Lecture Notes on Integer Linear Programming

Roel van den Broek

October 15, 2018

These notes supplement the material on (integer) linear programming covered by the lectures in the course *Algorithms for Decision Support*.

History

26 October, 2017 First version

15 October, 2018 Added missing note on dictionaries

Linear Programming

In an optimization problem we typically have to select the *best solution* from the set of all solutions, the *solution space*, that satisfy the *constraints* of the problem. The evaluation of the quality of a solution is based on an *objective* function that we have to either *minimize* or *maximize*.

A simple example of an optimization problem is the *"Healthy" Diet* problem. Bob is a Computer Science student who, surprisingly, is not particularly fond of going to the gym. As Bob still wants to following a somewhat healthy lifestyle, he decides to compose a diet that meets the daily reference intake of vitamins with the minimal amount of calories. Unfortunately for Bob, he can only get pizzas and burritos near his place. The nutritional values of a slice of pizza and a burrito are shown below, in Table 1.

	A	C	D	Calories
Pizza	225	100	200	600
Burrito	600	100	75	300
Intake	1800	550	600	

Table 1: The nutritional values of a slice of pizza and a burrito, as well as the required daily intake of vitamins. Please note that the numbers used in this example are fictional.

In this example, a solution to the optimization problem is a meal of pizzas and burritos. The variables of a diet are the number of slices of pizza, $x_p \geq 0$, and the number of burritos, $x_b \geq 0$. As low-calorie meals are preferred, the objective will be the minimization of calories in the diet,

$$\text{minimize } 600x_p + 300x_b.$$

Based on this objective function, the optimal solution would be to eat nothing at all. However, to achieve the daily reference intake, any

meal has to respect the constraints on the amount of vitamins,

$$225x_p + 600x_b \geq 1800 \quad \text{(A)},$$
$$100x_p + 100x_b \geq 550 \quad \text{(C)},$$
$$200x_p + 75x_b \geq 600 \quad \text{(D)}.$$

As not eating violates the vitamin constraints, the empty meal is an *infeasible* solution. A *feasible* solution satisfies all constraints of the optimization problem. Figure 1 shows the constraints as well as the area containing feasible solutions, called the *feasible region*[1]. Furthermore, we can see in Figure 1 that the optimal meal[2] consists of eating four burritos and one and a half slices of pizza each day, for a total of 2100 calories.

[1] The feasible region is empty if no solution satisfies all constraints, in which case the optimization problem itself is infeasible.

[2] **Disclaimer** you should not take this "optimal" solution as sound dietary advise. Any reliance you place on these diets is strictly at your own risk.

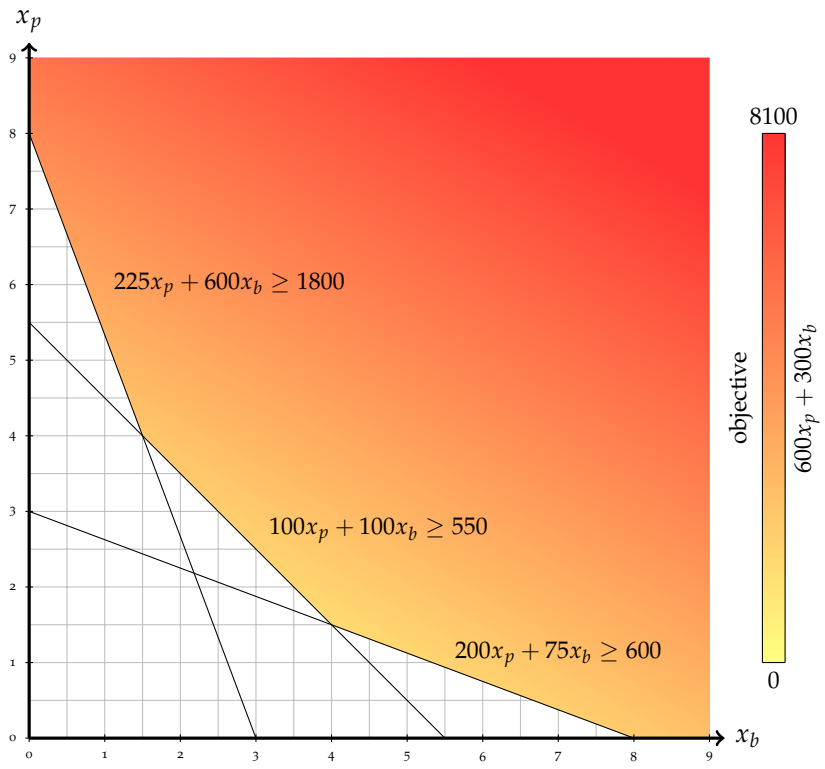

Figure 1: The colored area contains the feasible solution to the *"Healthy" Diet* problem. The gradient shows the value of the objective function in the solution space.

The formulation of the *"Healthy" Diet* problem is an example of *Linear Programmaing (LP)*, also known as *Linear Optimization*. In linear programming, a solution is represented of one or more variables, which are called decision variables, and the domain of each variable is an interval on the real line. Furthermore, both the objective and the constraints are linear[3] in the variables.

The general linear programming formulation of a minimization[4]

[3] The linearity property of linear programs means that, with two variables x and y, the objective and constraints can contain expressions such as $x + y$ and $y - x$, but, for example, not $\frac{x}{y}$, x^2 or $x \cdot y$, as the latter three are non-linear expressions.

[4] Maximizing the objective function $f(x)$ is equivalent to minimizing $-f(x)$.

problem is

$$\begin{aligned}
&\text{minimize} \quad \sum_{i=1}^{n} c_i x_i \\
&\text{subject to} \\
&\sum_{i=1}^{n} a_{i1} x_i \leq b_1 \\
&\qquad \vdots \\
&\sum_{i=1}^{n} a_{im} x_i \leq b_m \\
&\qquad x_i \geq 0 \qquad \forall i \in \{1, \ldots, n\} \quad \text{(domain)},
\end{aligned} \tag{1}$$

where solutions are encoded by n decision variables, x_1 to x_n, with associated costs c_1 to c_n, and the objective is to minimize the total cost. The decision variables are subject to m constraints of the form[5] $\sum_{i=1}^{n} a_{ij} x_i \leq b_j$, and n domain constraints, $x_i \geq 0$. An optimal solution is any solution that satisfies the constraints and has minimal cost[6].

[5] Linear constraints such as $\sum a_i x_i = b$ or $\sum a_i x_i \geq b$ can be rewritten to this form as well. Strict inequalities such as $\sum a_i x_i < b$ are — usually — not allowed in a linear program, since the optimal solution to the LP might not be well-defined. For example, in the single variable LP

$$\begin{aligned}
\max \quad & x \\
\text{s.t.} \quad & x < 1 \\
& x \geq 0,
\end{aligned}$$

no solution is maximal.

For many applications, it is easier to use the matrix form of an LP instead of the sum formulation shown in (1). Denote vectors $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{c} = (c_1, \ldots, c_n)$, $\boldsymbol{b} = (b_1, \ldots, b_m)$ and let

$$\mathcal{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}.$$

[6] An optimization problem is called *unbounded* it is feasible and we can find an arbitrarily good feasible solution, i.e. the constraints in the problem do not produce an upper bound on the goodness of feasible solutions.

Then the linear program (1) can be written in matrix form[7] as

$$\begin{aligned}
&\text{minimize} \quad \boldsymbol{c}^T \boldsymbol{x} \\
&\text{subject to} \\
&\mathcal{A}\boldsymbol{x} \leq \boldsymbol{b} \\
&\quad \boldsymbol{x} \geq 0.
\end{aligned} \tag{2}$$

[7] The notation $\boldsymbol{x} \geq 0$ indicates that all elements x_i of $\boldsymbol{x}$ should be at least zero.

Techniques to find the optimal solution of a linear program is not covered in the lecture notes. Examples are shown on the lecture slides and in the first two chapters of Chvatal[8].

[8] Vasek Chvatal. *Linear Programming*. Macmillan, 1983

Modeling

Linear programming is a flexible technique that can be applied to many real-world problems. A major advantage of modeling a problem as an LP is that linear programs are efficiently solvable. That is, the computation time of an LP is polynomial[9] in the number of

[9] In complexity theory we would denote this property as $LP \in \mathcal{P}$. More on this in a few lectures.

variables and constraints. With the current state-of-the-art of commercial and open-source solvers, it is rarely necessary or beneficial to implement your own custom solver. The major challenge of linear programming is in the problem modeling: how do we translate an optimization problem to a linear program that can be processed efficiently by a solver? What decision variables will we use to encode the solutions of the problem, and how can we rewrite the problem constraints to linear equations? To further complicate matters, there are problems for which we cannot formulate linear programs[10]. As there are no algorithms available that decide how, and if, an optimization problem can be modeled as an LP, modeling often has to be done manually. In the next sections, we will look at several examples of optimization problems, and show you how they can be modeled as linear programs.

[10] Fortunately, extensions to linear programming allow us to model a much broader class of optimization problems, albeit at the cost of computation time in solvers.

Assignment Problem

In the ASSIGNMENT problem, we have n jobs that need to be performed; each job takes T time to complete. As we obviously do not want to do the work ourselves, we hire n workers to perform the jobs. Each worker can be hired for at most T time units. The cost of hiring a certain worker depends on both the employment duration and the job he/she has to perform; if worker i spends a fraction a of its time on job j, it will cost us a times C_{ij}. The objective of the ASSIGNMENT problem is to assign the workers to jobs such that all jobs are completed with the lowest total cost.

To model this problem as a linear program, we need to have decision variables that encode all possible solutions, i.e. the different assignments. A solution to the ASSIGNMENT problem should state for each worker the time[11] it spends on each job. As the processing time of a job equals the maximum hiring duration of a worker, this is equivalent to stating the *fraction* of job j completed by worker i. Therefore, we introduce for each worker-job pair (i, j) the variable $x_{ij} \in [0, 1]$ that indicates the fraction of job j that is performed by worker i. The objective then becomes the minimization of the sum of all variables x_{ij} multiplied by their cost C_{ij}.

[11] Note that the problem description does not limit a worker to a single job; workers can switch between jobs.

To ensure that each job j is completed, we have the constraint that the sum of the fraction of work spend by all workers on job j is precisely one. Similarly, the work load of each worker i, the total fraction of their time allocated to all jobs, should be no more than one. The LP formulation of this model is shown below, (3).

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}x_{ij} \tag{3a}$$

s.t.

$$\sum_{i=1}^{n} x_{ij} \leq 1 \qquad \forall j \in \{1,\dots,n\} \quad \text{(work load)} \tag{3b}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in \{1,\dots,n\} \quad \text{(job completion)} \tag{3c}$$

$$x_{ij} \in [0,1] \quad \forall i,j \in \{1,\dots,n\} \quad \text{(domain)} \tag{3d}$$

In the current problem statement, workers are allowed to be assigned to multiple jobs, the only requirement is that they should not perform more than a single jobs worth of work. In practice, it might be inefficient if a worker has to switch jobs. Therefore, the constraint that each job is completed by a single worker is preferable. That is, worker i performs job j either entirely or not at all. To model this constraint we restrict the domain of the x_{ij} variables, constraint (3d), to binary values,

$$x_{ij} \in \{0,1\} \quad \forall i,j \in \{1,\dots,n\} \quad \text{(binary domain).} \tag{3d$'$}$$

The meaning of the variables stays the same,

$$x_{ij} = \begin{cases} 1 & \text{if worker } i \text{ performs job } j, \\ 0 & \text{otherwise.} \end{cases}$$

Integer Linear Programming

The program described by (3) with the additional constraints (3d′) is an example of *Integer Linear Programming*, abbreviated as *ILP* or *IP*, where each variable is restricted to integer values[12]. Integer linear programming is an important tool in combinatorial optimization, as many problems feature discrete decisions that can be modeled in an ILP. We will examine a few examples of such problems in these lecture notes.

In our first example of an LP, the *"Healty" Diet* problem, suppose that Bob wants to avoid meals that consist of partial burritos or pizza slices, as he does not like to eat leftovers of the previous day. Similar to the ASSIGNMENT problem, Bob only has to set the domain of the variables, x_p and x_b, to integers in his model to get the desired result:

[12] Models that contain both integer and continuous variables are known in literature as *Mixed Integer (Linear) Programs* or *MI(L)P*s.

$$\begin{aligned}
&\min \quad 600x_p + 300x_b && && (4a)\\
&\text{s.t.} && && \\
&225x_p + 600x_b \geq 1800 && \text{(A)} && (4b)\\
&100x_p + 100x_b \geq 550 && \text{(C)} && (4c)\\
&200x_p + 75x_b \geq 600 && \text{(D)} && (4d)\\
&x_p, x_b \in \mathbb{N}_0 && \text{(domain)} && (4e)
\end{aligned}$$

Figure 2 shows the resulting feasible region of the new integer linear program. Note that each feasible solution to the ILP is also feasible in the original LP, but not vice versa. In the ILP formulation, we now have three optimal solutions (x_p^*, x_b^*), namely $(2, 4)$, $(1, 6)$ and $(0, 8)$, each worth 2400 calories.

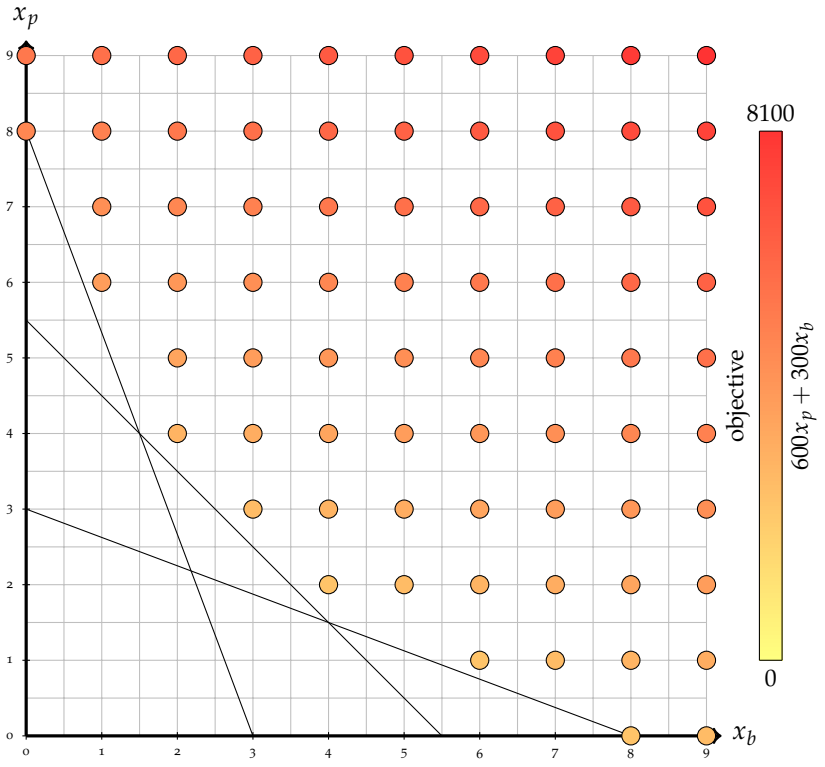

Figure 2: The colored dots are the feasible solution of the *"Healthy" Diet* problem without partial meals, the color of each dot shows the objective value.

Knapsack

In the classical KNAPSACK problem we have n valuable items, and we want to maximize our profit by selling some of them. However, the carrying capacity B of our knapsack is limited, so we have to decide

which items we will take with us. Item i has a value of c_i and weighs a_i units. Breaking items into smaller parts makes them worthless, so we are not allowed to take a fractional part of an item.

A solution to the KNAPSACK problem consists of a set of items. We can encode the solutions by introducing a variable x_i for every item i, indicating whether we take the item or not,

$$x_i = \begin{cases} 1 & \text{if we put item } i \text{ in the knapsack,} \\ 0 & \text{otherwise.} \end{cases}$$

These variables are binary, hence we are creating an integer linear program. Since an optimal solution should maximize the total profit of the selected items, and the main constraint of the problem is the capacity of knapsack, we can use the ILP model

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \\ & \sum_{i=1}^{n} a_i x_i \leq B \qquad \text{(capacity)} \\ & x_i \in \{0,1\} \quad \forall i \in \{1,\ldots,n\}. \end{aligned} \tag{5}$$

Maximum Independent Set Problem

An example of an optimization problem on graphs is the MAXIMUM INDEPENDENT SET problem. An *independent set* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ with the property that no two vertices in S are adjacent in graph G. The Maximum Independent Set problem consists of finding a largest independent set in the graph.

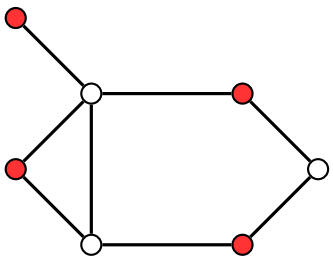

Figure 3: The colored vertices form an independent set of the graph, as no neighboring vertices are colored. Since this graph does not admit a larger independent set, the colored vertices are a maximum independent set.

Similar to the KNAPSACK problem, a natural solution representation is to associate a binary variable to each vertex in the graph, modeling the decision on whether the vertex is included in the independent set or not. Suppose that graph G has n vertices, v_1 to v_n. Then we associate a binary variable x_i to each vertex, such that

$$x_i = \begin{cases} 1 & \text{if } v_i \text{ is in the independent set,} \\ 0 & \text{otherwise.} \end{cases}$$

With this choice of variables, the objective simply becomes the maximization of the sum of all n variables. To define the constraints, we can observe that at most one vertex of each pair of neighboring vertices in the graph can be included in the independent set. This results in the following integer linear program,

$$\max \quad \sum_{i=1}^{n} x_i \tag{6a}$$

s.t.

$$x_i + x_j \leq 1 \qquad \forall (v_i, v_j) \in E \quad \text{(non-adjacent)} \tag{6b}$$

$$x_i \in \{0, 1\} \quad \forall v_i \in V \quad \text{(domain)}. \tag{6c}$$

Although modeling the MAXIMUM INDEPENDENT SET problem as an ILP is rather straightforward, finding the largest independent set is still a difficult task. In contrast to linear programs, which are all solvable to optimality in polynomial time (in the number of variables and constraints), there is no polynomial bound on the computation time known for many integer linear programs[13].

[13] For the MAXIMUM INDEPENDENT SET problem, it is unlikely that such a bound exists, as it is a known $\mathcal{NP}$-hard problem. For more information, see the lectures on complexity theory.

Facility Location Problem

Another classical example in combinatorial optimization is the CAPACITATED FACILITY LOCATION problem. A manufacturing company has to meet the demand of m customers, with the demand of customer $i \in \{1, \ldots, m\}$ equal to $D_i \geq 0$. The company wants to open a number of new production facilities, such that the costs of transporting the manufactured commodity to the customers is minimal. There are n possible building sites for the facilities, and the cost of transporting of one unit of the commodity from location $j \in \{1, \ldots, n\}$ to customer i is c_{ij}. Of course, constructing a facility costs money as well, the fixed construction cost at location j being F_j. Furthermore, the production capacity at location j is limited to C_j.

In this problem we have to make two types of decisions:

- where to open the facilities, and
- how customers obtain their goods.

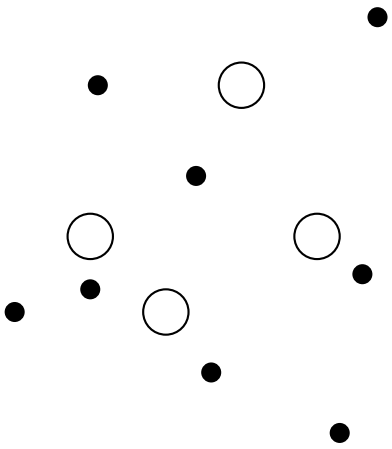

Figure 4: In the FACILITY LOCATION problem, there are n locations, shown in as circles, available for the facilities. These facilities have to serve m customers, the black dots.

In line with this distinction, we define two types of variables for location $j \in \{1, \ldots, n\}$ and customer $i \in \{1, \ldots, m\}$, namely the location variable

$$y_j = \begin{cases} 1 & \text{if we open a facility at location } j \\ 0 & \text{otherwise,} \end{cases}$$

and the transportation variable

$$q_{ij} = \text{ the quantity of goods transported from location } j \text{ to } i.$$

One way of modeling the problem is the mixed integer linear

program (7):

$$\min \quad \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij}q_{ij} + \sum_{j=1}^{n} F_j y_j \tag{7a}$$

s.t.

$$\sum_{j=1}^{n} q_{ij} = D_i \qquad \forall i \in \{1,\dots,m\} \quad \text{(demand)} \tag{7b}$$

$$\sum_{i=1}^{n} q_{ij} \le C_j y_j \qquad \forall j \in \{1,\dots,n\} \quad \text{(capacity)} \tag{7c}$$

$$q_{ij} \ge 0 \qquad \forall i \in \{1,\dots,m\}, j \in \{1,\dots,n\} \tag{7d}$$

$$y_j \in \{0,1\} \quad \forall j \in \{1,\dots,n\}. \tag{7e}$$

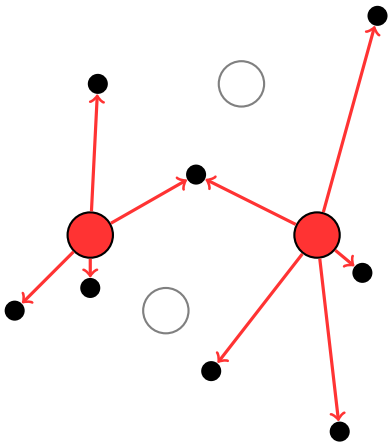

Figure 5: A possible solution to the Facility Location problem shown in Figure '4. The locations with a facility are colored. The arcs shown by which facilities each customer is served. This solution does not satisfy the constraint that each customer should be supplied by exactly one facility.

In this formulation, Equation (7b) ensures that each costumer is served. Equation (7c) limits the total amount of commodity shipped from a location to its capacity if a facility has been built, or 0 if there is no facility at the location. The objective, Equation (7a), is to minimize the total cost of facility construction and transportation. The domains of the variables are bounded by Equations (7d) and (7e).

In a real-world setting, it is usually preferable that customers receive receive their ordered goods from a single facility, as it simplifies the shipping process. In that case, we are no longer interested in what fraction of the demand we sent from a facility to a customer, as this will always be either zero or the entire demand. We can model this with the binary decision variables

$$x_{ij} = \begin{cases} 1 & \text{if customer } i \text{ receives its demand from location } j, \\ 0 & \text{otherwise.} \end{cases}$$

Substituting the x_{ij} variables for the fractional transportation variables q_{ij} in model (7) — multiplying with D_i where necessary — yields the integer linear program (8):

$$\begin{aligned} \min \quad & \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ij}D_i x_{ij} + \sum_{j=1}^{n} F_j y_j \\ \text{s.t.} \quad & \\ & \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in \{1,\dots,m\} \quad \text{(demand)} \\ & \sum_{i=1}^{n} D_i x_{ij} \le C_j y_j \qquad \forall j \in \{1,\dots,n\} \quad \text{(capacity)} \\ & x_{ij} \in \{0,1\} \quad \forall i \in \{1,\dots,m\}, j \in \{1,\dots,n\} \\ & y_j \in \{0,1\} \quad \forall j \in \{1,\dots,n\}. \end{aligned} \tag{8}$$

Towards solving an ILP: LP-relaxation

In our example of the *"Healty" Diet* problem, we restricted the domain of the decision variables to integer values to find a meal without leftovers. The solution space of the resulting integer linear program was a subset of the set of feasible solutions of the original problem, with the integral optimum containing slightly more calories than the best fractional solution. In general, restricting the domains of the variables will never lead to a better solution. The converse also holds: relaxing the domain of a variable from integers to a continuous real interval — that encompasses the original integral values — never results in a worse optimum. The linear program obtained by relaxing the integrality constraints of an ILP is known as the *LP-relaxation* of the original problem. As linear programs can be solved more efficiently than integer linear programs, LP-relaxations provide an efficient procedure to find a bound[14] on the optimum of an ILP:

1. Relax the integrality constraints.
2. Solve the resulting linear program.

If the optimal solution to the LP-relaxation happens to be integral, then you do not even have to solve the ILP itself, because the bound guarantees that it will not find a better solution. Even if the LP-relaxation has a fractional optimal solution, the bound it provides is crucial in solving the ILP, as we will see in a few sections.

In the example of the CAPACITATED FACILITY LOCATION problem, we have seen that there can be multiple models of the same problem, and the optimal objective value of these models should be the same. However, the LP-relaxations of the different models do not have not result in the same bound on the optimum, as their solutions spaces can be different. Due to the importance of the LP-relaxation bounds, we usually want to find a model with strong bound on the ILP. In the next sections, we will see some examples of problems with multiple models, for which we can prove that the LP-relaxation of one model gives a better bound than the other.

[14] The LP-relaxation provides a lower bound on the optimal objective value for minimization problems, and an upper bound in case of maximization.

The LP-relaxation of the integer linear program

$$\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & \\ & \mathcal{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \in \mathbb{N}_0 \end{aligned}$$

is the linear program

$$\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & \\ & \mathcal{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq 0. \end{aligned}$$

Facility Location Problem (Revisited)

A common variant of the FACILITY LOCATION problem is to remove the capacity constraints of the facilities, i.e. each facility can produce enough to supply all customers. Furthermore, in this UNCAPACITATED FACILITY LOCATION problem we require each customer to be served by exactly one facility. We can model this problem with a small modification to the ILP formulation shown in Equation (8). The capacity constraint is no longer needed. However, we do have to

ensure that customer i is only served from location j if we selected j as the construction site for a facility. Using the notation $d_{ij} = c_{ij}D_i$, we get the ILP formulation

$$\min \quad \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}x_{ij} + \sum_{j=1}^{n} F_j y_j \tag{9a}$$

s.t.

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in \{1,\ldots,m\} \tag{9b}$$

$$x_{ij} \leq y_j \qquad \forall i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\} \tag{9c}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\} \tag{9d}$$

$$y_j \in \{0,1\} \quad \forall j \in \{1,\ldots,n\}. \tag{9e}$$

Equation (9c) creates nm constraints, one for each of transportation variable, to prevent customers being served by non-existing facilities. We can construct a more compact model by observing that if all transportation variables x_{1j} to x_{mj} of location j are at most y_j, then the sum of the variables x_{ij} will not be more than my_j. Therefore, we can combine the nm separate transportation constraints into n aggregated transportation constraints per location:

$$\min \quad \sum_{j=1}^{n}\sum_{i=1}^{m} d_{ij}x_{ij} + \sum_{j=1}^{n} F_j y_j \tag{10a}$$

s.t.

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in \{1,\ldots,m\} \tag{10b}$$

$$\sum_{i=1}^{m} x_{ij} \leq m y_j \qquad \forall j \in \{1,\ldots,n\} \tag{10c}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\} \tag{10d}$$

$$y_j \in \{0,1\} \quad \forall j \in \{1,\ldots,n\}. \tag{10e}$$

Both the separated model, (9), and the aggregated model, (10), have the same optimal value, yet the latter ILP has fewer constraints, suggesting that the compact formulation might be the preferred model for this problem. However, a comparison of the LP-relaxations of both models — obtained by replacing the domain constraints in the ILPS with $0 \leq x_{ij} \leq 1$ and $0 \leq y_j \leq 1$ — will show that the separated model provides a stronger bound on the optimal solution of the ILP.

Theorem 1. *The lower bound on the optimum value of the* Uncapacitated Facility Location *problem obtained from the LP-relaxation*

of separated model, Equation (9)*, is at least as high as the bound of the LP-relaxation of the aggregated model, Equation* (10)*.*

Proof. Let $\boldsymbol{P}_{ILP}$, $\boldsymbol{P}_{LPS}$ and $\boldsymbol{P}_{LPA}$ be the sets of feasible solutions of respectively the ILP model[15], the LP-relaxation of the separated model (9) and the LP-relaxation of the aggregated model (10). Every feasible solution to the ILP is feasible for the two LP-relaxations as well, hence

$$\boldsymbol{P}_{ILP} \subseteq \boldsymbol{P}_{LPS} \quad \text{and}$$
$$\boldsymbol{P}_{ILP} \subseteq \boldsymbol{P}_{LPA}.$$

[15] The solution space and optimal value of models (9) and (10) is the same.

Recall our earlier observation that, for any $j \in \{1, \ldots, n\}$,

$$\forall i \in \{1, \ldots, m\} : x_{ij} \leq y_j \implies \sum_{i=1}^{m} x_{ij} \leq m y_j.$$

Since the two models differ only in constraints (9c) respectively (10c), each solution in $\boldsymbol{P}_{LPS}$ is also a feasible solution to the aggregated model:

$$\boldsymbol{P}_{LPS} \subseteq \boldsymbol{P}_{LPA}.$$

In contrast, not all feasible solutions to the LP-relaxation of the aggregated model satisfy the constraints in (9c). Let's consider the following example of two customers, c_1 and c_2, and two locations, l_1 and l_2. A possible solution to this problem is $y_1 = y_2 = 1/2$, $x_{11} = x_{22} = 1$ and $x_{12} = x_{21} = 0$, as is shown in Figure 6. Each customer is fully served in this solution and, since

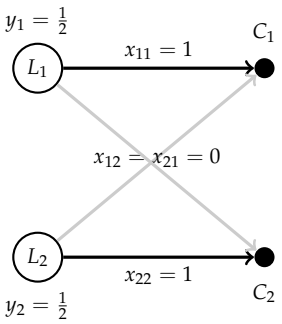

Figure 6: A fractional solution to the Uncapacitated Facility Location problem with two locations and two customers.

$$x_{11} + x_{21} = 1 + 0 \leq 2 * \frac{1}{2} = 2 * y_1 \quad \text{and}$$
$$x_{12} + x_{22} = 1 + 0 \leq 2 * \frac{1}{2} = 2 * y_2,$$

this solution is feasible for the LP-relaxation of the aggregated model in Equation (10). However, the solution is not in $\boldsymbol{P}_{LPS}$, as it does not satisfy all constraints in the LP-relaxation of the separated model:

$$x_{11} = 1 > \frac{1}{2} = y_1.$$

Therefore, we have the following relation on the feasible solution sets:

$$\boldsymbol{P}_{ILP} \subseteq \boldsymbol{P}_{LPS} \subseteq \boldsymbol{P}_{LPA}.$$

Let the optimal values of the three solution sets be Z_{ILP}, Z_{LPS} and Z_{LPA}, then the relation above implies that

$$Z_{ILP} \leq Z_{LPS} \leq Z_{LPA},$$

showing that the bound of the LP-relaxation of the separated model is at least as good as the bound obtained from the aggregated model of the Uncapacitated Facility Location problem. □

Minimum Spanning Tree

A common task when working on graphs is to find a *minimum spanning tree*. A *tree* T of a connected, undirected graph $G = (V, E)$ is a subset of the edges E that is both connected and without cycles. Tree T is a *spanning tree* if it connects all n vertices V in G. Let c_e be the cost of including edge $e \in E$ in the spanning tree, then a *minimum spanning tree* is a spanning tree of minimum total cost of the included edges. An example is shown in Figure 7.

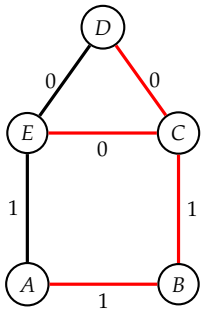

Figure 7: An undirected graph with edge costs. The edges of a *minimum spanning tree* are colored.

Minimum spanning trees are usually constructed with an optimal, greedy algorithm, as it is much more efficient than known ILP models. Nevertheless, formulating the MINIMUM SPANNING TREE problem as an integer linear program allows us to look at some useful modeling patterns and provides another opportunity to compare LP-relaxations.

We will formulate two different models of the MINIMUM SPANNING TREE problem in this section. Both models use the same binary variables, one for each edge $e \in E$ in the graph:

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

We use two basic properties of a spanning tree T to create the first model:

1. $|T| = |V| - 1$, i.e. the size of the tree is precisely one less than the number of vertices in the graph[16].
2. T has no cycles.

[16] The proof of this property is left as an exercise to the reader. Have fun.

The first property can be formulated directly as a constraint. To model the second property, we use the (equivalent) property that any subset $S \subset V$ connected by $|S|$ or more edges contains a cycle. Let

$$E(S) = \{e = (v, w) \in E \mid v, w \in S\},$$

be the set of edges between vertices in $S \subset V$, then the spanning tree T has no cycle in the set of vertices S if the sum of all x_e with $e \in E(S)$ is at most $|S| - 1$. This type of constraint is known as a *subtour elimination* constraint. To ensure that the entire spanning tree does not contain a cycle, we have to add such a constraint for every subset S of V, as can be seen in the first ILP model in Equation (11),

$$\min \quad \sum_{e \in E} c_e x_e \tag{11a}$$

s.t.

$$\sum_{e \in E} x_e = |V| - 1 \tag{11b}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \quad \forall S \subset V \quad \text{(subtour)} \tag{11c}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E. \tag{11d}$$

Our second model does not translate the acyclicity property directly to a set of constraints. Instead of subtour elimination, it uses the property that in any partition $(S, V \setminus S)$[17] of the graph $G = (V, E)$ the spanning tree has at least one edge from S to $V \setminus S$. Let the *cut set* $\delta(S)$ of $S \subset V$ be the set of edges connecting S to $V \setminus S$,

$$\delta(S) = \{e = (v, w) \in E \mid v \in S, w \in V \setminus S\},$$

then the *cut set model* is

[17] A partition $(S, V \setminus S)$ of a graph $G = (V, E)$ splits the graph into two sets, $S \subseteq V$ and $V \setminus S$, removing all edges that do not connect a vertex in S to a vertex in $V \setminus S$. That is, the partition $(S, V \setminus S)$ is the graph $G' = (V, E')$ with $E' = \{e = (v, w) \in E \mid v \in S, w \in V \setminus S\}$.

$$\min \quad \sum_{e \in E} c_e x_e \tag{12a}$$

s.t.

$$\sum_{e \in E} x_e = |V| - 1 \tag{12b}$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subset V \quad \text{(cut set)} \tag{12c}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E. \tag{12d}$$

Unfortunately, the number of constraints in the two models grows exponentially with the size of the graph due to Equation (11b) and (12c). Although this makes both models impractical for large graphs, we can prove that the LP-relaxation of the subtour model gives a stronger lower bound on the optimum.

Theorem 2. *The lower bound on the optimal value of the* Minimum Spanning Tree *problem obtained from the LP-relaxation of the subtour model, Equation* (11)*, is as least as high as the bound of the LP-relaxation of the cut set model, Equation* (12) *.*

Proof. Let $\boldsymbol{P}_{LPS}$ and $\boldsymbol{P}_{LPC}$ be the feasible regions of the LP-relaxations of respectively the subtour model and the cut set model. We start by showing that each feasible solution to relaxed subtour model is feasible for the relaxed cut set models as well.

First note that, for any subset S of V, we have that

$$E(S) \cup \delta(S) \cup E(V \setminus S) = E, \tag{13}$$

as each edge $e \in E$ connects either two vertices in S, two vertices in $V \setminus S$ or a vertex in S to a vertex in $V \setminus S$.

Let $\boldsymbol{x} = \{x_e \mid e \in E\} \in \boldsymbol{P}_{LPS}$, i.e. a feasible solution to the LP-relaxation of the subtour model, then for any vertex set $S \subset V$

$$\begin{aligned}
|V| - 1 &= \sum_{e \in E} x_e && \text{by Equation (11b)} \\
&= \sum_{e \in E(S)} x_e + \sum_{e \in \delta(S)} x_e + \sum_{e \in E(V \setminus S)} x_e && \text{by Equation (13)} \\
&\leq |S| - 1 + \sum_{e \in \delta(S)} x_e + |V| - |S| - 1 && \text{by Equation (11c)} \\
&= |V| - 2 + \sum_{e \in \delta(S)} x_e,
\end{aligned}$$

which we can rewrite to

$$\sum_{e \in \delta(S)} x_e \geq 1. \tag{14}$$

As this holds for every subset of vertices in V, any feasible solution to the LP-relaxation of model (11) satisfies the cut set constraints in Equation (12c); hence, $\boldsymbol{x}$ belongs to $\boldsymbol{P}_{LPC}$, the feasible region of the LP-relaxation of the cut set model.

The example in Figure 8 shows that converse does not hold, as some solutions in $\boldsymbol{P}_{LPC}$ are not feasible with respect to the LP-relaxation of the subtour model. We conclude that

$$\boldsymbol{P}_{LPS} \subset \boldsymbol{P}_{LPC},$$

and, analogue to the proof of Theorem 1, that the lower bound of subtour model is at least as good as the lower bound of the cut set model. □

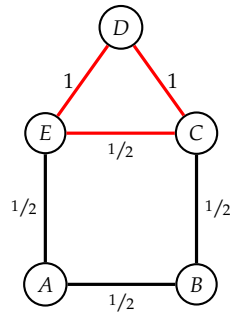

Figure 8: A fractional, feasible solution to the LP-relaxation of the cut set model, Equation (12), with the values of variables x_e on the edges. The total cost of this solution is 3/2. The colored edges do not satisfy the subtour constraint in Equation (11c).

Modeling the Minimum Spanning Tree problem as an ILP might seem like a pointless exercise, as both models contain an exponential number of constraints, and faster algorithms are available for this problem. However, the two constraint types shown in this section are applicable to other, more difficult problems as well. For example, a solution to the classical Travelling Sales Person problem, where we have to find a shortest tour that visits all vertices in a graph, should not contain any cycles smaller than the number of vertices in the graph. An ILP model with subtour constraints is much more interesting[18] for such a computationally hard problem, than it is for the Minimum Spanning Tree problem.

[18] We still have the "minor" problem of the exponential number of subtour constraints. However, it turns out that, in practice, you rarely need all of them to find a solution without subtours. A common strategy is to start without (many) subtour constraints, and iteratively solve the model, adding new constraints for any subtours in the resulting solution, until a solution without subtours is found.

Solving an ILP: branch-and-bound

We have seen examples of modeling optimization problems as integer linear programs, and showed that we can get a bound on the optimal

objective value by relaxing the integrality constraints. However, this does not provide us with an algorithm to actually solve the ILP models. We might be fortunate enough to obtain an integral solution from the LP-relaxation, but this will not happen for all problem types and instances.

The standard framework used to solve an ILP is *branch-and-bound*, where we repeatedly divide the problem in smaller subproblems. This framework creates a tree structure, with at the root the original problem. Each parent node in the tree splits, or *branches*, into multiple child nodes by creating copies of the parent problem with additional constraints, such that the child solution spaces partition the parent solution space.

If we would only branch, then the leaves of the tree would correspond to subproblems that either are infeasible or have all their variables fixed. By constructing the entire tree, we enumerate all feasible solutions to the original problem, and are thus guaranteed to find the optimum. However, as the size of the tree tends to grow exponentially with the number of variables in the model, it will take a long time to actually find the optimal solution.

To avoid complete enumeration of solution space, we bring the *bounding* part of branch-and-bound into play. It uses the LP-relaxation of the integer linear program of the problem at each node to obtain a lower bound — in case of a minimization problem[19] — and a heuristic that produces an upper bound on the optimal value by constructing a good, integral solution[20]. Instead of simply branching at each possible node, we use the bounds on the optimal value of the subproblems to eliminate or *bound* unpromising branches early.

[19] In a maximization problem, the LP-relaxation gives the upper bound and the construction heuristic the lower bound.

[20] Many heuristics can be used to find feasible, but not necessarily optimal, solutions to the ILP. A simple example is to round a fractional solution of the LP-relaxation to feasible integer values.

There are then four different outcomes at each node:

1. The subproblem of the branch is infeasible. In this case, we do not split the subproblem in smaller parts. **This node becomes a leaf**.

2. The solution of the LP-relaxation is integral. We have found the optimal solution of this node, so further branching on this subproblem will not result in a better solution. **This node becomes a leaf**. We update the best solution seen so far if necessary.

3. The lower bound on the current subproblem is at least as high as the objective value of the best solution found so far. Since we have already discovered a solution that is better than the best this branch has to offer, we gain nothing by exploring it any further. **This node becomes a leaf**.

4. The lower bound on the current subproblem is lower than the objective value of the best solution found so far. If we have a heuristic available, we will compute an upper bound on this subproblem,

and update the best solution seen so far if needed. **We continue branching on this node**.

As this procedure only prunes unpromising branches, we get a tremendous speed up of the search process without losing the optimal solution. However, branch-and-bound does not provide any guarantees on the computation time, and is heavily affected by the implementation. As mentioned earlier, branch-and-bound is a framework, meaning that many parts have to be filled in by the user:

- **Model** Clearly, the model has a large influence on the search process. It determines the variables on which we can branch as well as the bound of the LP-relaxation. The better the bound of the LP-relaxation, the more likely it is that we can eliminate bad branches early.
- **Integer heuristic** Similar to the LP-relaxation bound, a good heuristic for integral solutions allows early pruning of unpromising branches, reducing the solution space.
- **Search strategy** Since we are essentially constructing a tree node by node during branch-and-bound, we need to decide during the search, when the tree is not yet constructed fully, which node of the tree we wish to evaluate and branch on. Possible approaches are depth-first, breadth-first or selection strategies that expand the most "promising" node.
- **Branching strategy** In addition to selecting a node to branch on, we need to decide *how* we will branch. That is, how will we split the solution space of the selected node. A common approach when branching on decision variables is to split its domain into two parts. For example, if we have a binary decision variable $x_i \in \{0,1\}$, we can branch into $x_i = 0$ and $x_i = 1$. In case of an integral decision variable $x_j \geq 0$, we might create two[21] branches, splitting the domain in $0 \leq x_j \leq 4$ and $x_j \geq 5$. If the subproblem contains more than one decision variable on which we can branch, we need to select one of them. For example, a simple strategy for decision variables is to select the variable with the most fractional[22] value.

  Branching on other aspects, such as $x_i + x_j \leq 6$ and $x_i + x_j \geq 7$, is possible as well, as long as at least one of the branches preserves the optimal solution of the parent node in the branch-and-bound tree.

[21] Similar to a binary decision variable, if an integral decision variable has an upper bound of k, with k not too large, we branch into k child nodes, one for each possible value of the variable.

[22] The rationale behind this strategy is that fixing a highly fractional — close to $1/2$ — binary variable is likely to have a big impact on the overall solution, which hopefully allows us to prune the branch with "wrong" decision early in the search.

Knapsack (Revisited)

To illustrate the branch-and-bound framework, let us consider the following example of the KNAPSACK problem. Our knapsack has a

capacity B of 15, and we have five items that we could carry in the backpack. The values c_i and weights a_i are shown in Table 2. Using these data as input for ILP formulation (5) of the KNAPSACK problem gives us model (15):

i	1	2	3	4	5
c_i	8	12	7	15	12
a_i	4	8	3	6	5
c_i/a_i	2	$1^1/_2$	$2^1/_3$	$2^1/_2$	$2^2/_5$

Table 2: The values c_i, weights a_i and value-to-weight ratios of the five items in KNAPSACK problem.

$$\begin{aligned} \max \quad & 8x_1 + 12x_2 + 7x_3 + 15x_4 + 12x_5 \\ \text{s.t.} \quad & 4x_1 + 8x_2 + 3x_3 + 6x_4 + 5x_5 \leq 15 \\ & x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}. \end{aligned} \tag{15}$$

To use the branch-and-bound framework, we need to specify the search and branching strategies. The simple breadth-first-search strategy is used to explore the branch-and-bound tree, evaluating the nodes level by level. We base our branching strategy on the intuition that items with a large value-to-weight ratio are more favorable to put in the knapsack than low value, high weight items in the KNAPSACK problem. Therefore, it is likely that the exclusion of an item with a high value-to-weight ratio will lead to a bad solution, allowing us to cut off the corresponding branch in the tree early in the search process. The branching strategy that exploits this idea is to branch on the binary decision variables in non-increasing value-to-weight ratio: $x_5 \to x_4 \to x_3 \to x_1 \to x_2$.

Lower bounds can be obtained at each node by rounding down the fractional part of the optimal solution[23] to the LP-relaxation. Figure 9 shows the branch-and-bound tree of our example.

The exploration order of the nodes of the branch-and bound tree is shown below:

N_0. The root node corresponding to the original ILP model shown in(15). The lower bound heuristic constructs an initial solution; the global lower bound becomes 34. We branch on decision variable x_5.

N_1. The upper bound on the optimal value of this node is less than the current best solution. We eliminate this branch.

N_2. The optimal solution to the LP-relaxation is equal to the solution at the root. We branch on x_4.

N_3. The upper bound of this node is less than the current best solution. We eliminate this branch.

N_4. The optimal solution of the LP-relaxation is equal to the solution at the root. We branch on x_3.

N_5. The optimal solution of the LP-relaxation is integral, and is better than the current best solution; the global lower bound becomes 35. As the solution to the LP-relaxation is optimal for the ILP as well, we do not branch on this node.

[23] An optimal solution to the LP-relaxation can be constructed efficiently by greedily selecting items with the highest value-to-weight ration until the knapsack is full.

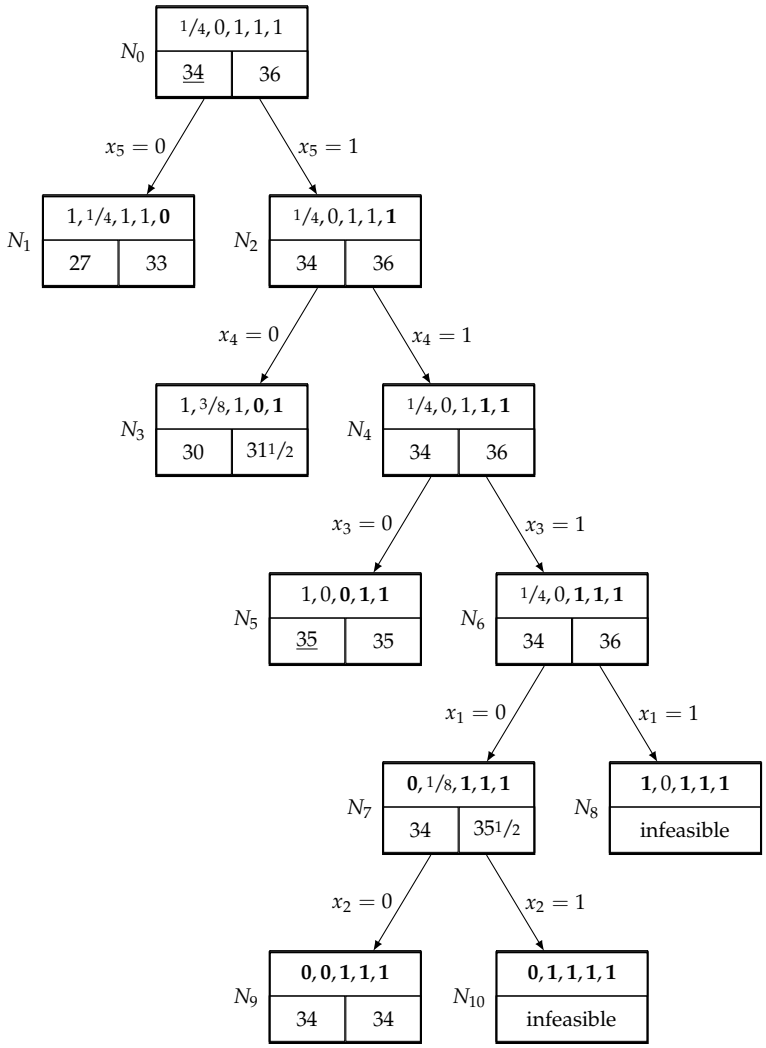

Figure 9: A branch-and-bound tree for the instance of KNAPSACK shown in Table 2. Each node shows at the top an optimal solution to the LP-relaxation of the subproblem, and at the bottom right the corresponding objective value. This is an upper bound on the optimal solution to the ILP. A lower bound, obtained by rounding down fractional decision variables in the LP solution, can be found in the bottom left of a node. The labels at the arcs show branching choices. A lower bound is underlined if it improves the current best lower bound. Bold values in the solution indicate variables that are fixed by branching. The branch-and-bound tree shows that 35 is the optimal value of this KNAPSACK instance.

N_6. Th optimal solution of the LP-relaxation is equal to the solution at the root. We branch on x_1.

N_7. We branch on x_2.

N_8. The node is infeasible, because the total weight of the fixed items, 18, exceed the maximum capacity of the knapsack. We do not branch on this node.

N_9. The upper bound of this node is less than the current best solution. We eliminate this branch.

N_{10}. The node is infeasible, because the total weight of the fixed items, 22, exceed the maximum capacity of the knapsack. We do not branch on this node.

The optimal solution is then the best solution that we have seen over all nodes. In this case, the solution consists of items 1, 4 and 5, with a total value of 35.

Valid Inequalities

Branch-and-bound relies heavily on a strong LP-relaxation bound on the optimum of each node in the search tree. In previous sections we have seen that modeling choices in the ILP affect the strength of the bound. However, a good model is not always sufficient to find the integral optimum in reasonable time. In that case, we can strengthen the LP-relaxation by introducing additional constraints, called *valid inequalities* or *cutting planes*. These constraints cut off part of the LP solution space to tighten the bound on the integral optimum, without removing integral solutions. By preserving the ILP space, we ensure that the strengthened model remains valid with regard to the original integer linear program. Enhancing branch-and-bound by adding the valid inequalities at each node to cut off fractional solutions is known as *branch-and-cut*.

Gomory's cuts

Suppose that we have an integer linear program with decision variables $x_1, \ldots, x_n \geq 0$, and a constraint of the form

$$\sum_i a_i x_i = b.$$

Then, for any integral solution that satisfies this constraint, the inequality obtained by rounding down the constant and the coefficients in the constraint,

$$\sum_i \lfloor a_i \rfloor x_i \leq \lfloor b \rfloor,$$

holds as well, since the decision variables are all non-negative. As the inequality does not cut off integral solutions, it is valid for the ILP model. This type of valid inequality is known as *Gomory's cut*.

One of the advantages of Gomory cuts is that we can generate new cutting planes efficiently from the final dictionary[24] of the LP-relaxation. Consider the following example ILP,

$$\begin{aligned} \max \quad & 2x_1 + x_2 \\ \text{s.t.} \quad & \\ & x_1 - x_2 \leq 1 \\ & 2x_1 + 2x_2 \leq 7 \\ & x_1, x_2 \in \mathbb{N}_0, \end{aligned} \tag{16}$$

depicted in Figure 10.

We turn an inequality constraints to an equality with a *slack variable*, modeling the gap between the left- and right-hand side of the original inequality. Introducing slack variables x_3 and x_4 in our model yields

$$\begin{aligned} \max \quad & 2x_1 + x_2 \\ \text{s.t.} \quad & \\ & x_1 - x_2 + x_3 = 1 \\ & 2x_1 + 2x_2 + x_4 = 7 \\ & x_1, x_2, x_3, x_4 \in \mathbb{N}_0. \end{aligned} \tag{17}$$

The optimal solution of the LP-relaxation of this model is $x_1 = 2\frac{1}{3}, x_2 = 1\frac{1}{4}, x_3 = x_4 = 0$, with a value of $5\frac{3}{4}$. In the dictionary of this solution, the non-zero decision variables x_1 and x_2 are expressed in the other, zero-valued decision variables,

$$x_1 = 2\frac{1}{4} - \frac{1}{2}x_3 - \frac{1}{4}x_4 \tag{18}$$

$$x_2 = 1\frac{1}{4} + \frac{1}{2}x_3 - \frac{1}{4}x_4. \tag{19}$$

We can derive a Gomory cut from the constraint of any fractional decision variable. For example, as $x_2 = 1\frac{1}{4}$, rewriting Equation (19) to

$$x_2 - \frac{1}{2}x_3 + \frac{1}{4}x_4 = 1\frac{1}{4},$$

and rounding down gives the Gomory cut

$$x_2 - \lfloor\frac{1}{2}\rfloor x_3 + \lfloor\frac{1}{4}\rfloor x_4 = x_2 - x_3 \leq \lfloor 1\frac{1}{4}\rfloor = 1. \tag{20}$$

Not only are Gomory cuts generated *efficiently* from the final dictionary of an LP-relaxation, these cutting planes are also *effective*. In

[24] The dictionary format of a solution to a linear program expresses all non-zero decision variables (after introducing slack variables) in the remaining zero-valued decision variables. See the first two chapters of Chvatal for more on this topic.

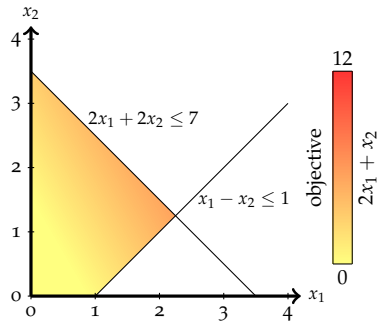

Figure 10: The feasible region of the LP-relaxation of model (16).

our example, the current optimal LP-relaxation solution does not satisfy inequality (20), since

$$1\frac{1}{4} - 0 > 1.$$

The property of cutting off the current optimal solution of the LP-relaxation holds for any Gomory cut generated from the constraint of a fraction decision variable. This property is a direct result of the expressing the non-zero decision variable in terms of the zero-valued decision variables:

$$x_k = b_k + \sum_{i:x_i=0} a_i x_i = b_k > \lfloor b_k \rfloor \text{ if } x_i \text{ is fractional.}$$

Therefore, adding the valid inequality in Equation (20) as a constraint to our example model model (17) and solving the LP-relaxation yields a new optimum of $5\frac{1}{2}$ at $x_1 = 2, x_2 = 1\frac{1}{2}$, as is illustrated in Figure 11.

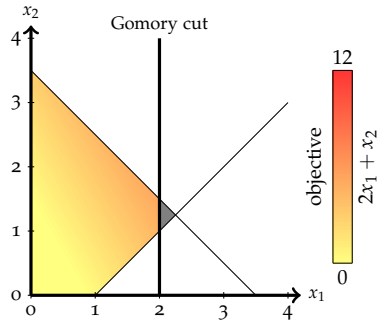

Figure 11: The feasible region of the LP-relaxation of model (16) with the Gomory cut in Equation (20). The gray area is removed by the Gomory cut.

Since the optimum of the LP-relaxation of the extended model is fractional as well, we can repeat the process of adding Gomory cuts and solving the resulting LP-relaxation, until an integral solution is found. This procedure, the *cutting plane algorithm*, is guaranteed to converge to an integral solution, although the number of cutting planes added to the model can be exponential. In branch-and-cut, we do not necessarily have to continue until an integral solution is found, as a good bound on the objective of a node in the search tree is sufficient in many cases.

Problem specific cuts

Although the previous valid inequalities are generally applicable, cutting planes that exploit the structure of a problem are often able to strengthen the bound of an LP-relaxation much more efficiently. In this section we will look at two of these problem specific valid inequalities.

The first example relates to the MAXIMUM INDEPENDENT SET problem. It is based on the observation that at most half of the vertices in a cycle in a graph can be included in an independent set. This results for model (6) in the valid inequality

$$\sum_{i \in C} x_i \leq \frac{|C| - 1}{2},$$

where $C \subseteq V$ is an odd cycle in the graph. The *odd cycle*[25] *inequality* eliminates fractional solutions such as the one shown in Figure 12.

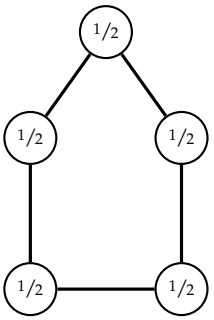

Figure 12: The fractional solution to the MAXIMUM INDEPENDENT SET problem that assigns $1/2$ to each vertex in the odd cycle is optimal, with a value of $2\frac{1}{2}$, but does not satisfy the *odd cycle* inequality.

Another type of problem specific valid inequalities can be derived for the KNAPSACK problem. Suppose that we have a subset of items

[25] Of course, a similar inequality exists for even cycles as well. However, as feasible fractional solution satisfy the even cycle inequality, we do not strengthen the LP-relaxation bound by adding it to our model.

$I \subset \{1, \ldots, n\}$ with a total weight larger than the capacity of our knapsack. Then we know that we cannot carry all items in the subset with us, i.e. every feasible solution in model (5) should satisfy

$$\sum_{i \in I} x_i \leq |I| - 1.$$

This type of cutting plane, known as a *cover inequality*, is applicable to many other problems as well, and is commonly used strategy in modern solvers to tighten the bound of the LP-relaxation. As an example of its usage, take the subset of items $I = \{1, 3, 4, 5\}$ for the problem instance listed in Table 2. Since

$$\sum_{i \in I} a_i = 4 + 3 + 6 + 5 > 15,$$

the set I covers our knapsack. The corresponding cover inequality

$$x_1 + x_3 + x_4 + x_5 \leq 3$$

cuts off the optimal solution to the LP-relaxation, $x_3 = x_4 = x_5 = 1$, $x_1 = 1/4$.

References

Vasek Chvatal. *Linear Programming*. Macmillan, 1983.