_i \in \mathbb{N}_0$, we replace this requirement by

$$x_i \in \mathbb{R}, \ x_i \geqslant 0.$$

After this relaxation, J becomes an input of an LP problem. We solve the LP instance in polynomial time (in $|J|$ and so in $|I|$). Let the optimal solution be called ROPT (R stands for relaxed) achieved by $J^* = (x_1^*, x_2^*, \ldots, x_n^*)$. ROPT and the x_i^* values are in general non-integral, that is, ROPT does not necessarily correspond to a feasible solution for U. Let OPT denote the optimal solution of the original discrete optimization problem (or the ILP) on input I (or J). Since ROPT optimizes the

objective function over a larger domain of the input variables than OPT does (real values against integer values), we must have

$$\text{ROPT} \leqslant \text{OPT}$$

if U is a minimization problem, or

$$\text{ROPT} \geqslant \text{OPT}$$

if U is a maximization problem.

In order to make some use of ROPT, we convert J^* to an integer-valued *feasible* solution J of the ILP. This conversion process is called *rounding*. This J is not guaranteed to solve U optimally. If we call SUBOPT the cost of J, we have

$$\text{ROPT} \leqslant \text{OPT} \leqslant \text{SUBOPT}$$

if U is a minimization problem, or

$$\text{ROPT} \geqslant \text{OPT} \geqslant \text{SUBOPT}$$

if U is a maximization problem. This sub-optimal solution is what we can exploit in several ways.

Suppose that we want the compute OPT exactly using a branch-and-bound algorithm. This algorithm requires a bound B on OPT. We can start with any feasible solution and use its cost to initialize B. We then start exploring the search tree, and adaptively improve B as we discover better solutions. The effectiveness of pruning during this search depends heavily on the bound B. An uneducated initialization of B may be much poorer than OPT, leading to inefficient pruning. We may instead use the sub-optimal solution SUBOPT to initialize the bound B. Since SUBOPT is obtained by changing OPT *slightly*, it may be the case that SUBOPT is not much different from OPT, and this way of initializing B may significantly boost the performance of our branch-and-bound algorithm by enhancing the effectiveness of pruning.

A second use of the sub-optimal solution is in the design of approximation algorithms. The above relations among the different optimal costs imply that

$$\frac{\text{SUBOPT}}{\text{OPT}} \leqslant \frac{\text{SUBOPT}}{\text{ROPT}}$$

if U is a minimization problem, or

$$\frac{\text{SUBOPT}}{\text{OPT}} \geqslant \frac{\text{SUBOPT}}{\text{ROPT}}$$

if U is a maximization problem. In particular, the ratio SUBOPT/ROPT is a provable bound on the approximation ratio SUBOPT/OPT. If SUBOPT/ROPT is good, we get a decent approximation algorithm for U. To sum up, this method of using LP to design approximation algorithms has the following three steps.

1. *Formulation* of the discrete optimization problem as an ILP (or $0,1$-valued LP) problem.
2. *Relaxation* of the ILP problem to an LP problem, and subsequent optimal solution of the relaxed LP problem.
3. *Rounding* the optimal solution for the LP problem to a feasible solution of the original discrete optimization problem.

The first two steps can be done in polynomial time. The last step is the most difficult one. In particular, if we plan to solve U optimally, the third step may be equivalent to NP-Complete.

However, as long as we are content with approximate solutions, this gives us a powerful technique of designing approximation algorithms for a variety of difficult discrete optimization problems.

In the last section, we have seen several examples of ILP formulation. In the rest of this section, we see some examples of relaxation and rounding.

26.4.1 Minimum vertex cover

Consider the MIN_VERTEX_COVER problem and its ILP formulation given in Section 26.3.2. The relaxation step treats the variables x_u to satisfy the real-valued inequalities $0 \leqslant x_i \leqslant 1$. Let $(x_u^*)_{u \in V}$ be a solution of the relaxed LP. We round this to a vector $(x_u)_{u \in V}$ using the formula

$$x_u = \begin{cases} 0 & \text{if } 0 \leqslant x_u^* < 0.5, \\ 1 & \text{if } 0.5 \leqslant x_u^* \leqslant 1. \end{cases}$$

In this case, x_u is *literally* the rounded value of x_u^*.

First, we need to show that this rounded solution $(x_u)_{u \in V}$ is a feasible solution. Take any $e = (u, v) \in E$. Then, the relaxed LP constraint $x_u^* + x_v^* \geqslant 1$ must be satisfied. This in turn implies that either $x_u^* \geqslant 0.5$ or $x_v^* \geqslant 0.5$ (or possibly both). But then, we must have taken either $x_u = 1$ or $x_v = 1$ (or both). Therefore the edge e is covered by the rounded solution.

Next, we derive the approximation ratio guaranteed by the rounded solution. If $x_u^* < 0.5$, we take $x_u = 0$ which satisfies $x_u \leqslant 2x_u^*$. If $x_u^* \geqslant 0.5$, we again have $2x_u^* \geqslant 1 = x_u$. Therefore the approximation ratio satisfies

$$\rho \leqslant \frac{\text{SUBOPT}}{\text{ROPT}} = \frac{\sum_{u \in V} x_u}{\sum_{u \in V} x_u^*} \leqslant \frac{\sum_{u \in V} 2x_u^*}{\sum_{u \in V} x_u^*} = 2.$$

We have therefore designed a 2-approximation algorithm for MIN_VERTEX_COVER.

26.4.2 Minimum set cover

We use the ILP formulation of the MIN_SET_COVER problem discussed in Section 26.3.2. The relaxation step replaces the constraints $x_i \in \{0, 1\}$ by $0 \leqslant x_i \leqslant 1$ for all $i = 1, 2, \ldots, m$. Let $(x_1^*, x_2^*, \ldots, x_m^*)$ be an optimal solution of the relaxed LP instance.

Rounding this solution is not done in the literal sense. For each $a \in X$, denote by f_a the frequency of a, that is, $f_a = |\text{Idx}(a)|$. Let $f = \max_{a \in X} f_a$ denote the maximum frequency. We take the rounded solution $(x_1, x_2, \ldots, x_m)$ as

$$x_i = \begin{cases} 0 & \text{if } x_i^* < 1/f \\ 1 & \text{if } x_i^* \geqslant 1/f \end{cases}$$

for all $i = 1, 2, \ldots, m$.

To prove the feasibility of the rounded solution, we take any $a \in X$. The constraint $\sum_{i \in \text{Idx(a)}} x_i^* \geqslant 1$ must be satisfied by the relaxed optimal solution. If all the x_i^* values in the sum are $< 1/f$, we have $\sum_{i \in \text{Idx(a)}} x_i^* < f_a/f \leqslant 1$, a contradiction. So $x_i^* \geqslant 1/f$ for at least one $i \in \text{Idx}(a)$. For each such i, we have taken $x_i = 1$, that is, a is covered by the rounded solution.

The rounding method ensures that $x_i \leqslant f x_i^*$ for all $i = 1, 2, \ldots, n$. Therefore the approximation ratio satisfies

$$\rho \leqslant \frac{\text{SUBOPT}}{\text{ROPT}} = \frac{\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^*} \leqslant \frac{\sum_{i=1}^{n} f x_i^*}{\sum_{i=1}^{n} x_i^*} = f,$$

that is, we have an f-approximation algorithm for the minimum set cover problem.

26.4.3 Maximum satisfiability

The maximum satisfiability problem is studied in Section 25.2.2. Indeed, a simple randomized algorithm is given with expected approximation factor $\geqslant 1/2$. Using LP relaxation techniques, we can do better. We formulate the MAX_SAT problem as an ILP problem as discussed in Section 26.3.4. Relaxing the variables x_i and y_j to real values in the interval $[0, 1]$ gives an LP instance. Let the optimal solution for this LP instance be $(x_1^*, x_2^*, \ldots, x_n^*; y_1^*, y_2^*, \ldots, y_m^*)$. We therefore have $\text{ROPT} = \sum_{j=1}^{m} y_j^*$.

The LP solution may give fractional values of x_i^*. We use randomized rounding to generate a Boolean value for x_i. To be precise, we take $x_i = 1$ with probability x_i^*, and $x_i = 0$ with probability $1 - x_i^*$. That means that we generate a random real number $r \in [0, 1]$. We take $x_i = 1$ if $r \leqslant x_i^*$, or $x_i = 0$ if $r > x_i^*$. Now, the variables x_i and consequently the variables y_j are random variables. Let S denote the random variable standing for the number of clauses that are satisfied. Calculations show that the expected value of S satisfies the inequality

$$\text{E}[S] \geqslant \left(1 - \frac{1}{e}\right) \sum_{j=1}^{m} y_j^*.$$

Therefore the expected approximation factor is

$$\text{E}(\rho) = \text{E}[S]/\text{OPT} \geqslant \text{E}[S]/\text{ROPT} \geqslant 1 - \frac{1}{e} \approx 0.632.$$

This is a definite improvement over the randomized algorithm of Section 25.2.2. In fact, it can be shown that if ϕ is in k-CNF, then we have

$$\text{E}(\rho) \geqslant 1 - \left(\frac{k-1}{k}\right)^k.$$

For a formula in 3-CNF ($k = 3$), this implies that

$$\text{E}(\rho) \geqslant \frac{19}{27} \approx 0.704.$$

26.5 LP Duality

Duality is an important concept attached to LP instances, and is quite useful for designing approximation algorithms. We have already seen an example of duality in connection with network flows (Section 21.5). We proved that for any flow f and for any cut S, T, we have $|f| \leqslant c(S, T)$. In