1 Introduction

An *optimization* problem (or model) P can be generally expressed as:

$$\begin{aligned} \min(\text{or}\max) f(x) \\ S \\ x \in D \end{aligned} \tag{1}$$

where $f(x)$ is a real-valued function on variables x, D is the domain of x, and S is a finite set of constraints. In general, x is a tuple $(x_1, \ldots, x_n)$, D is the cartesian product $D_1 \times \cdots \times D_n$, and it holds that $x_j \in D_j$. Formally, a constraint $c \in S$ is a function associated to a subset x_c (the support, or scope, of c) of the variables and whose value can be either true (the constraint is satisfied) or false (the constraint is violated). In *mathematical* optimization, we usually assume that each individual domain D_j is numeric, i.e., a subset of $\mathbb{R}$, and constraints can be expressed in algebraic form as inequalities $g_i(x) \leq b_i$, for $i \in 1, \ldots, m$.

There are other optimization paradigms that do not make this assumption: for example, in constraint programming, *a variable can have as domain the letters of some alphabet, and constraints do not need to have an algebraic form. We will not treat those approaches further in this course.*

Any $x \in D$ is called a *solution* of P. A solution that satisfies all constraints in S is called a *feasible solution*. We denote the set of feasible solutions to an optimization problem P as $F(P)$.

If the domain D is discrete, we talk about *discrete optimization*. In addition, if D is finite, meaning that the set of solutions is finite, we talk about *combinatorial optimization*.

A feasible solution x^* is *optimal* if

$$f(x^*) \leq f(x) \quad \forall x \in F(P) \tag{2}$$

Note that the optimal solution is not necessarily unique.

An optimization problem is *infeasible* if it has no feasible solution, i.e., if $F(P) = \emptyset$. It is called *unbounded* if there is no lower limit on $f(x)$ for $x \in F(P)$. In the following, we will almost always assume that an optimization problem is always infeasible, unbounded or with finite optimum.

This excludes seemingly innocuous problems like, e.g., $\min e^{-x}, x \geq 0$.

An optimization problem is solved if one of its optimal solutions is found *and* proven to be optimal, or when it is proven to be infeasible or unbounded.

1.1 Optimization Paradigms

A mathematical optimization model provides an abstraction of an optimization problem in the real world that we might be interested in solving: the variables x encode the decisions that need to be made, the constraints in S encode, as the name suggests, constraints that need to be collectively satisfied, and the objective function represents the cost of those decisions.

Many practical problems in decision making, as well as more theoretical questions, can be casted and formulated as mathematical optimization problems, so that mathematical optimization has become a key tool in many domains.

Or mathematical programming, *as it used to be called.*

Note that the purpose of mathematical optimization is two-fold: on the one hand, it provides a *language* that people can use to describe optimization problems in a mathematically sound and unambiguous way (the mathematical *model*). This language should, ideally, be very expressive, in the sense that it allows people to faithfully describe many different real world problems with ease. On the other hand, mathematical optimization is also in charge of devising and analyzing solution methods (*algorithms*) to solve those problems once they have been modeled.

Unfortunately, the two goals are conflicting: if the language is too generic, and thus with wider applicability, then the corresponding class of optimization problems might not admit an efficient solution algorithm, and thus the model would be utterly useless. At the same time, if the language is too restrictive, it might admit very efficient algorithms, but those can be applied in very few real-world cases, again not very useful.

To provide some context, an optimization problem as (1) is, in general, intractable, in the sense that there exists no algorithms (at all) for solving it.

So, we need to find the right compromise between applicability and solution efficiency. Here are some examples.

1.1.1 *Linear Programming (LP)*

A linear program consists in the minimization of a linear function subject to a finite list of linear constraints. In general we have the form:

$$\begin{aligned} &\min c^\top x \\ &\quad a_i^\top x \sim b_i \quad i = 1, \dots, m \\ &\quad l_j \le x_j \le u_j \quad j = 1, \dots, n \end{aligned} \tag{3}$$

where $\sim \in \{\le, \ge, =\}$, $l_j \in \mathbb{R} \cup \{-\infty\}$, and $u_j \in \mathbb{R} \cup \{+\infty\}$. The domain of a single variable is thus an interval in $\mathbb{R}$.

The language of linear programming is admittedly very restrictive: not many problems can be formulated as linear programs. The true power of linear programming stems from the fact that not only it can be solved very efficiently, both in theory and in practice, but it provides a theoretical and algorithmic foundation for building state of the art solution methods for wider (and more interesting) classes of problems, using LP as a black box.

1.1.2 *Integer Linear Programming (MIP)*

One of the fundamental restrictions of linear programming is its inability to model discrete decisions. This is remedied by integer linear programming, where objective function and constraints are still restricted to be linear, but we are allowed to require that some variables can only take an integer value within their interval. In general we have the form:

$$\begin{aligned} &\min c^\top x \\ &\quad a_i^\top x \sim b_i \quad i = 1, \dots, m \\ &\quad l_j \le x_j \le u_j \quad j = 1, \dots, n \\ &\quad x_j \in \mathbb{Z} \quad \forall j \in J \subseteq N = \{1, \dots, n\} \end{aligned} \tag{4}$$

If $J = N$ we have a pure integer program, otherwise we have a mixed integer linear program. Of course, if $J = \emptyset$ we are back to linear programming.

The integrality constraint, while a seemly small change, has far reaching consequences, both in applicability and in computational complexity: the paradigm has a much wider applicability, but it is no longer solvable in polynomial time. LP and MIP are, nowadays, a mature technology: they are routinely used in applications to solve optimization problems in a wide range of domains.

To the contrary, it makes the problem NP-hard.

1.1.3 *Convex Optimization*

The other direction in which we can extend linear programming is, obviously, nonlinearity. We must be cautious in this direction, though, as allowing arbitrary nonlinearities in the objective (and/or constraints), still gives an intractable problem. It turns out, however, that if we restrict to *convex* functions and constraints, then we can recover most (but not all) of the nice properties of LP, while still giving a very meaningful extension to the paradigm.

In general we have the form:

$$\begin{aligned} \min\ & f(x) \\ & g_i(x) \le b_i \quad i = 1, \dots, m \\ & l_j \le x_j \le u_j \quad j = 1, \dots, n \end{aligned} \tag{5}$$

where $f(x)$ and all $g_i(x)$ are required to be convex functions.

We will formality define convexity in the next chapter.

The power of convex optimization comes from two sides:

- it is a non trivial extension of LP, but it can still be solved quite efficiently, both in theory and in practice;
- the algorithms developed for convex optimization can be applied, to some extent and with weaker guarantees, also to the nonconvex case. In particular, modern ML has proved to be a very important application of convex optimization techniques.

Note that we can combine the two extensions (convexity and integrality), and obtain what is called mixed integer convex programming (or convex mixed integer nonlinear programming).

1.2 Examples

In the following, we will show a few simple examples of each individual paradigm. We will also conclude with an application in control systems theory that showcases the flexibility of mathematical optimization.

1.2.1 *Diet problem*

A farmer wants to determine the minimum cost diet for their animals, with the constraint that certain minimal nutritional requirements are met. There are n foods to choose from on the market, each with a unit cost c_j. The nutritional requirements consider m basic nutrients, each with a minimum quantity b_i. We finally denote with a_{ij} the amount of nutrient i in a unit of food j. Using variables x_j to encode the amount of food j in the diet, the problem can be formulated as:

$$\begin{aligned}
\min \sum_{j=1}^{n} c_j x_j & \\
\sum_{j=1}^{n} a_{ij} x_j \geq b_i & \qquad i = 1, \dots, m \\
x_j \geq 0 & \qquad j = 1, \dots, n
\end{aligned}$$

As such, the diet problem is naturally a linear program.

1.2.2 *Knapsack problem*

Let a set of n items be given, each with profit p_j and weight w_j. We also have a container (the nominal *knapsack*) with capacity B. We want to find a subset of the items of maximum total profit that does not exceed the capacity of the container. Using *binary* variables x_j with the following meaning:

We will see that binary variables are very common in MIP modeling.

$$x_j = \begin{cases} 1 & \text{if item } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

the problem can be written as:

$$\begin{aligned}
\max \sum_{j=1}^{n} p_j x_j & \\
\sum_{j=1}^{n} w_j x_j \leq B & \\
x_j \in \{0, 1\} & \qquad j = 1, \dots, n
\end{aligned}$$

As such, the knapsack problem is a (binary) integer linear program.

1.2.3 *Portfolio optimization*

Suppose we have a portfolio of n different assets. For each of them, we have the expected return μ_j, usually obtained by averaging some historical data. The risk of the investment is encoded by the covariance matrix R of our assets, again obtained from some historical analysis. We want to optimize our portfolio, i.e., decide which percentage of our budget to allocate to each asset, in order to obtain at least a target average return ρ, at the minimum risk. Using continuous variables x_j to encode the percentage of our budget to allocate to asset j, we get the model:

$$\begin{aligned}
\min x^\top R x & \\
\sum_{j=1}^{n} \mu_j x_j \geq \rho & \\
\sum_{j=1}^{n} x_j = 1 & \\
0 \leq x_j \leq 1 & \qquad j = 1, \dots, n
\end{aligned}$$

This is a convex optimization problem, as the covariance matrix is positive semidefinite by construction.

1.2.4 *LQR*

We have a discrete-time linear system described by state equations:

We are assuming, for simplicity, that the matrices A, B are time invariant, but the method can be easily extend to the time-dependent case.

$$x_{t+1} = Ax_t + Bu_t$$

We assume starting conditions $x_0 = x_{\text{init}}$ and a finite horizon T. In this setting, we want to compute the control sequence $u_0, \dots, u_{T-1}$ such that:

- $x_0, x_1, \dots, x_T$ is small (*good control*) and
- $u_0, u_1, \dots, u_{T-1}$ is small (*small effort*).

Usually, those are conflicting goals: we can drive x to zero quickly with a large effort u, but we want to minimize some weighted sum of the two. One of the simplest approaches in control system theory is the so-called *linear quadratic regulator*, or LQR. Denoting with $U = [u_0, u_1, \dots, u_{T-1}]$ the complete control vector, we can define the objective to be minimized as:

$$J(U) = \sum_{t=0}^{T-1} \left(x_t^\top Q x_t + u_t^\top R u_t\right) + x_T^\top Q x_T$$

We can assume matrix Q to be symmetric and positive semidefinite, while matrix R is symmetric and positive definite. Note that matrices Q and R act as relative weights to combine good control and small effort.

Again, we assume the matrices Q, R to be time invariant to simplify notation, but they need not be.

Overall, this optimal control problem can thus be formulated as:

$$\begin{aligned} &\min J(U) \\ &\quad x_{t+1} = Ax_t + Bu_t \qquad\qquad t = 0, \dots, T-1 \\ &\quad x_0 = x_{\text{init}} \end{aligned}$$

While formally an optimization problem, without further restrictions this problem is quite simple and can be solved analytically in closed form. One way to do it is to recognize its least-square structure. A more elegant solution approach is based on *dynamic programming* (DP). By introducing the so-called *value function*:

A least-square problem has the form

$$\min_x \|Ax - b\|^2$$

and admits the closed form solution

$$x = (A^\top A)^{-1} A^\top b$$

$$V_t(z) = \begin{cases} \min \sum_{j=t}^{T-1} \left(x_j^\top Q x_j + u_j^\top R u_j\right) + x_T^\top Q x_T \\ x_{j+1} = Ax_j + Bu_j & j = t, \dots, T-1 \\ x_t = z \end{cases}$$

which represents the minimum LQR *cost-to-go*, starting from state z at time t, we can easily show that we can organize the computation in a backward manner, from V_{t+1} to V_t, using the DP *optimality principle*:

The optimal value we are looking for is thus $V_0(x_0)$.

We will not see dynamic programming further in this course, but it is covered by subsequent courses, so stay tuned!

$$V_t(z) = \min_w \underbrace{z^\top Q z + w^\top R w}_{\text{cost at time } t \text{ if } u_t = w} + \underbrace{V_{t+1}(Az + Bw)}_{\text{cost-to-go from where you land at } t+1}$$

After some tedious algebraic manipulation, we can get to the iterative scheme of Algorithm 1.

Algorithm 1: Dynamic programming approach for LQR.

```
$P_T = Q$
**for** $t = T, \dots, 1$ **do**
    $P_{t-1} = Q + A^\top P_t A - A^\top P_t B(R + B^\top P_t B)^{-1} B^\top P_t A$
**end**
**for** $t = 0, \dots, T-1$ **do**
    $K_t = -(R + B^\top P_{t+1} B)^{-1} B^\top P_{t+1} A$
    $u_t = K_t x_t$
    $x_{t+1} = A x_t + B u_t$
**end**
**return** $U = [u_0, \dots, u_{T-1}]$
```

While this basic case can be solved (quite efficiently) in closed form, which is quite atypical in mathematical optimization, as soon as we start imposing additional restrictions, or changing the objective, we get proper optimization problems, that need to be solved with the tools covered in this course.

A first very natural change is imposing simple bounds on the state and control variables:

$$\begin{aligned} \underline{x} &\leq x \leq \overline{x} \\ \underline{u} &\leq u \leq \overline{u} \end{aligned}$$

This alone prevents the usage of the closed form described above, and requires the solution of the problem as a (convex) quadratic program.

Another type of change is replacing the quadratic norm in the objective: for example, suppose that we want to minimize the ℓ_∞ norms of x and u. The optimization problem reads:

$$\begin{aligned} \min \; & \|X\|_\infty + \|U\|_\infty \\ & x_{t+1} = A x_t + B u_t && t = 0, \dots, T-1 \\ & x_0 = x_{\text{init}} \end{aligned}$$

with $X = [x_0, \dots, x_T]$. Despite its appearance, this can actually be formulated as a linear program, because minimizing the ℓ_∞ norm of a vector can be encoded in a linear problem with some artificial variables and constraints.

In details, $\min \|x\|_\infty$ *can be written as:*

$$\begin{aligned} \min \; & z \\ & x_i \leq z && j = 1, \dots, n \\ & -x_i \leq -z && j = 1, \dots, n \end{aligned}$$

Finally, consider the case in which the control u_t is not continuous, but can only take discrete steps, say because it corresponds to the number of pills to take in a medical treatment, or the levels at which a heating machine can be set by the final user. This case can be "easily" dealt with at the modeling stage by just declaring the control variables as integer, and thus obtain a MIP or a MIQP, depending of the objective function.