بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

طراحی الگوریتم‌ها

حسین فلسفین

Linear Programming

اکنون قصد معرفی مسئلهٔ ***linear programming*** را داریم. شاید این سؤال در ذهن پدید آید که این مسئله چه ارتباطی با راهبرد شاخه و کران دارد؟ در جواب باید گفت که: این مقدمه‌ای است برای معرفی یک الگوریتم مبتنی بر راهبرد شاخه و کران برای حل نمونه‌های مسئلهٔ معروف ***integer linear programming***.

A linear programme (LP) is a problem of maximising or minimising a linear expression subject to a number of linear constraints which take the form of linear expressions being less than or equal to '$\leq$', greater than or equal to '$\geq$' or equal to '$=$' given numbers.

It is usual, but not necessary, for the variables to be restricted to be non-negative by the last set of inequalities.

Maximise

$$2x_1 + 3x_2 - x_3$$

subject to

$$x_1 + x_2 \leqslant 3$$

$$-x_1 + 2x_3 \geqslant -2$$

$$-2x_1 + x_2 - x_3 = 0$$

$$x_1, x_2, x_3 \geqslant 0$$

$$x_1, x_2, x_3 \in \mathbb{R}$$

$x_1 = 1, x_2 = 2, x_3 = 0$ ☞ ***Best Objective Value*** $= 8$

به بیان دقیقتر:

A linear programming problem (LP) is a class of the mathematical programming problem, a constrained optimization problem, in which we seek to find a set of values for continuous variables ($x_1, x_2, \ldots, x_n$) that maximizes or minimizes a linear objective function z, while satisfying a set of linear constraints (a system of simultaneous linear equations and/or inequalities).

The expression to be maximised or minimised is known as the objective function. Clearly it is possible to convert a maximisation to a minimisation (and vice versa) by negating the objective function.

$$\begin{aligned}
\text{Minimize} \quad & 3x_1 - 11x_2 + 5x_3 + x_4 \\
\text{subject to} \quad & x_1 + 5x_2 - 3x_3 + 6x_4 \leq 7 \\
& -x_1 + x_2 + x_3 - 2x_4 \geq 3 \\
& x_1,\ x_2,\ x_3,\ x_4 \geq 0
\end{aligned}$$

"Linear" No x^2, xy, arccos(x), etc.

"Programming" " Planning" (term predates computer programming).

The simplex algorithm is the oldest linear-programming algorithm. The simplex algorithm does not run in polynomial time in the worst case, but it is fairly efficient and widely used in practice.

The simplex algorithm can require exponential time. The first polynomial-time algorithm for linear programming was the ellipsoid algorithm, which runs slowly in practice. A second class of polynomial-time algorithms are known as interior-point methods.

تذکر: در این کورس، از شما انتظار نداریم که یک نمونه از مسئلهٔ ***LP*** را حل کنید. نحوهٔ حل یک نمونه از مسئلهٔ ***LP*** توسط هریک از روش‌های فوق، ورای حد این کورس است.

Applications of linear programming

Linear programs arise in a variety of practical applications. Linear programming has a large number of applications. Any textbook on operations research is filled with examples of linear programming, and linear programming has become a standard tool taught to students in most business schools.

Many problems of optimal decision making can be reduced to an instance of the linear programming problem—a problem of optimizing a linear function of several variables subject to constraints in the form of linear equations and linear inequalities.

Applications of linear programming

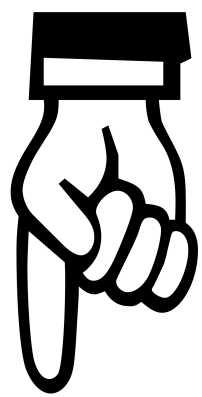
Linear programming has proved to be flexible enough to model a wide variety of important applications, such as airline crew scheduling, transportation and communication network planning, oil exploration and refining, and industrial production optimization. In fact, linear programming is considered by many as one of the most important achievements in the history of applied mathematics. The classic algorithm for this problem is called the simplex method. It was discovered by the U.S. mathematician George Dantzig in the 1940s. Although the worst-case efficiency of this algorithm is known to be exponential, it performs very well on typical inputs.

Formulating problems as linear programs

It is important to be able to recognize when we can formulate a problem as a linear program. Once we cast a problem as a polynomial-sized linear program, we can solve it in polynomial time by the ellipsoid algorithm or interior-point methods. Several linear-programming software packages can solve problems efficiently, so that once the problem is in the form of a linear program, such a package can solve it.

We express the maximum-flow problem as a linear program

Now, we consider the important problem of maximizing the flow of a material through a transportation network (pipeline system, communication system, electrical distribution system, and so on). We will assume that the transportation network in question can be represented by a connected weighted digraph with n vertices numbered from 1 to n and a set of edges E, with the following properties:

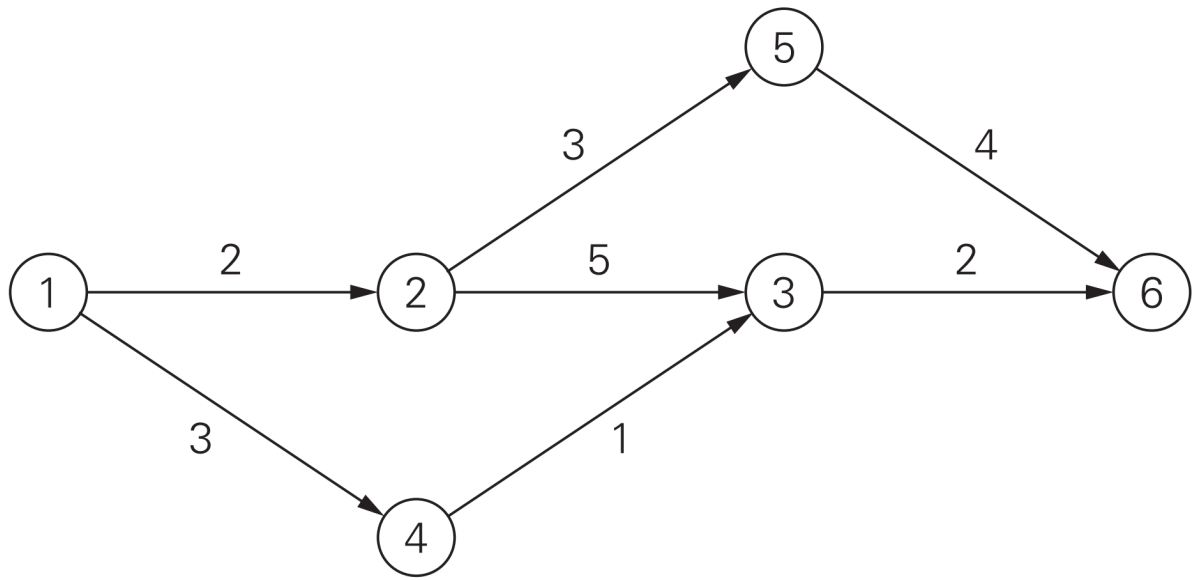


* *It contains exactly one vertex with no entering edges; this vertex is called the source and assumed to be numbered* 1.

* *It contains exactly one vertex with no leaving edges; this vertex is called the sink and assumed to be numbered* n.

* *The weight* u_{ij} *of each directed edge* (i, j) *is a positive integer, called the edge capacity. (This number represents the upper bound on the amount of the material that can be sent from* i *to* j *through a link represented by this edge.)*

A digraph satisfying these properties is called a flow network or simply a network.



It is assumed that the source and the sink are the only source and destination of the material, respectively; all the other vertices can serve only as points where a flow can be redirected without consuming or adding any amount of the material. In other words, the total amount of the material entering an intermediate vertex must be equal to the total amount of the material leaving the vertex. This condition is called the flow-conservation requirement.

If we denote the amount sent through edge (i, j) by x_{ij}, then for any intermediate vertex i, the flow-conservation requirement can be expressed by the following equality constraint:

$$\sum_{j:\,(j,i)\in E} x_{ji} = \sum_{j:\,(i,j)\in E} x_{ij} \quad \text{for } i = 2, 3, \ldots, n-1,$$

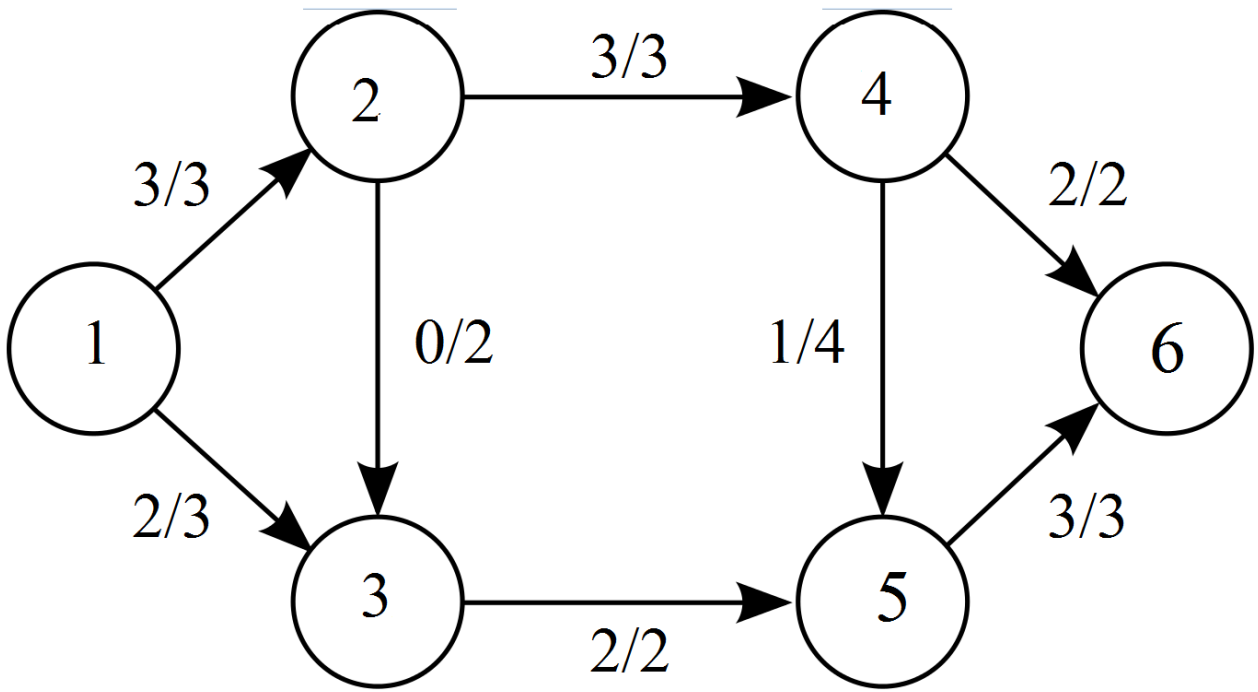
where the sums in the left- and right-hand sides express the total inflow and outflow entering and leaving vertex i, respectively.

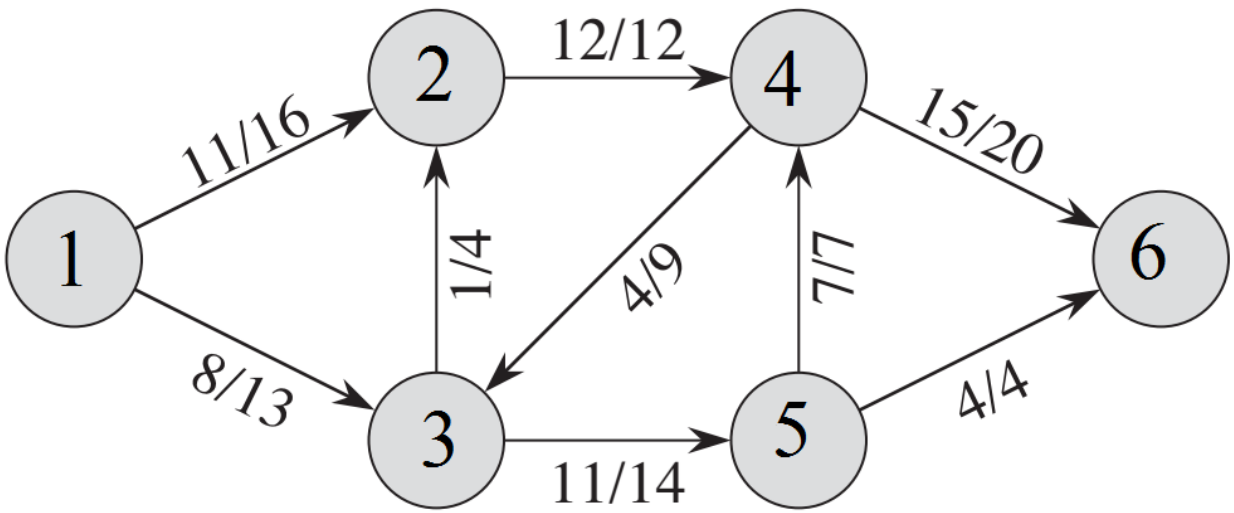
Since no amount of the material can change by going through intermediate vertices of the network, it stands to reason that the total amount of the material leaving the source must end up at the sink. Thus, we have the following equality:

$$\sum_{j:\,(1,j)\in E} x_{1j} = \sum_{j:\,(j,n)\in E} x_{jn}.$$

This quantity, the total outflow from the source—or, equivalently, the total inflow into the sink—is called the value of the flow. We denote it by v. It is this quantity that we will want to maximize over all possible flows in a network.

Thus, a (feasible) flow is an assignment of real numbers x_{ij} to edges (i, j) of a given network that satisfy flow-conservation constraints and the capacity constraints $0 \leq x_{ij} \leq u_{ij}$ for every edge $(i, j) \in E$.




1
2
3
4
5
6
11/16
8/13
12/12
1/4
4/9
7/7
15/20
4/4
11/14

تبدیل مسئلهٔ جریان بیشینه به مسئلهٔ LP

The maximum-flow problem can be stated formally as the following optimization problem:

$$\text{maximize} \quad v = \sum_{j:(1,j)\in E} x_{1j}$$

$$\text{subject to} \quad \sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = 0 \quad \text{for } i = 2, 3, \ldots, n-1$$

$$0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i,\ j) \in E.$$

We can solve the above linear programming problem by the simplex method or by another algorithm for general linear programming problems.

The Assignment Problem Model

The mathematical model for the assignment problem uses the following decision variables:

$$x_{ij} = \begin{cases} 1 & \text{if assignee } i \text{ performs task } j, \\ 0 & \text{if not,} \end{cases}$$

for $i = 1, 2, \ldots, n$ *and* $j = 1, 2, \ldots, n$. *Thus, each* x_{ij} *is a binary variable (it has value* 0 *or* 1*). Binary variables are important in OR for representing yes/no decisions. In this case, the yes/no decision is: Should assignee* i *perform task* j*? By letting* Z *denote the total cost, the assignment problem model is*

$$\text{Minimize} \qquad Z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij},$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{for } i = 1, 2, \ldots, n,$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \text{for } j = 1, 2, \ldots, n,$$

and

$$x_{ij} \geq 0, \qquad \text{for all } i \text{ and } j$$
$$(x_{ij} \text{ binary}, \qquad \text{for all } i \text{ and } j).$$

یک نکتهٔ جالب و مهم در مورد مدل فوق برای مسئلهٔ تخصیص

The first set of functional constraints specifies that each assignee is to perform exactly one task, whereas the second set requires each task to be performed by exactly one assignee. If we delete the parenthetical restriction that the x_{ij} be binary, the model clearly is a special type of linear programming problem and so can be readily solved. Fortunately we can delete this restriction.

It might appear that this problem demands integer linear programming in order to ensure that x_{ij} can only take the values 0 or 1. Fortunately, however, the integrality property mentioned above holds. If we solve an assignment problem as a conventional LP model, we can be certain that the optimal solution will give integer values to the x_{ij} (0 or 1).

We are not always so lucky!

پس تا اینجا، مسئلهٔ ***LP*** را معرفی کردیم، و گفتیم که این مسئله را می‌توان با الگوریتم‌هایی که پیچیدگی زمانی بدترین حالت آنها از مرتبهٔ چندجمله‌ای است حل کرد. همچنین، به‌عنوان یک کاربرد از این مسئله، تشریح کردیم که چطور می‌توان یک نمونه از مسئلهٔ جریان بیشینه را به یک نمونهٔ ***LP*** تبدیل کرد، و با بهره‌گیری از یک ***LP-solver***، نمونهٔ حاصل را حل کرد تا از این طریق، مسئلهٔ جریان بیشینهٔ نظیر نیز حل شود. این جلسه به‌سراغ مسئلهٔ ***integer linear programming*** خواهیم رفت (تعریف، نحوهٔ حل، و کاربردها).

An integer linear programming problem (ILP) is a linear programming problem in which at least one of the variables is restricted to integer values.

It is important to stress that the simplex method and Karmarkar's interior point algorithm can successfully handle only linear programming problems that do not limit its variables to integer values.

When variables of a linear programming problem are required to be integers, the linear programming problem is said to be an integer linear programming problem. Except for some special cases, integer linear programming problems are much more difficult.

ILP models are generally much more difficult to solve than comparably sized LP models.

NP-HARDNESS: There is no known polynomial-time algorithm for solving an arbitrary instance of the general integer linear programming problem and such an algorithm quite possibly does not exist.

Other approaches such as the branch-and-bound technique are typically used for solving integer linear programming problems.

Reduction/Modeling/Formulation

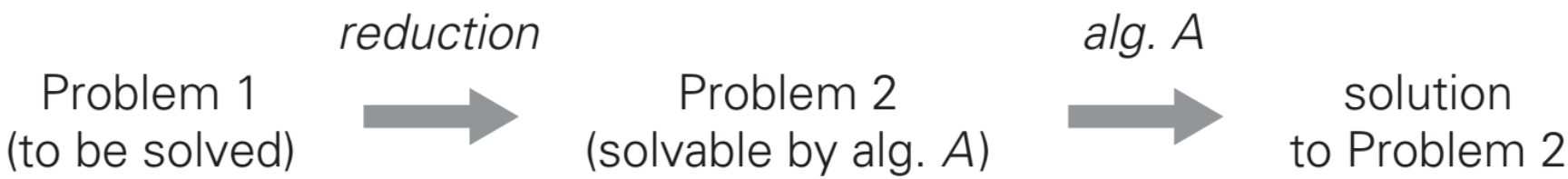
Problem reduction: *If you need to solve a problem, reduce it to another problem that you know how to solve.*

Many problems of optimal decision making can be reduced to an instance of the (integer) linear programming problem.

Problem reduction calls for transforming a given problem to another problem that can be solved by a known algorithm. Among examples of applying this idea to algorithmic problem solving, reductions to (integer) linear programming are especially important.

نمونه‌هایی از تبدیل را جلسهٔ پیش و ابتدای این جلسه دیدیم: مسئلهٔ جریان بیشینه و مسئلهٔ تخصیص

Problem reduction strategy



A simple example

Let us see how the 0-1/fractional knapsack problem can be reduced to a linear programming problem.

Recall from that the knapsack problem can be posed as follows: Given a knapsack of capacity W and n items of weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$, find the most valuable subset of the items that fits into the knapsack.

We consider first the continuous (or fractional) version of the problem, in which any fraction of any item given can be taken into the knapsack:

☞ *Let x_j, $j = 1, 2, \ldots, n$, be a variable representing a fraction of item j taken into the knapsack.*

☞ *Obviously, x_j must satisfy the inequality $0 \leq x_j \leq 1$.*

☞ *Then the total weight of the selected items can be expressed by the sum $\sum_{j=1}^{n} w_j x_j$, and their total value by the sum $\sum_{j=1}^{n} v_j x_j$.*

Thus, the continuous version of the knapsack problem can be posed as the following linear programming problem:

$$\text{maximize} \quad \sum_{j=1}^{n} v_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq W$$

$$0 \leq x_j \leq 1 \quad \text{for } j = 1, \ldots, n.$$

تذکر: البته این صرفاً یک مثال برای آشنایی با روند مدلسازی است، وگرنه روشی که در فصل راهبرد حریصانه برای حل نمونه‌های مسئلهٔ کوله‌پشتی کسری پیشنهاد کردیم بهتر از مدلسازی و سپس استفاده از یک ***LP-solver*** است.

In the discrete (or 0-1) version of the knapsack problem, we are only allowed either to take a whole item or not to take it at all. Hence, we have the following integer linear programming problem for this version:

$$\text{maximize} \quad \sum_{j=1}^{n} v_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \le W$$

$$x_j \in \{0, 1\} \quad \text{for } j = 1, \ldots, n.$$

This seemingly minor modification makes a drastic difference for the complexity of this and similar problems constrained to take only discrete values in their potential ranges.

Despite the fact that the 0-1 version might seem to be easier because it can ignore any subset of the continuous version that has a fractional value of an item, the 0-1 version is, in fact, much more complicated than its continuous counterpart.

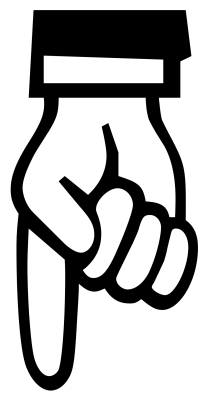
پس در مجموع:

Linear programming concerns optimizing a linear function of several variables subject to constraints in the form of linear equations and linear inequalities. There are efficient algorithms capable of solving very large instances of this problem with many thousands of variables and constraints, provided the variables are not required to be integers. The latter, called integer linear programming, constitute a much more difficult class of problems.

Another example: set-covering problem

Cover all the elements $\{1, 2, 3, 4, 5\}$ ***using the minimum number of subsets*** $\{1, 2, 3, 5\}$, $\{1, 2, 4, 5\}$, $\{1, 3, 4\}$, $\{2, 3, 4, 5\}$, $\{3, 4, 5\}$.

We use 0-1 ***variables*** δ_j ***which equal*** 1 ***if and only if the*** j***th subset is used. Each element of the original set gives rise to a constraint. In order to cover an element the sum of the variables representing the subsets containing the element must be at least*** 1***. The model then becomes:***



Minimise

$$\delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5$$

subject to

$$\delta_1 + \delta_2 + \delta_3 \geqslant 1$$

$$\delta_1 + \delta_2 + \delta_4 \geqslant 1$$

$$\delta_1 + \delta_3 + \delta_4 + \delta_5 \geqslant 1$$

$$\delta_2 + \delta_3 + \delta_4 + \delta_5 \geqslant 1$$

$$\delta_1 + \delta_2 + \delta_4 + \delta_5 \geqslant 1$$

$$\delta_1, \delta_2, \delta_3, \delta_4, \delta_5 \epsilon \{0, 1\}$$

استفاده از راهبرد شاخه و کران برای حل نمونه‌های مسئلهٔ *ILP*

Maximise

$$x_1 + x_2$$

subject to

$$2x_1 + 2x_2 \geqslant 3$$

$$-2x_1 + 2x_2 \leqslant 3$$

$$4x_1 + 2x_2 \leqslant 19$$

$$x_1, x_2 \geqslant 0$$

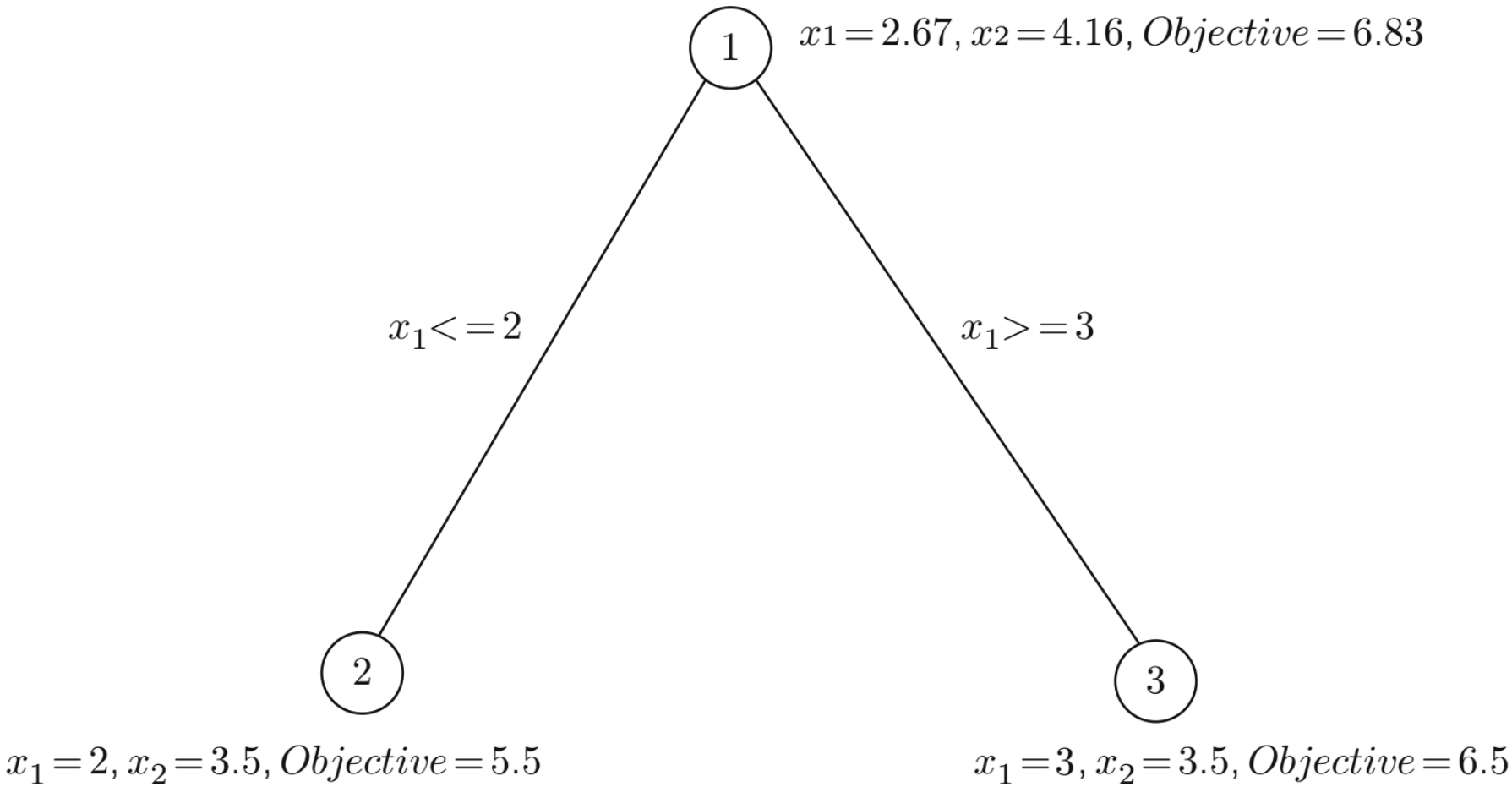
$$x_1, x_2 \in \mathbb{Z}$$

The first step in this method is to remove the integrality requirements and solve the resultant LP model. This is referred to as the LP relaxation. LP is computationally 'easy' compared with ILP. If the optimal solution to the LP relaxation turns out to be integral then this will also be an optimal solution to the ILP. There is an important class of models where this will always be the case. However, for this example, we obtain the fractional solution:

$$x_1 = 2\frac{2}{3}, \quad x_2 = 4\frac{1}{6}, \quad objective = 6\frac{5}{6}.$$

We now choose one of the integer variables (x) whose value has come out fractional (e.g., $N+f$ where N is an integer and $0 < f < 1$). There is no loss of generality in stipulating that either $x \leq N$ or $x \geq N+1$ since x is not permitted to take any fractional value between N and $N+1$. Both these possibilities rule out the current fractional solution. For the example we choose the dichotomy $x_1 \geq 2$ or $x_1 \leq 3$.

REMARK: The choice of variable x_1 over x_2 is arbitrary here. In practice it can be made heuristically, i.e. according to plausible (but not guaranteed) rules which might be hoped to produce the optimal integer solution as quickly as possible. For example, it might be thought that since x_1 is further from its closer integer than x_2, this will have more effect than choosing x_2. It is convenient to represent this process as a branching in a tree


1
$x_1 = 2.67, x_2 = 4.16, Objective = 6.83$
$x_1 <= 2$
$x_1 >= 3$
2
3
$x_1 = 2, x_2 = 3.5, Objective = 5.5$
$x_1 = 3, x_2 = 3.5, Objective = 6.5$

At each node we give the solution of the corresponding LP relaxation. Node 2 corresponds to the original model together with the appended constraint $x_1 \leq 2$. *Node 3 corresponds to the original model together with the appended constraint* $x_1 \geq 3$.

به بیان دیگر:

* *First solve the problem as a continuous problem that is an LP problem. If the solutions satisfy integrality conditions, then stop.*

* *Assume that a variable x is required to be an integer, but is found to be fractional. The portion of the feasible space indicated by $\lfloor x \rfloor < x < \lceil x \rceil$ cannot include any feasible integer solutions of x. Therefore, a feasible integer value of x must satisfy one of the two conditions: either $x \leq \lceil x \rceil$ or $x \geq \lfloor x \rfloor$.*

* *When one applies these two conditions to the continuous LP, the search space is divided into two mutually exclusive subproblems.*

* *The original problem has now been branched or divided into two subproblems.*

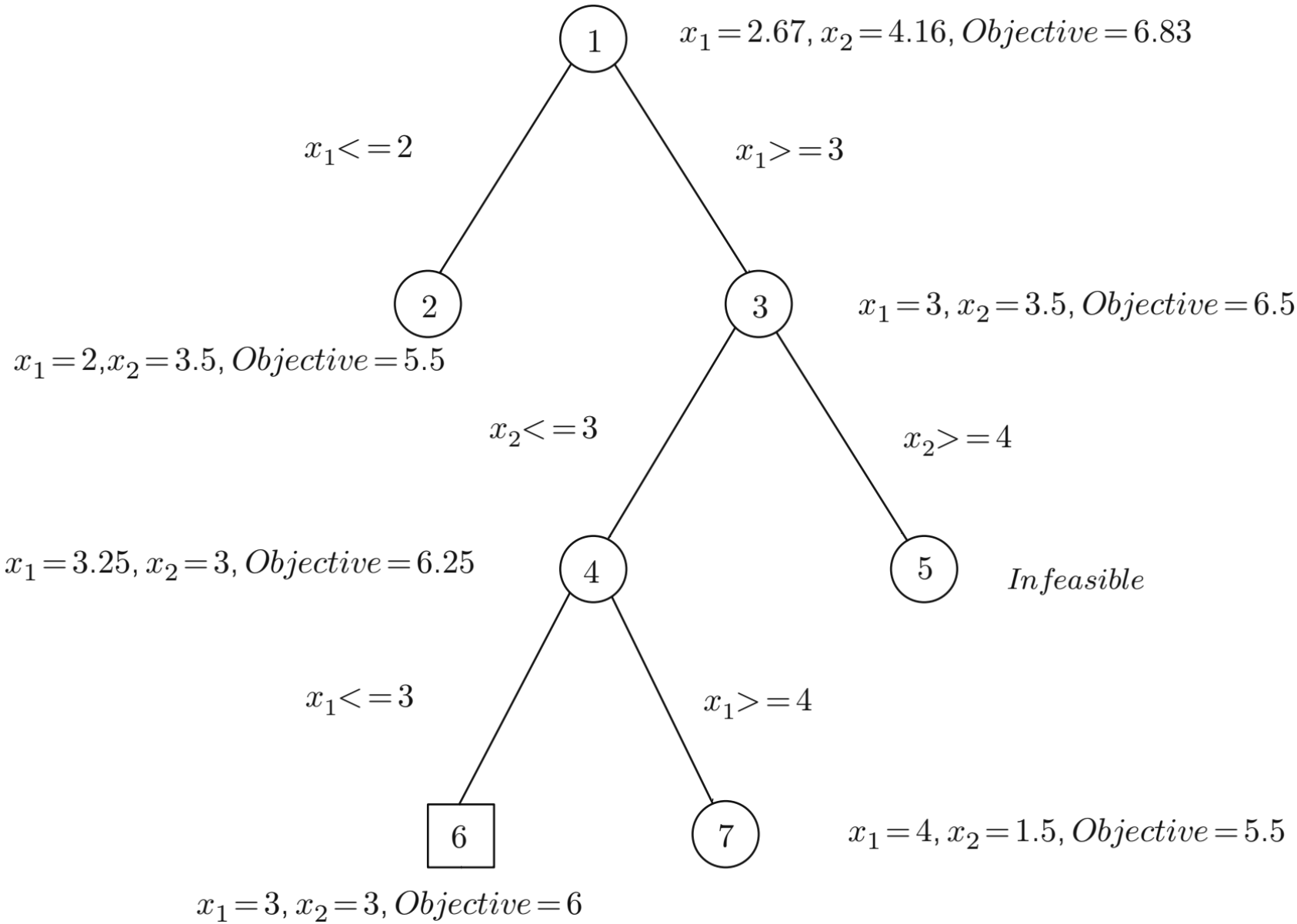
در دو موضع نیاز به تصمیم‌گیری داریم:

When an LP solution contains several fractional integer variables, the decision of which integer variable to branch on next is needed. The following rules are commonly used for choosing a branching variable:

1. *Variable with fractional value closest to 0.5*
2. *Variable with highest impact on objective function*
3. *Variable with the least index*

A decision is also needed as to which unpruned node to explore first. The most commonly used search strategies include

1. *Depth-first (last-in first-out; solve the most recently generated subproblem first)*
2. *Best-bound-first*


1
$x_1 = 2.67, x_2 = 4.16, Objective = 6.83$
$x_1 <= 2$
$x_1 >= 3$
2
3
$x_1 = 3, x_2 = 3.5, Objective = 6.5$
$x_1 = 2, x_2 = 3.5, Objective = 5.5$
$x_2 <= 3$
$x_2 >= 4$
$x_1 = 3.25, x_2 = 3, Objective = 6.25$
4
5
$Infeasible$
$x_1 <= 3$
$x_1 >= 4$
6
7
$x_1 = 4, x_2 = 1.5, Objective = 5.5$
$x_1 = 3, x_2 = 3, Objective = 6$

At node 5 the LP relaxation (and therefore the corresponding ILP) is infeasible. This result is likely to happen as we are adding progressively stronger constraints down a branch. Once a node has become infeasible there is no sense in developing it further. It is said to have been fathomed.

Node 4 is then developed in preference to the other waiting node 2 if we use the heuristic that the bound at node 4 is better than that at node 2.

At node 6 we obtain an integer solution with an objective value of 6. Again there is no sense in developing this node further which is also said to have been fathomed.

Since the objective value is better than that at node 2, no better integer solution can be obtained by developing that node. It is also, therefore, said to have been fathomed.

Solving the LP relaxation at node 7 gives an objective value of $5\frac{1}{2}$ which again is below the current bound allowing us to terminate this branch. Since there are no more waiting nodes the integer solution found at node 6 has been shown to be optimal.

Note that in some texts, the term pruned may be replaced by fathomed to indicate that no further exploration beyond that point is necessary.