A Survey of Optimization

5

by **Patrick Boily** and **Kevin Cheung**

Traditionally, optimization has been one of the most-frequently used arrows in the operations researcher's and quantitative analyst's quiver.

From its humble beginning as an offshoot of calculus (see Chapter 2) to its current status as the crown jewel in a variety of industrial contexts (scheduling, financial engineering, transportation networks, rankings, machine learning, etc.), optimization allows users to find the largest output, the smallest wait time, the winning conditions, and so on.

Optimization problems seen in calculus classes are often solved using differential tools. In this whirlwind tour of the optimization landscape, we discuss problems that do not lend themselves to such an approach, providing a quick survey of optimization problems and algorithms, modeling techniques, an software.

1: And popular techniques for solving them.

5.1 Beginnings

We start by looking at some of the most common types of **single-objective optimization problems** that arise in practice.[1] The following toy problems introduce some of the fundamental notions.

1. Let S be the set of all the four-letter English words. What is the maximum number of ℓ's a word in S can have?

   There are numerous four-letter words that contain the letter ℓ – for example, "line", "long", "tilt", and "full". From this short list alone, we know the maximum number of ℓ's is at least 2 and at most 4. As "llll" is not an English word, the maximum number cannot be 4. Can the maximum number be 3? Yes, because "lull" is a four-letter word with three ℓ's.

   This example illustrates some fundamental ideas in optimization. In order to say that 3 is the correct answer, we need to

   - search for a word that has three ℓ's, and
   - provide an argument that rules out any value higher than 3.

   In this example, the only possible value of ℓ higher than 3 is 4, which was easily ruled out. That cannot always be done – if the problem was to find the maximum number of y's, would the same approach work?

2. A pirate lands on an island with a knapsack that can hold 50kg of treasure. She finds a cave with the following items:

Item	Weight	Value	Value/kg
iron shield	20kg	\$2800.00	\$140.00/kg
gold chest	40kg	\$4400.00	\$110.00/kg
brass sceptre	30kg	\$1200.00	\$40.00/kg

Which items can she bring back home in order to maximize her reward without breaking the knapsack?

If the pirate does not take the gold chest, she can take both the iron shield and the brass sceptre for a total value of \$4000. If she takes the gold chest, she cannot take any of the remaining items. However, the value of the gold chest is \$4400, which is larger than the combined value of the iron shield and the brass sceptre. Hence, the pirate should just take the gold chest.

Here, we performed a case analysis and **exhausted all the promising possibilities** to arrive at our answer. Note that a **greedy strategy** that chooses items in descending value per weight would give us the sub-optimal solution of taking the iron shield and brass sceptre.

Even though there are problems for which the greedy approach would return an optimal solution, the second example is not such a problem. The general version of this problem is the classic **binary knapsack problem** and is known to be **NP-hard**.[2]

2: Informally, NP-hard optimization problems are problems for which no algorithm can provide an output in polynomial time – when the problem size is large, the run time explodes.

Many real-world optimization problems are NP-hard. Despite the theoretical difficulty, practitioners often devise methods that return "good-enough solutions" using **approximation methods** and heuristics. There are also ways to obtain **bounds** to gauge the **quality** of the solutions obtained. We will be looking at these issues at a later stage.

5.2 Single-Objective Optimization Problems

A typical single-objective optimization problem consists of a **domain set** $\mathcal{D}$, an **objective function** $f : \mathcal{D} \to \mathbb{R}$, and predicates $\mathcal{C}_i$ on $\mathcal{D}$, where $i = 1, \ldots, m$ for some non-negative integer m, called **constraints**.

We want to find, if possible, an element $\mathbf{x} \in \mathcal{D}$ such that $\mathcal{C}_i(\mathbf{x})$ holds for $i = 1, \ldots, m$ and the value of $f(\mathbf{x})$ is either as high (in the case of **maximization**) or as low (in the case of **minimization**) as possible. Compactly, **single-objective optimization problems** are written down as:

$$\begin{array}{ll} \min & f(\mathbf{x}) \\ \text{s.t.} & \mathcal{C}_i(\mathbf{x}) \quad i = 1, \ldots, m \\ & \mathbf{x} \in \mathcal{D}, \end{array}$$

in the case of minimizing $f(\mathbf{x})$, or

$$\begin{array}{ll} \max & f(\mathbf{x}) \\ \text{s.t.} & \mathcal{C}_i(\mathbf{x}) \quad i = 1, \ldots, m \\ & \mathbf{x} \in \mathcal{D}, \end{array}$$

in the case of maximizing $f(\mathbf{x})$.

Here, "s.t." is an abbreviation for "subject to." Technically, "min" should be replaced with "inf" (and "max" with "sup") since the minimum value is not necessarily attained. However, we will abuse notation and ignore this subtle distinction.

Some common domain sets include:

- $\mathbb{R}_+^n$ (the set of n-tuples of non-negative real numbers)
- $\mathbb{Z}_+^n$ (the set of n-tuples of non-negative integers)
- $\{0,1\}^n$ (the set of binary n-tuples)

The **Binary Knapsack Problem** (BKP) can be formulated using the notation we have just introduced. Suppose that there are n **items**, with item i having **weight** w_i and **value** $v_i > 0$ for $i = 1, \ldots, n$.

Let K denote the **capacity** of the knapsack. Then the BKP can be formulated as:

$$\begin{array}{ll} \max & \sum_{i=1}^{n} v_i x_i \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \leq K \\ & x_i \in \{0,1\} \quad i = 1, \ldots, n. \end{array}$$

Note that there is only one constraint given by the inequality modeling the capacity of the knapsack. For the pirate example discussed previously, the BKP is:

$$\begin{array}{ll} \max & 2800x_1 + 4400x_2 + 1200x_3 \\ \text{s.t.} & 20x_1 + 40x_2 + 30x_3 \leq 50 \\ & x_1, x_2, x_3 \in \{0,1\}. \end{array}$$

5.2.1 Feasible and Optimal Solutions

An element $\mathbf{x} \in \mathcal{D}$ satisfying all the constraints (i.e., $\mathcal{C}_i(\mathbf{x})$ holds for all $i = 1, \ldots, m$) is called a **feasible solution** and its **objective function value** is $f(\mathbf{x})$. For a minimization (resp. maximization) problem, a feasible solution $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ (resp. $f(\mathbf{x}^*) \geq f(\mathbf{x})$) for every feasible solution $\mathbf{x}$ is called an **optimal solution**.

The objective function value of an optimal solution, if it exists, is the **optimal value** of the optimization problem. If an optimal value exists, it is by necessity unique, but the problem can have multiple optimal solutions. Consider, for instance, the following example:

$$\begin{array}{ll} \min & x + y \\ \text{s.t.} & x + y \geq 1 \\ & \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2 \end{array}$$

This problem has an optimal solution

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 - t \\ t \end{bmatrix}$$

for every $t \in \mathbb{R}$, but a unique optimal value of 1.

5.2.2 Infeasible/Unbounded Problems

It is possible that there exists no element $\mathbf{x} \in \mathcal{D}$ such that $\mathcal{C}_i(\mathbf{x})$ holds for all $i = 1, \ldots, m$. In such a case, the optimization problem is said to be **infeasible**. The following problem, for instance, is infeasible:

$$\begin{array}{ll} \min & x \\ \text{s.t.} & x \leq -1 \\ & x \geq 0 \\ & x \in \mathbb{R} \end{array}$$

Indeed, any solution x must be simultaneously non-negative and smaller than -1, which is patently impossible. An optimization problem that is not infeasible can still fail to have an optimal solution, however.

For instance, the problem

$$\begin{array}{ll} \max & x \\ \text{s.t.} & x \in \mathbb{R} \end{array}$$

is not infeasible, but the max/sup does not exist since the objective function can take on values larger than any candidate maximum. Such a problem is said to be **unbounded**.

On the other hand, the problem

$$\begin{array}{ll} \min & e^{-x} \\ \text{s.t.} & x \in \mathbb{R}, \end{array}$$

has a positive objective function value for every feasible solution. Even though the objective function value approaches 0 as $x \to \infty$, there is no feasible solution with an objective function value of 0. Note that this problem is **not** unbounded as the objective function value is bounded below by 0.

5.2.3 Possible Tasks

Given an optimization problem, the most natural task is to find an optimal solution (provided that one exists) and to demonstrate that it is optimal.

However, depending on the context of the problem, one might be instead tasked to find:

- a feasible solution (or show that none exists);
- a local optimum;
- a good bound on the optimal value;
- all global solutions;
- a "good" (but **not necessarily optimal**) solution, quickly;
- a "good" solution that is **robust to small changes** in problem data, and/or
- the N best solutions.

In many contexts, the last three tasks are often more important than finding optimal solutions. For example, if the problem data comes from

measurements or forecasts, one needs to have a solution that is still feasible when deviations are taken into account.

Additionally, producing multiple "good" solutions could allow decision makers to choose a solution that has desirable properties (such as political or traditional requirements) but that is not represented by, or difficult to represent with, problem constraints.

5.3 Classification of Optimization Problems and Types of Algorithms

The computational difficulty of optimization problems, then, depends on the properties of the domain set, constraints, and the objective function.

5.3.1 Classification

Problems without constraints are said to be **unconstrained**. For example, least-squares minimization in statistics can be formulated as an unconstrained problem, and so can

$$\begin{array}{ll} \min & x^2 - 3x \\ \text{s.t.} & x \in \mathbb{R} \end{array}$$

Problems with linear constraints g_i (i.e., linear inequalities or equalities) and a linear objective function f form an important class of problems in **linear programming**.

Linear programming problems are by far the **easiest** to solve in the sense that efficient algorithms exist both in theory and in practice. Linear programming is also the backbone for solving more complex models [2].

Convex problems are problems with a **convex** domain set, which is to say a set $\mathcal{D}$ such that

$$t\mathbf{x}_1 + (1-t)\mathbf{x}_2 \in \mathcal{D}$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{D}$ and for all $t \in [0,1]$, and convex constraints g_i and function f, which is to say,

$$h(t\mathbf{x}_1 + (1-t)\mathbf{x}_2) \leq th(\mathbf{x}_1) + (1-t)h(\mathbf{x}_2)$$

for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{D}$, and for all $t \in [0,1]$, $h \in \{f, g_i\}$.

Convex optimization problems have the property that **every local optimum is also a global optimum**. Such a property permits the development of effective algorithms that could also work well in practice. Linear programming is a special case of convex optimization.

Nonconvex problems (such as problems involving integer variables and/or nonlinear constraints that are not convex) are the hardest problems to solve. In general, nonconvex problems are **NP-hard**. Such problems often arise in scheduling and engineering applications.

In the rest of the chapter, we will primarily focus on linear programming and nonconvex problems whose linear constraints g_i and objective function f are linear, but with domain set $\mathcal{D} \subseteq \mathbb{R}^k \times \mathbb{Z}_+^{n-k}$.

These problems cover a large number of applications in operations research, which are often discrete in nature. We will not discuss optimization problems that arise in statistical learning and engineering applications that are modeled as nonconvex continuous models since they require different sets of techniques and methods – more information is available in [1], and in Chapters 4 and 31.

5.3.2 Algorithms

We omit the specific algorithmic details of various optimization methods,[3] as consultants and analytsts are usually expected to use **off-the-shelf** solvers for the various tasks, but it could prove insightful for analysts to know of the various types of algorithms or methods that exist for solving optimization problems.

3: Which would be better left for a graduate course on the subject anyway.

Algorithms fall into three families: **heuristics**, **exact**, and **approximate**.

Heuristics These are normally quick to execute but do not provide guarantees of optimality. For example, the **greedy heuristic** for the knapsack problem is very quick but does not always return an optimal solution.[4]

4: In fact, no guarantee exists for the "validity" of a solution in that case.

Other heuristics methods include **ant colony**, **particle swarm**, and **evolutionary algorithms**, just to name a few. There are also heuristics that are stochastic in nature and have proof of convergence to an optimal solution. **Simulated annealing** and **multiple random starts** are such heuristics.

Unfortunately, there is no guarantee on the **running time to reach optimality** and there is no way to **identify when one has reached an optimum point**.

Exact Methods Some approaches return a global optimum after a finite run time.

However, most exact methods can only guarantee that constraints are approximately satisfied (though the potential violations fall below some pre-specified tolerance). It is therefore possible for the **returned solutions to be infeasible** for the actual problem.

There also exist exact methods that fully control the error. When using such a method, an optimum is usually given as a **box guaranteed to contain an optimal solution** rather than a single element.

Returning boxes rather than single elements are helpful in cases, for example, where the optimum cannot be expressed exactly as a vector of floating point numbers.

Such exact methods are used mostly in academic research and in areas such as medicine and avionics where the tolerance for errors is practically zero.

Approximate Methods Some algorithms eventually zoom in on sub-optimal solutions, while providing a guarantee: this solution is at most ε away from the optimal solution, say.

In other words, approximate methods also provide a proof of **solution quality**.

5.4 Linear Programming

Linear programming (LP) was developed independently by G.B. Dantzig and L. Kantorovich in the first half of the 20$^{\text{th}}$ century to solve resource planning problems.

Even though linear programming is insufficient for many modern-day applications in operations research, it was used extensively in economic and military contexts in the early days.

To motivate some key ideas in linear programming, we begin with an example.

Example: A roadside stand sells lemonade and lemon juice. Each unit of lemonade requires 1 lemon and 2 litres of water to prepare, and each unit of lemon juice requires 3 lemons and 1 litre of water to prepare. Each unit of lemonade gives a profit of 3$ dollars upon selling, while each unit of lemon juice gives a profit of 2$ dollars.

With 6 lemons and 4 litres of water available, how many units of lemonade and lemon juice should be prepared in order to maximize profit?

If we let x and y denote the number of units of lemonade and lemon juice, respectively, to prepare, then the profit is the **objective function**, given by $(3x + 2y)$\$.

Note that a number of constraints must be satisfied by x and y:

- x and y should be **non-negative**;
- the number of lemons needed to make x units of lemonade and y units of lemon juice is $x + 3y$ and cannot exceed 6;
- the number of litres of water needed to make x units of lemonade and y units of lemon juice is $2x + y$ and cannot exceed 4;

Hence, to determine the maximum profit, we need to maximize $3x + 2y$ subject to x and y satisfying the constraints $x + 3y \leq 6$, $2x + y \leq 4$, $x \geq 0$, and $y \geq 0$.

A more compact way to write the problem is as follows:

$$
\begin{array}{lrcrcl}
\max & 3x & + & 2y & & \\
\text{s.t.} & x & + & 3y & \leq & 6 \\
 & 2x & + & y & \leq & 4 \\
 & x & & & \geq & 0 \\
 & & & y & \geq & 0. \\
 & x & , & y & \in & \mathbb{R}.
\end{array}
$$

It is customary to omit the specification of the domain set in linear programming since the variables always take on real numbers. Hence, we can simply write

$$\begin{array}{llrcrcl} \max & 3x & + & 2y & & \\ \text{s.t.} & x & + & 3y & \leq & 6 \\ & 2x & + & y & \leq & 4 \\ & x & & & \geq & 0 \\ & & & y & \geq & 0. \end{array}$$

We can solve the above maximization problem graphically, as follows. We first sketch the set of $[x, y]^\top$ satisfying the constraints, called the **feasible region**, on the $(x, y)-$plane.

We then take the objective function $3x + 2y$ and turn it into the equation of a line $3x + 2y = c$ where c is a parameter. Note that as the value of c increases, the line defined by the equation $3x + 2y = c$ moves in the direction of the normal vector $[3, 2]^\top$. We call this direction the **direction of improvement**. Determining the maximum value of the objective function, called the optimal value, subject to the contraints amounts to finding the maximum value of c so that the line defined by the equation $3x + 2y = c$ still intersects the feasible region.

Figure 5.1 shows the (objective function) lines with $c = 0, 4, 6.8$.

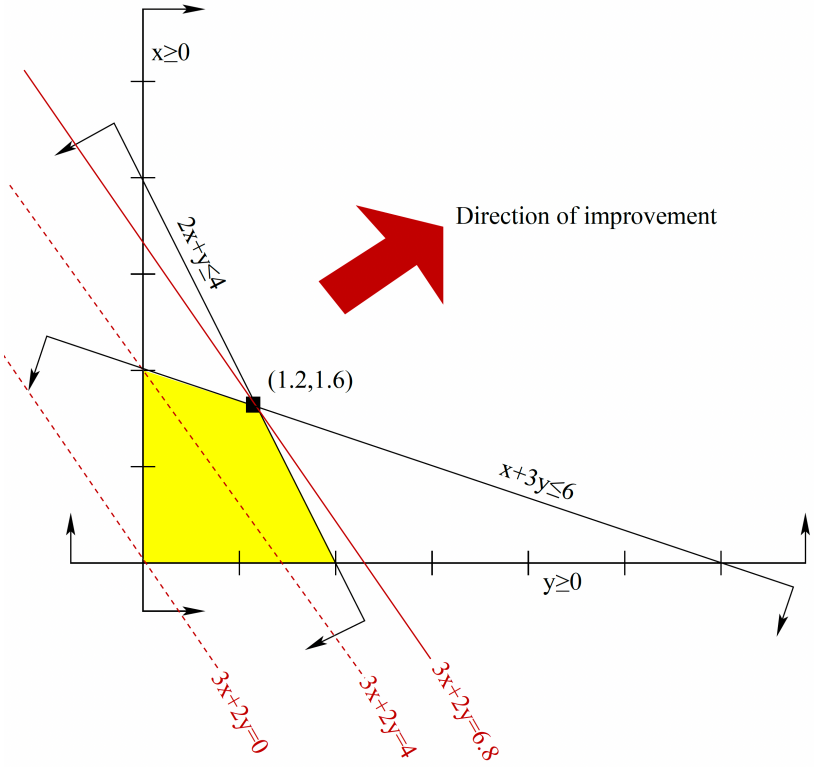


Figure 5.1: Graphical solution for the lemonade and lemon juice optimization problem; the feasible region is shown in yellow, and level curves of the objective function in red.

We can see that if c is greater than 6.8, the line defined by $3x + 2y = c$ will not intersect the feasible region. Hence, the profit cannot exceed 6.8 dollars.

As the line $3x + 2y = 6.8$ does intersect the feasible region, 6.8 is the maximum value for the objective function. Note that there is only one

point in the feasible region that intersects the line $3x + 2y = 6.8$, namely $[x^*, y^*]^\top = [1.2, 1.6]^\top$. In other words, to maximize profit, we want to prepare 1.2 units of lemonade and 1.6 units of lemon juice.

This solution method can hardly be regarded as rigorous because we relied on a picture to conclude that $3x + 2y \leq 6.8$ for all $[x, y]^\top$ satisfying the constraints. But we can also obtain this result **algebraically**.

Note that multiplying both sides of the constraint $x + 3y \leq 6$ by 0.2 yields

$$0.2x + 0.6y \leq 1.2,$$

and multiplying both sides of the constraint $2x + y \leq 4$ by 1.4 yields

$$2.8x + 1.4y \leq 5.6.$$

Hence, any $[x, y]^\top$ that satisfies both

$$x + 3y \leq 6 \quad \text{and} \quad 2x + y \leq 4$$

must also satisfy

$$(0.2x + 0.6y) + (2.8x + 1.4y) \leq 1.2 + 5.6,$$

which simplifies to $3x + 2y \leq 6.8$, as desired.

It is always possible to find an algebraic proof like the one above for linear programming problems, which adds to their appeal. To describe the full result, it is convenient to call on **duality**, a central notion in mathematical optimization.

5.4.1 Linear Programming Duality

Let P denote following linear programming problem:

$$\begin{array}{ll} \min & \mathbf{c}^\mathbf{T}\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} \geq \mathbf{b} \end{array}$$

where $\mathbf{c} \in \mathbb{R}^n$ $\mathbf{b} \in \mathbb{R}^m$ $\mathbf{A} \in \mathbb{R}^{m \times n}$ (inequality on $m-$tuples is applied component-wise.)

Then for every $\mathbf{y} \in \mathbb{R}^m_+$ (that is, all components of $\mathbf{y}$ are non-negative), the inferred inequality $\mathbf{y}^\mathbf{T}\mathbf{A}\mathbf{x} \geq \mathbf{y}^\mathbf{T}\mathbf{b}$ is valid for all $\mathbf{x}$ satisfying $\mathbf{A}\mathbf{x} \geq \mathbf{b}$.

Furthermore, if $\mathbf{y}^\mathbf{T}\mathbf{A} = \mathbf{c}^\mathbf{T}$, the inferred inequality becomes $\mathbf{c}^\mathbf{T}\mathbf{x} \geq \mathbf{y}^\mathbf{T}\mathbf{b}$, making $\mathbf{y}^\mathbf{T}\mathbf{b}$ a lower bound on the optimal value of P. To obtain the largest possible bound, we can solve

$$\begin{array}{ll} \max & \mathbf{y}^\mathbf{T}\mathbf{b} \\ \text{s.t.} & \mathbf{y}^\mathbf{T}\mathbf{A} = \mathbf{c}^\mathbf{T} \\ & \mathbf{y} \geq \mathbf{0}. \end{array}$$

This problem is called the **dual problem** of P, and P is called the **primal problem**. A remarkable result relating P and its dual P' is the **Duality Theorem for Linear Programming**: if P has an optimal solution, then so does its dual problem P', and the optimal values of the two problems are the same.

A **weaker result** follows easily from the discussion above: the objective function value of a feasible solution to the dual problem P' is a lower bound on the objective function value of a feasible solution to P. This result is known as **weak duality**. Despite the fact that it is a simple result, its significance in practice cannot be overlooked because it provides a way to gauge the quality of a feasible solution to P.

For example, suppose we have at hand a feasible solution to P with objective function value 3 and a feasible solution to the dual problem P' with objective function value 2. Then we know that the objective function value of our current solution to P is within 1.5 times the actual optimal value since the optimal value cannot be less than 2.

In general, a linear programming problem can have a more complicated form. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{c} \in \mathbb{R}^n$. Let $\mathbf{a}^{(i)^\mathbf{T}}$ denote the ith row of $\mathbf{A}$, $\mathbf{A}_j$ denote the jth column of $\mathbf{A}$, and P denote the minimization problem, with variables in the tuple $\mathbf{x} = [x_1, \cdots, x_n]^\top$, given as follows:

- the objective function to be minimized is $\mathbf{c}^\mathbf{T}\mathbf{x}$;
- the constraints are $\mathbf{a}^{(i)^\mathbf{T}}\mathbf{x} \sqcup_i b_i$, where $\sqcup_i$ is $\leq$, $\geq$, or $=$ for $i = 1, \ldots, m$, and
- for each $j \in \{1, \ldots, n\}$, x_j is constrained to be non-negative, non-positive, or **free**.

Then the **dual problem** P' is defined to be the maximization problem, with variables in the tuple $\mathbf{y} = [y_1, \cdots, y_m]^\top$ given as follows:

- the objective function to be maximized is $\mathbf{y}^\mathbf{T}\mathbf{b}$;
- for $j = 1, \ldots, n$, the jth constraint is

$$\begin{cases} \mathbf{y}^\mathbf{T}\mathbf{A}_j \leq c_j & \text{if } x_j \text{ is constrained to be non-negative} \\ \mathbf{y}^\mathbf{T}\mathbf{A}_j \geq c_j & \text{if } x_j \text{ is constrained to be nonpositive} \\ \mathbf{y}^\mathbf{T}\mathbf{A}_j = c_j & \text{if } x_j \text{ is free.} \end{cases}$$

- and for each $i \in \{1, \ldots, m\}$, y_i is constrained to be non-negative if $\sqcup_i$ is $\geq$; y_i is constrained to be non-positive if $\sqcup_i$ is $\leq$; y_i is free if $\sqcup_i$ is $=$.

The following table can help remember the correspondences:

Primal (min)	Dual (max)
$\geq$ constraint	≥ 0 variable
$\leq$ constraint	≤ 0 variable
$=$ constraint	free variable
≥ 0 variable	$\geq$ constraint
≤ 0 variable	$\leq$ constraint
free variable	$=$ constraint

Below is an example of a **primal-dual pair** of problems based on the above definition.

Consider the primal problem:

$$\left|\begin{array}{rrrrrrrl} \min & x_1 & - & 2x_2 & + & 3x_3 & & \\ \text{s.t.} & -x_1 & & & + & 4x_3 & = & 5 \\ & 2x_1 & + & 3x_2 & - & 5x_3 & \geq & 6 \\ & & & 7x_2 & & & \leq & 8 \\ & x_1 & & & & & \geq & 0 \\ & & & x_2 & & & & \text{free} \\ & & & & & x_3 & \leq & 0. \end{array}\right.$$

Here, $\mathbf{A} = \begin{bmatrix} -1 & 0 & 4 \\ 2 & 3 & -5 \\ 0 & 7 & 0 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 5 \\ 6 \\ 8 \end{bmatrix}$, and $\mathbf{c} = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$.

Since the primal problem has three constraints, the dual problem has three variables:

- the first constraint in the primal is an equation, the corresponding variable in the dual is free;
- the second constraint in the primal is a $\geq$-inequality, the corresponding variable in the dual is non-negative;
- the third constraint in the primal is a $\leq$-inequality, the corresponding variable in the dual is non-positive.

Since the primal problem has three variables, the dual problem has three constraints:

- the first variable in the primal is non-negative, the corresponding constraint in the dual is a $\leq$-inequality;
- the second variable in the primal is free, the corresponding constraint in the dual is an equation;
- the third variable in the primal is non-positive, the corresponding constraint in the dual is a $\geq$-inequality.

Hence, the dual problem is:

$$\left|\begin{array}{rrrrrrrl} \max & 5y_1 & + & 6y_2 & + & 8y_3 & & \\ \text{s.t.} & -y_1 & + & 2y_2 & & & \leq & 1 \\ & & & 3y_2 & + & 7y_3 & = & -2 \\ & 4y_1 & - & 5y_2 & & & \geq & 3 \\ & y_1 & & & & & & \text{free} \\ & & & y_2 & & & \geq & 0 \\ & & & & & y_3 & \leq & 0. \end{array}\right.$$

In some references, the primal problem is always a **maximization problem** – in that case, what we have considered to be a primal problem is their dual problem and *vice-versa*.[5]

5: Note that the **Duality Theorem for Linear Programming** remains true for the more general definition of the primal-dual pair of linear programming problems.

5.4.2 Methods for Solving LP Problems

There are currently two families of methods used by modern-day linear programming solvers: **simplex methods** and **interior-point methods**.

We will not get into the technical details of these methods, except to say that the algorithms in either family are iterative, that there is no

known simplex method that runs in polynomial time, but efficient polynomial-time interior-point methods abound in practice. We might wonder why anyone would still use simplex methods, given that they are not polynomial-time methods: simply put, simplex methods are in general more **memory-efficient** than interior-point methods, and they tend to return solutions that have few nonzero entries.

More concretely, suppose that we want to solve the following problem:

$$\begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0}. \end{array}$$

For ease of exposition, we assume that $\mathbf{A}$ has full row rank. Then, each iteration of a simplex method maintains a current solution $\mathbf{x}$ that is **basic**, in the sense that the columns of $\mathbf{A}$ corresponding to the nonzero entries of $\mathbf{x}$ are linearly independent. In contrast, interior-point methods will maintain $\mathbf{x} > \mathbf{0}$ throughout (whence the name "interior point").

When we use an off-the-shelf linear programming solver, the choice of method is usually not too important since solvers have good default settings. Simplex methods are typically used in settings when a problem needs to be resolved after minor changes in the problem data or in problems with additional integrality constraints discussed in the next section.

5.5 Mixed-Integer Linear Programming

While the simplicity of linear programming (and duality) make it an appealing tool, its modeling power is insufficient in many real-life applications (for example, there is no simple linear programming formulation of the BKP).

Fortunately, allowing the domain set to restrict one or more variables to integer values drastically extends the modeling power. The price we pay is that there is no guarantee that the problems can be solved in polynomial time.

Example: Recall the lemonade and lemon juice problem introduced in the previous section: there is a unique optimal solution at $[x, y]^\top = [1.2, 1.6]^\top$ for a profit of 6.8.

But this solution requires the preparation of **fractional units** of lemonade and lemon juice. What if the number of prepared units needs to be integers?

The solution is to add integrality constraints:

$$\begin{array}{llrcrcl} \max & 3x & + & 2y & & & \\ \text{s.t.} & x & + & 3y & \leq & 6 \\ & 2x & + & y & \leq & 4 \\ & x & & & \geq & 0 \\ & & & y & \geq & 0 \\ & x & , & y & \in & \mathbb{Z}. \end{array}$$

This problem is no longer a linear programming problem; rather, it is an **integer linear programming problem**. Note that we can solve this problem via a case analysis. The second and third inequalities tell us that the possible values for x are 0, 1, and 2.

- If $x = 0$, the first inequality gives $3y \leq 6$, implying that $y \leq 2$. Since we are maximizing $3x + 2y$, we want y to be as large as possible; $[x, y]^\top = [0, 2]^\top$ satisfies all the constraints with an objective function value of 4.
- If $x = 1$, the first inequality gives $3y \leq 5$, implying that $y \leq 1$. Note that $[x, y]^\top = [1, 1]^\top$ satisfies all the constraints with an objective function value of 5.
- If $x = 2$, the second inequality gives $y \leq 0$. Note that $[x, y]^\top = [2, 0]^\top$ satisfies all the constraints with an objective function value of 6.

Thus, $[x^*, y^*]^\top = [2, 0]^\top$ is an **optimal solution**. How does this compare to the solution of the LP problem of the previous section, both in terms of location of the solution and value of the objective function?

A **mixed-integer linear programming problem** (MILP) is a problem of minimizing or maximizing a linear function subject to finitely many linear constraints such that the number of variables are finite, with at least one of them required to take on integer values.

If all the variables are required to take on integer values, the problem is called a **pure integer linear programming problem** or simply an **integer linear programming problem**. Normally, we assume the problem data to be rational numbers to rule out pathological cases.

Many solution methods for solving MILPs have been devised and some of them first solve the **linear programming relaxation** of the original problem, which is the problem obtained from the original problem by dropping all the integrality requirements on the variables.

For instance, if P_M denotes the following MILP:

$$
\begin{array}{llrcrcrcl}
\min & & x_1 & & & + & x_3 & & \\
\text{s.t.} & -x_1 & + & x_2 & + & x_3 & \geq & 1 \\
& -x_1 & - & x_2 & + & 2x_3 & \geq & 0 \\
& -x_1 & + & 5x_2 & - & x_3 & = & 3 \\
& x_1 & , & x_2 & , & x_3 & \geq & 0 \\
& & & & & x_3 & \in & \mathbb{Z}.
\end{array}
$$

then the linear programming relaxation P_1 of P_M is:

$$
\begin{array}{llrcrcrcl}
\min & & x_1 & & & + & x_3 & & \\
\text{s.t.} & -x_1 & + & x_2 & + & x_3 & \geq & 1 \\
& -x_1 & - & x_2 & + & 2x_3 & \geq & 0 \\
& -x_1 & + & 5x_2 & - & x_3 & = & 3 \\
& x_1 & , & x_2 & , & x_3 & \geq & 0.
\end{array}
$$

Observe that the optimal value of P_1 is a lower bound for the optimal value of P_M since the feasible region of P_1 contains all the feasible solutions to P_M, thus making it possible to find a feasible solution to P_1

with objective function value which is better than the optimal value of P_M.

Hence, if an optimal solution to the **LP relaxation** happens to be a feasible solution to the original problem, then it is also an optimal solution to the original problem. Otherwise, there is an integer variable having a nonintegral value v.

What we then do is to create two new sub-problems as follows:

- one requiring the variable to be at most the greatest integer less than v,
- the other requiring the variable to be at least the smallest integer greater than v.

This is the basic idea behind the **branch-and-bound method**. We now illustrate these ideas on P_M. Solving the linear programming relaxation P_1, we find that $\mathbf{x}' = \frac{1}{3}[0, 2, 1]^\top$ is an optimal solution to P_1. Note that $\mathbf{x}'$ is not a feasible solution to P_M because x_3' is not an integer.

We now create two sub-problems P_2 and P_3. P_2 is obtained from P_1 by adding the constraint $x_3 \leq \lfloor x_3' \rfloor$,[6] and P_3 is obtained from P_1 by adding the constraint $x_3 \geq \lceil x_3' \rceil$.

6: $\lfloor a \rfloor$ denotes the **floor of** a and $\lceil a \rceil$ denotes the **ceiling of** a.

Hence, P_2 is the problem

$$
\begin{array}{llrcrcrcl}
\min & & x_1 & & & + & x_3 & & \\
\text{s.t.} & -x_1 & + & x_2 & + & x_3 & \geq & 1 \\
& -x_1 & - & x_2 & + & 2x_3 & \geq & 0 \\
& -x_1 & + & 5x_2 & - & x_3 & = & 3 \\
& & & & & x_3 & \leq & 0 \\
& x_1 & , & x_2 & , & x_3 & \geq & 0,
\end{array}
$$

and P_3 is the problem

$$
\begin{array}{llrcrcrcl}
\min & & x_1 & & & + & x_3 & & \\
\text{s.t.} & -x_1 & + & x_2 & + & x_3 & \geq & 1 \\
& -x_1 & - & x_2 & + & 2x_3 & \geq & 0 \\
& -x_1 & + & 5x_2 & - & x_3 & = & 3 \\
& & & & & x_3 & \geq & 1 \\
& x_1 & , & x_2 & , & x_3 & \geq & 0.
\end{array}
$$

Note that any feasible solution to P_M must be a feasible solution to either P_2 or P_3. Using the help of a solver, one can see that P_2 is infeasible. The problem P_3, however, has an optimal solution at $\mathbf{x}^* = \frac{1}{5}[0, 4, 5]^\top$, which is also feasible for P_M. Hence, $\mathbf{x}^*$ is an optimal solution of P_M.

In many instances, there are multiple choices for the variable on which to branch, and for which sub-problem to solve next. These choices can have an impact on the **total computation time**. But there are no hard-and-fast rules (at the moment) to determine the best branching path. This in area of ongoing research.

5.5.1 Cutting Planes

Difficult MILP problems often cannot be solved by branch-and-bound methods alone. A technique that is typically employed in solvers is to add valid inequalities to strengthen the linear programming relaxation.

Such inequalities, known as **cutting planes**, are known to be satisfied by all the feasible solutions to the original problem but not by all the feasible solutions to the initial linear programming relaxation.

Example: consider the following PILP problem:

$$\begin{array}{llrcrcl} \min & 3x & + & 2y & & \\ \text{s.t.} & 2x & + & y & \geq & 1 \\ & x & + & 2y & \geq & 4 \\ & x & , & y & \in & \mathbb{Z}. \end{array}$$

An optimal solution to the linear programming relaxation is given by

$$[x^+, y^+]^\top = \frac{1}{3}[-2, 7]^\top.$$

Note that adding the inequalities $2x + y \geq 1$ and $x + 2y \geq 4$ yields $3x + 3y \geq 5$, or equivalently,

$$x + y \geq \frac{5}{3}.$$

Since $x + y$ is an integer for every feasible solution $[x, y]^\top$, $x + y \geq 2$ is a valid inequality for the original problem, but is violated by $[x^+, y^+]^\top$. Hence, $x + y \geq 2$ is a cutting plane.

Adding this to the linear programming relaxation, we have

$$\begin{array}{llrcrcl} \min & 3x & + & 2y & & \\ \text{s.t.} & 2x & + & y & \geq & 1 \\ & x & + & 2y & \geq & 4 \\ & x & + & y & \geq & 2. \end{array}$$

which, upon solving, yields $[x^*, y^*]^\top = [-1, 3]^\top$ as an optimal solution.

Since all the entries are integers, this is also an optimal solution to the original problem. In this example, adding a single cutting plane solved the problem. In practice, one often needs to add numerous cutting planes and then continue with branch-and-bound to solve nontrivial MILP problems.

Many methods for generating cutting planes exist – the problem of generating effective cutting planes efficiently is still an active area of research [4].

5.6 Useful Modeling Techniques

So far, we have discussed the kinds of optimization problems that can be solved and certain methods available for solving them. Practical success, however, depends upon the effective **translation** and **formulation** of a

problem description into a mathematical programming problem, which often turns out to be as much an art as it is a science.

We will not be discussing formulation techniques in detail (see [7] for a deep dive into the topic) – instead, we highlight modeling techniques that often arise in business applications, which our examples have not covered so far.

5.6.1 Activation

Sometimes, we may want to set a binary variable y to 1 whenever some other variable x is positive. Assuming that x is bounded above by M, the inequality

$$x \leq My$$

will model the condition. Note that if there is no valid upper bound on x, the condition cannot be modeled using a linear constraint.

5.6.2 Disjunction

Sometimes, we want $\mathbf{x}$ to satisfy at least one of a list of inequalities; that is,

$$\mathbf{a}^{(1)^\mathbf{T}}\mathbf{x} \geq b_1 \vee \mathbf{a}^{(2)^\mathbf{T}}\mathbf{x} \geq b_2 \vee \cdots \vee \mathbf{a}^{(k)^\mathbf{T}}\mathbf{x} \geq b_k.$$

To formulate such a **disjunction** using linear constraints, we assume that, for $i = 1, \ldots, k$, there is a lower bound M_i on $\mathbf{a}^{i^\mathbf{T}}\mathbf{x}$ for all $\mathbf{x} \in \mathcal{D}$. Note that such bounds automatically exist when $\mathcal{D}$ is a bounded set, which is often the case in applications.

The disjunction can now be formulated as the following system where y_i is a new 0-1 variable for $i = 1, \ldots, k$:

$$\left|\begin{array}{l} \mathbf{a}^{(1)^\mathbf{T}}\mathbf{x} \geq b_1 y_1 + M_1(1 - y_1) \\ \mathbf{a}^{(2)^\mathbf{T}}\mathbf{x} \geq b_2 y_2 + M_2(1 - y_2) \\ \qquad \vdots \\ \mathbf{a}^{(k)^\mathbf{T}}\mathbf{x} \geq b_k y_k + M_k(1 - y_k) \\ y_1 + \cdots + y_k \geq 1. \end{array}\right.$$

Note that $\mathbf{a}^{i^\mathbf{T}}\mathbf{x} \geq b_i y_i + M_i(1 - y_i)$ reduces to $\mathbf{a}^{i^\mathbf{T}}\mathbf{x} \geq b_i$ when $y_i = 1$, and to $\mathbf{a}^{i^\mathbf{T}}\mathbf{x} \geq M_i$ when $y_i = 0$, which holds for all $\mathbf{x} \in \mathcal{D}$.

Therefore, y_i is an activation for the i^{th} constraint, and at least one is activated because of the constraint

$$y_1 + \cdots + y_k \geq 1.$$

5.6.3 Soft Constraints

Sometimes, we may be willing to pay a price in exchange for specific constraints to be violated (perhaps they represent "nice-to-have" conditions instead of "must-be-met" conditions). Such constraints are referred to as **soft constraints**.

There are situations in which having soft constraints is advisable, say when enforcing all constraints results into an infeasible problem, but a solution is nonetheless needed.

We illustrate the idea on a modified BKP. As usual, there are n items and item i has weight w_i and value $v_i > 0$ for $i = 1, \ldots, n$. The capacity of the knapsack is denoted by K. Suppose that we prefer not to take more than N items, but that the **preference** is not an actual constraint.

We assign a penalty for its violation and use the following formulation:

$$\begin{array}{lll} \max & \sum_{i=1}^{n} v_i x_i - py & \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \leq K & \\ & \sum_{i=1}^{n} x_i - y \leq N & \\ & x_i \in \{0,1\} & i = 1, \ldots, n \\ & y \geq 0. & \end{array}$$

Here, p is a non-negative number of our choosing. As we are maximizing

$$\sum_{i=1}^{n} v_i x_i - py,$$

y is pushed towards 0 when p is "large". Therefore, the problem will be biased towards solutions that try to violate $x_1 + \cdots + x_n \leq N$ as little as possible.

Experimentation is required to determine What value to select for p; the general rule is that if violation is costly in practice, we should set p to be (relatively) high; otherwise, we set it to a moderate value relative to the coefficients of the variables in the objective function value.

Note that when $p = 0$, the constraint $x_1 + \cdots + x_n \leq N$ has no effect because y can take on any positive value without incurring a penalty.

5.7 Software Solvers

A wide variety of solvers exist for all kinds of optimization problems. The NEOS Server is a free online service that hosts many solvers and is a great resource for experimenting with different solvers on **small** problems.

For **large** or **computationally challenging** problems, it is advisable to use a solver installed on a dedicated private machine/server. Commercial solvers can also prove useful:

- IBM ILOG Cplex ;
- Gurobi , or
- FICO Xpress Optimization .

There are popular open-source solvers as well, although they are not as powerful as the commercial tools:

- CBC ;

- GLPLK
- SCIP (requires a commercial licence for consulting work);
- JuliaOpt, to name a few.

We mention in passing that learning how to use of any of these solvers effectively requires a significant time investment. In addition, it is common to build optimization models using a modeling system such as GAMS and LINDO, or a modeling language such as AMPL, ZIMPL, or JuMP.

Note that in the data science and machine learning context, more straightforward methods like **gradient descent**, **stochastic gradient descent** and **Newton's method** are usually sufficient for most applications.

5.8 Data Envelopment Analysis

Operations research (OR) is a mish-mash of various mathematical methods used to solve complex industrial problems, especially optimization problems, which are being tackled in management and other non-industrial contexts.

Data Envelopment Analysis (DEA), based on linear programming, is used to measure the relative performance of units in an organization such as a government department, a school, a company, etc. Typically, a unit's **efficiency** is defined as the quotient of its **outputs**[7] by its **inputs**.[8]

7: Activities of the organization such as service levels or number of deliveries.

8: The resources supporting the organization's operations, such as wages or value of the in-store stock.

In an organization with only one type of input and one type of output, the comparison is simple. For instance, a fictional organization could have the simple input/out data in the table below:

Unit	Input	Output	Efficiency
A	10	10	100%
B	10	20	200%
C	5	15	300%
D	15	10	67%

However, if there are more than one input or output, the comparisons are less obvious: in the table below, is unit A more efficient than unit B?

Unit	Input 1	Input 2	Output 1	Output 2
A	10	5	10	20
B	10	15	20	5
C	5	15	15	15
D	15	5	10	20

Unit A has fewer total inputs than unit B (as well as fewer outputs of type 1, but it has a substantially more outputs of type 2. Without a system in place to measure relative efficiency, comparison between (potentially incommensurate) units is unlikely to be fruitful.

The **relative efficiency** of unit k is defined by

$$\mathrm{RE}_k = \frac{\sum_j w_{k,j} O_{k,j}}{\sum_i v_{k,i} I_{k,i}},$$

where

- $\{O_{k,j} \mid j = 1, \ldots, n\}$ represent the n **outputs** from unit k,
- $\{I_{k,i} \mid i = 1, \ldots, m\}$ represent the m inputs from unit k,
- $\{w_{k,j} \mid j = 1, \ldots, n\}$ and $\{v_{k,i} \mid i = 1, \ldots, m\}$ are the **associated unit weights**.

For a specific unit k, the DEA model maximizes the **weighted sum of outputs** for a **fixed weighted sum of inputs** (usually set to 100), subject to the weighted sum of outputs of every unit being at most equal to the weighted sum of its inputs when using the DEA weights of unit k.

In other words, the optimal set of weights for a given unit could not give another unit a relative efficiency greater than 1.

This is equivalent to solving the following linear program for each unit k_0:

$$\left|\begin{array}{ll} \max & \sum_{j=1}^{n} w_{k_0,j} O_{k_0,j} \\ \text{s.t.} & \sum_{i=1}^{m} v_{k_0,i} I_{k_0,i} = 100 \\ & \sum_{j=1}^{n} w_{k_0,j} O_{\ell,j} - \sum_{i=1}^{m} v_{k_0,i} I_{\ell,i} \leq 0, \quad 1 \leq \ell \leq K \\ & (w_{k_0,j}, v_{k_0,i}) \geq \boldsymbol{\varepsilon}, \quad 1 \leq j \leq n,\ 1 \leq i \leq m \end{array}\right.$$

where $\boldsymbol{\varepsilon} \geq 0$ is a parameter vector to be modified by the user.

If we define $\mathbf{w}_\ell$, $\mathbf{v}_\ell$, $\mathbf{O}_\ell$ and $\mathbf{I}_\ell$ as the vectors of output weights, input weights, outputs and inputs, respectively, for unit ℓ, while $\mathbf{O}$ and $\mathbf{I}$ represent the row matrix of outputs and the row matrix of inputs for all the units, then the linear problem can be re-written simply as

$$\left|\begin{array}{ll} \max & \mathbf{w}_{k_0}^\top \mathbf{O}_{k_0} \\ \text{s.t.} & \mathbf{v}_{k_0}^\top \mathbf{I}_{k_0} = 100 \\ & \mathbf{w}_{k_0}^\top \mathbf{O} - \mathbf{v}_{k_0}^\top \mathbf{I} \leq \mathbf{0} \\ & -(\mathbf{w}_{k_0}, \mathbf{v}_{k_0}) \leq -\boldsymbol{\varepsilon} \end{array}\right.$$

This problem can be solved by the method of **Lagrange multipliers** (see Section 2.5.5) or by using dedicated **numerical solvers** (see previous Section 5.7).

With the data from the example above,the DEA program for unit A, for instance, becomes

$$\left|\begin{array}{lrcrcrcrcl} \max & 10w_{A,1} & + & 20w_{A,2} & & & & & & \\ \text{s.t.} & & & & & 10v_{A,1} & + & 5v_{A,2} & = & 100 \\ & 10w_{A,1} & + & 20w_{A,2} & - & 10v_{A,1} & - & 5w_{A,2} & \leq & 0 \\ & 20w_{A,1} & + & 5w_{A,2} & - & 10v_{A,1} & - & 15w_{A,2} & \leq & 0 \\ & 15w_{A,1} & + & 15w_{A,2} & - & 5v_{A,1} & - & 15w_{A,2} & \leq & 0 \\ & 10w_{A,1} & + & 20w_{A,2} & - & 15v_{A,1} & - & 5w_{A,2} & \leq & 0 \\ & w_{A,1} & , & w_{A,2} & , & v_{A,1} & , & v_{A,2} & \geq & \varepsilon \end{array}\right.$$

5.8.1 Challenges and Pitfalls

By allowing non-universal (unit-specific) weights, DEA allows each unit to present itself in the **best possible light**, which could potentially lead most units to be deemed efficient. This issue is mitigated to some extent when the number of units K is greater than the product of the number of outputs by the number of inputs $n \cdot m$.

When the number of units is small, a lack of differentiation among units is uninformative since all units could benefit from the best-case scenario described above. When there is differentiation, however, it can be quite telling: units with low DEA relative efficiency have achieved a low score **even when given a chance to put their best foot forward**.

Another concern is that a unit could artificially seem efficient by completely eliminating unfavourable outputs or inputs (i.e.if the associated input/output weights are 0). Constraining the weights to take values in some fixed range can help avoid this issue.

In the example that was discussed above, when we set $\varepsilon = 0$, all units have a relative efficiency of 100. If we set $\varepsilon = 2$, however, the relative efficiency for each unit is

$$\text{RE}_A = 100, \quad \text{RE}_B = 67.7, \quad \text{RE}_C = 100, \quad \text{and} \quad \text{RE}_D = 90.$$

Evidently, insisting that **all** the factors be considered may affect the results.

External factors can easily be added to the model as either inputs or outputs. Available resources are classified as inputs; activity levels or performance measures are classified as outputs.

When units can also be assessed according to some other measure (such as profitability, average rate of success for a task, or environmental cleanliness, say), it can be tempting to solely use the second metric to rank the units.

The combination of **efficiency** and **profitability** (or of any two measures, really) can however offer insights and suggestions:

Flagships are units who score high on both measures and that can provide examples of good operating practices (as long as it is recognized that they are also likely beneficiaries of favourable conditions).

Sleepers score low on efficiency but high on the other measure, which is probably more a consequence of favourable conditions than good management; as such, they become candidates for efficiency drives.

Dogs score high on efficiency but low on the other measure, which indicates good management but unfavourable conditions. In extreme case, these units are candidates for closures, their staff members could be re-assigned to other units.

Question Marks are units who score low on both measures; they are subject to unfavourable conditions, but this could also be a consequence of bad management. Attempts should be made to increase the efficiency of these units so that they become Sleepers or Flagships.

Finally, note that in any reasonable application, the linear program to be solved (or its dual) can be fairly **complicated** and sophisticated software can be required to obtain a solution. That is emblematic of industrial optimization problems.

5.8.2 Advantages and Disadvantages

The main **benefits** of DEAs are that:

- there is no need to explicitly specify a mathematical form for the production function;
- they have been proven to be useful in uncovering relationships that remain hidden from other methodologies;
- they are capable of handling multiple inputs and outputs;
- they can be used with any input-output measurements, and
- the sources of inefficiency can be analysed and quantified for every evaluated unit.

On the other hand, there are also **disadvantages** to using DEAs:

- the results are known to be sensitive to the selection of inputs and outputs;
- it is impossible to test for the best specification, and
- the number of efficient units on the frontier tends to increase with the number of inputs and output variables.

As is the case for all applications of quantitative methods to real-world problems, DEAs will ultimately prove useless **unless users understand how they function and how to interpret their results**.

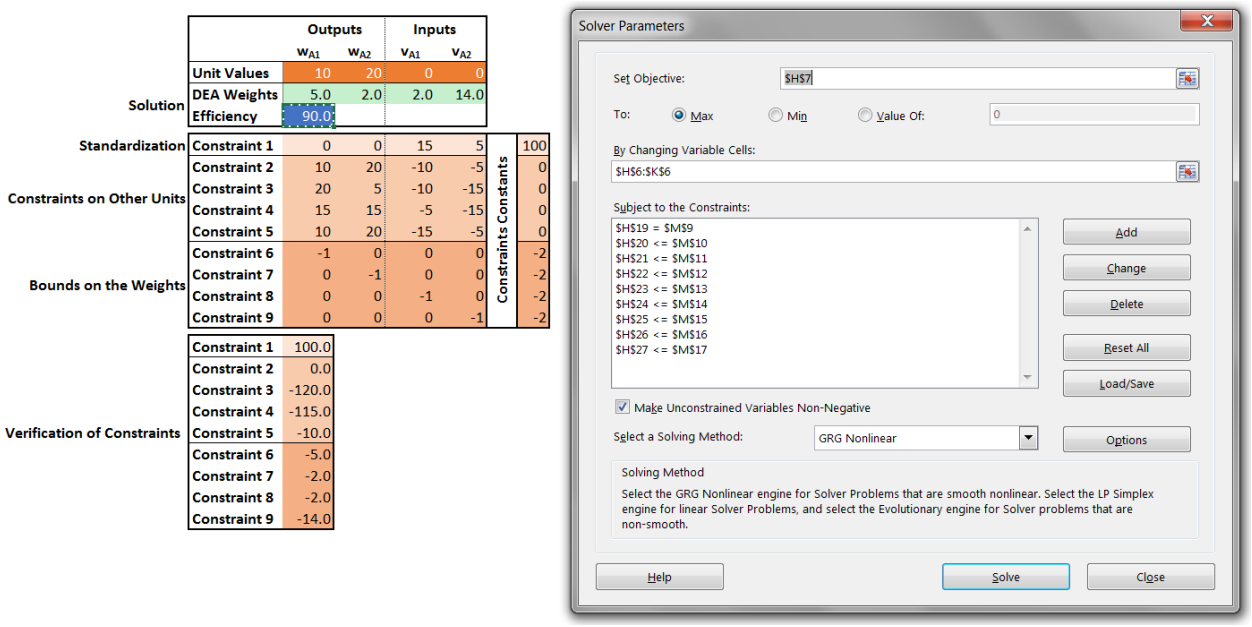
5.8.3 SAS, Excel, and R DEA Solvers

For small problems, the numerical cost of solving the problem is not too onerous. Consequently, such problems can typically be solved without having to purchase a commercial solver.

As an illustration, consider the problem of finding the relative efficiency of unit D in the example arising from the data presented above (using a minimal weight threshold of $\varepsilon = 2$, say). Thus, we are looking for the solution to

$$
\begin{array}{lrcrcrcrcl}
\max & 10w_{D,1} & + & 20w_{D,2} & & & & & & \\
\text{s.t.} & & & & & 15v_{D,1} & + & 5v_{D,2} & = & 100 \\
& 10w_{D,1} & + & 20w_{D,2} & - & 10v_{D,1} & - & 5w_{D,2} & \leq & 0 \\
& 20w_{D,1} & + & 5w_{D,2} & - & 10v_{D,1} & - & 15w_{D,2} & \leq & 0 \\
& 15w_{D,1} & + & 15w_{D,2} & - & 5v_{D,1} & - & 15w_{D,2} & \leq & 0 \\
& 10w_{D,1} & + & 20w_{D,2} & - & 15v_{D,1} & - & 5w_{D,2} & \leq & 0 \\
& w_{D,1} & , & w_{D,2} & , & v_{D,1} & , & v_{D,2} & \geq & 2
\end{array}
$$

This is a small problem, and Excel's numerical solver can be used to yield a relative efficiency of 90% (see Figure 5.2 for an illustration).



		Outputs		Inputs	
		w_{A1}	w_{A2}	v_{A1}	v_{A2}
Solution	Unit Values	10	20	0	0
	DEA Weights	5.0	2.0	2.0	14.0
	Efficiency	90.0			
Standardization	Constraint 1	0	0	15	5
Constraints on Other Units	Constraint 2	10	20	-10	-5
	Constraint 3	20	5	-10	-15
	Constraint 4	15	15	-5	-15
	Constraint 5	10	20	-15	-5
Bounds on the Weights	Constraint 6	-1	0	0	0
	Constraint 7	0	-1	0	0
	Constraint 8	0	0	-1	0
	Constraint 9	0	0	0	-1

Constraints Constants
100
0
0
0
0
-2
-2
-2
-2

Verification of Constraints	
Constraint 1	100.0
Constraint 2	0.0
Constraint 3	-120.0
Constraint 4	-115.0
Constraint 5	-10.0
Constraint 6	-5.0
Constraint 7	-2.0
Constraint 8	-2.0
Constraint 9	-14.0

Figure 5.2: Excel's numerical solver for unit D in the simple DEA problem.

There are a number of non-technical issues with the solver, including the fact that a different worksheet has to be created for every single unit. With larger datasets, this approach may not be practical.

SAS's `proc optmodel`, available in version 9.2+ as part of the OR(R) suite, can also be used; but some additional work has to be done to automate the descriptions of the programs to be solved. R's `rDEA` and `deaR` packages provide other options.

5.8.4 Case Study: Barcelona Schools

In this section, we present an illustration of a **resource utlization model** which uses a DEA-like approach.[9]

9: Other optimization case studies can be found in [3].

- **Title:** On centralized resource utilization and its re-allocation by using DEA [6]
- **Authors:** Cecilio Mar-Molinero, Diego Prior, Maria-Manuela Segovia, Fabiola Portillo
- **Date:** 2012
- **Methods:** Data envelopment analysis, simulations

Abstract The standard DEA model allows different **Decision-Making Units** (DMUs) to set their own priorities for the inputs and outputs that form part of the efficiency assessment. In the case of a centralized organization with many outlets, such as an education authority that is responsible for many schools, it may be more sensible to operate in the most efficient way, but under a common set of priorities for all DMUs. The centralized resource allocation model does just this; the optimal resource reallocation is found for Spanish public schools and it is shown that the most desirable operating unit is a by-product of the estimation.

Data The data consists of 54 secondary public schools in Barcelona during the year 2008, each with three **discretionary inputs** (teaching hours per week, x_1; specialized teaching hours per week, x_2; capital investments in the last decade, x_3), one **non-discretionary input** (total number of students present at the beginning of the academic year, X) and two **outputs** (number of students passing their final assessment, y_1, and number of students continuing their studies at the end of the academic year, y_2).

A subset of the data is shown in Table 5.5.

School #	y_1	y_2	x_{1d}	x_{2d}	x_{3d}	X_{1nd}
1	260.65	378.00	44.00	3.00	8	384.00
2	195.18	213.00	32.01	3.00	18	225.00
3	242.75	429.70	56.98	4.00	84	446.00
4	283.02	350.00	49.50	3.00	39	356.00
5	376.76	650.80	77.50	5.50	61	657.00
6	252.19	429.00	49.40	2.00	56	440.00
7	225.50	247.34	33.15	1.50	43	248.00
8	363.85	364.34	45.90	2.00	36	381.00
9	261.87	272.00	44.37	2.00	24	288.00
10	235.40	251.00	35.49	1.50	51	259.00
11	198.63	223.34	42.00	1.50	46	227.00
12	159.78	248.00	36.96	2.00	2	250.00
13	98.09	193.00	35.20	1.50	55	203.00
14	214.92	219.00	33.60	1.50	32	229.00
15	136.07	269.20	33.80	1.50	54	271.00
16	214.68	346.00	54.39	2.00	33	347.00
17	117.12	196.00	29.00	1.50	7	212.00
18	261.89	334.00	42.40	2.00	47	339.00

Table 5.5: Sample from the Barcelona public school dataset used with the radial and simplified models.

Challenges A first challenge is that the machinery of DEA cannot directly be brought to bear on the problem since the models under consideration are at best DEA-like. Another challenge is that the number of unknowns to be estimated in the original model is quadratic in the number of units. Consequently, the original model must be simplified to avoid difficulties when the number of units is large. Fortunately, the proposed simplifications can be interpreted logically in the context of re-allocation of resources.

Finally, there are situations where a solution to the simplified problem can be obtained even when the constraints on the total number of units is relaxed, allowing for the possibility of reaching the similar output levels with fewer inputs, in effect advocating for the closure of some units.

While this is a technically-correct solution, it might could prove to be an **unadvisable** one for a variety of non-technical reasons: closing schools is not usually a politically and/or societally palatable strategy. This latter factor should also be incorporated in the decision-making process.

Project Summary and Results In the standard DEA model, each unit sets its own priorities, and is evaluated using unit-specific weights. In a **de-centralized environment**, the standard approach is reasonable, but under a central authority where a common set of priorities needs to be met by all units (such as the branches of a bank, or recycling collection vehicles in a city), that approach needs to be modified.

In a school setting, school board administrators may wish to evaluate teachers in a similar manner independently of the school at which they work. Centralized assessment imposes a common set of weights. For weakly centralized management, it is a further assumption that any input excess of inefficient units can be re-allocated among the efficient units, but only as long as this does not contravene the built-in inflexibility of the system, which may make re-allocation rather difficult.

Strongly centralized management, on the other hand, allow for re-allocation of the majority of inputs and outputs among all the units (inefficient or efficient) with the aim of optimizing the performance of the entire system. The original **radial** model of Lozano and Villa [5] is not, strictly speaking, a data envelopment model:

$$\min \theta \text{ (objective)}$$

$$\text{s.t. } \sum_{r=1}^{54}\sum_{j=1}^{54} \lambda_{j,r} x_{i,j} - \theta \sum_{j=1}^{54} x_{i,j} \le 0, \quad \text{for } i = 1,2,3$$

(discretionary inputs)

$$\sum_{r=1}^{54}\sum_{j=1}^{54} \lambda_{j,r} X_j - \sum_{j=1}^{54} X_j \le 0,$$

(non-discretionary input)

$$\sum_{r=1}^{54} y_{kr} - \sum_{r=1}^{54}\sum_{j=1}^{54} \lambda_{j,r} y_{k,j} \le 0, \quad \text{for } k = 1,2$$

(outputs)

$$\sum_{j=1}^{54} \lambda_{j,r} = 54, \quad \text{for } r = 1,\ldots,54$$

$$-\lambda_{j,r} \le 0, \quad \text{for } j,r = 1,\ldots,54, \quad \theta \text{ free}$$

Indeed, this model is not asking every unit to select the weights that make it look as good as possible when comparing itself to the remaining units under the same assessment; rather, it is asking for the system as a whole to find the weights that present it in the best possible light possible, then it assesses the performance of the units separately, using the optimal system weights.

This conceptual shift leads to proposed closures.The main drawback of the radial model is the large number of weights to estimate. A simplification is proposed: if some of the units can be cloned, or equivalently, if some of the units can be closed and their resources re-allocated to other units, then the radial model becomes substantially simpler, and the number of weights to estimate is linear in the number of units (as opposed to quadratic).

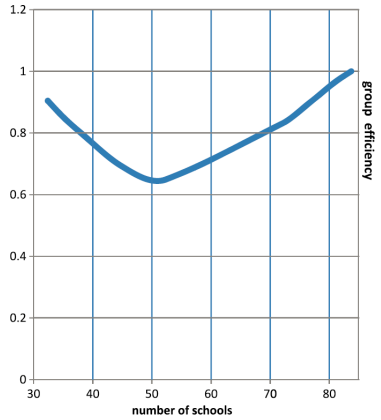


Figure 5.3: Results of the re-allocation process in the Barcelona public school dataset.

The new problem is DEA-like:

$$\begin{aligned}
&\min \theta \text{ (objective)}\\
&\text{s.t. } \sum_{j=1}^{54} \lambda_j x_{i,j} - \theta \sum_{j=1}^{54} x_{i,j} \leq 0, \quad \text{for } i = 1,2,3\\
&\text{(discretionary inputs)}\\
&\sum_{j=1}^{54} \lambda_j X_j - \sum_{j=1}^{54} X_j \leq 0\\
&\text{(non-discretionary inputs)}\\
&\sum_{r=1}^{54} y_k - \sum_{j=1}^{54} \lambda_j y_{k,j} \leq 0, \quad \text{for } k = 1,2\\
&\text{(outputs)}\\
&\sum_{j=1}^{54} \lambda_j = 54\\
&-\lambda_j \leq 0, \quad \text{for } j = 1,\ldots,54, \quad \theta \text{ free}
\end{aligned}$$

The numerical solution to the radial model shows a group efficiency of 66%, meaning that the outputs of the system could be produced while reducing the discretionary inputs by $\theta = 34\%$. The simplified model reaches the same group efficiency by cloning units 25 (24.26 times), 26 (20.02 times), 36 (4.71 times), 17 (2.69 times), and 44 (1.70 times).

The re-allocation of inputs and outputs among the 54 schools would produce the aforementioned reduction of the 34% in discretionary inputs.

A simulation experiment shows the effect of dropping the constraint on the number of units: the group efficiency obtained by solving the simplified system for various values of n from 32 to 81 is seen in Figure 5.3.

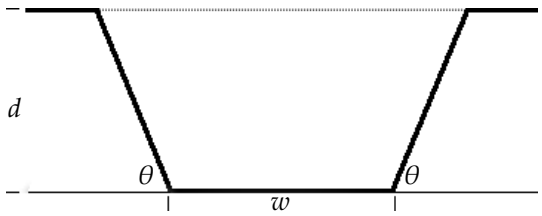
Sure enough, the original solution is good, appearing near the minimum, which reaches $\theta = 0.64$ at $n = 50.36$. This group efficiency corresponds to cloning units 25 (23.96 times), 26 (17.62 times), and 29 (7.87 times). Obviously, schools (and their resources) cannot be cloned, so what are we to make of this result?[10]

10: It could be argued that unit 25 and 26, for instance, are ideal schools under the common priorities imposed by the system: should new schools have to be built, attempts could be made to emulate the stars. Of course, in practice, other factors could come into play.

5.9 Exercises

Some of the questions in this section may need to be solved by a combination of the techniques provided in Chapters 2, 4, and 5.

1. Find the extrema of the function defined by $f(x) = x - \sin(x)$ over the interval $[-2, 12]$.
2. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by $f(x, y) = A - (x^2 + Bx + y^2 + Cy)$, where A, B, C are constants. What values must they take so that f admits a maximum value of 15 when $(x, y) = (-2, 1)$? What if it is a minimal value of 15 when $(x, y) = (-2, 1)$?
3. Consider a factory that produces various types of deluxe pickle jars. The monthly number of jars Q of a specific kind of pickled radish that can be produced at the factory is given by $Q(K, L) = 1000K^{0.21}L^{0.79}$, where K is the number of dedicated canning machines, and L is the monthly number of employee-hours spent on the pickled radish. The pay rate for the employees is 22\$/hour; the monthly maintenance cost for each canning machine is 300\$. If the factory owners want to maintain monthly production at 40,000 jars of pickled radish, what combination of number of canning machines and employee-hour will minimize the total production costs?
4. The distance d at which a projectile can be fired depends on the temperature t and the humidity level h, according to
$$E(t, h) = 12,000 - t^2 - 2ht - 2h^2 + 200t + 260h,$$
where t is measured in °F and $0 \leq h \leq 100$. Under what atmospheric conditions should we fire the projectile to maximize the distance it travels? To minimize it?
5. The area of the vertical sections of an irrigation canal is 50 square feet. The average flow of liquid in the canal is inversely proportional to the perimeter of the trapezoid, excluding the length length of the dotted segment, which we will denote by p. In order to maximise the flow, we must then minimize p. Determine the depth d, base w and angle θ that maximizes the flow.



6. Find the extrema of $f(x, y) = x^2 - y$, subject to $x^2 - y^2 = 1$.
7. Find the extrema of $f(x, y, z) = 2\sqrt{x} + y + 4 \ln z$ subject to $x^2 + y + z^2 = 16$.
8. Solve the linear program: $\arg\min\{0.5x_1 + x_2 \mid x_1 + x_2 \geq 1, x_1 + 0.5x_2 \geq 1, x_1, x_2 \geq 0\}$.

Chapter References

[1] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[2] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. 1st. Athena Scientific, 1997.

[3] P. Boily and J. Schellinck. *Introduction to Quantitative Consulting*. Quadrangle/Data Action Lab, 2025.

[4] G. Cornuéjols. 'Valid inequalities for mixed integer linear programs'. In: *Math. Program.* 112.1 (2008), pp. 3–44.

[5] S. Lozano and G. Villa. 'Centralized Resource Allocation Using Data Envelopment Analysis'. In: *Journal of Productivity Analysis* 22.1 (July 2004), pp. 143–161.

[6] C. Mar-Molinero et al. 'On centralized resource utilization and its reallocation by using DEA'. In: *Ann. Oper. Res.* 221.1 (2014), pp. 273–283.

[7] H. P. Williams. 'Model Building in Linear and Integer Programming'. In: *Computational Mathematical Programming*. Ed. by Klaus Schittkowski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 25–53.