1 Introduction

1.1 Modeling / formalization

Optimization problems arise in numerous applications. The user is confronted with the task to make some choice in order to minimize (maximize) some cost (performance) under some conditions. In its original form the problem can be formulated in various ways, such as

- design a wing shape which minimizes the drag while assuring a certain lift;
- find the path for a robot arm which consumes the minimal time while visiting given points;
- design an identification experiment on a plant which gives maximal information in given time;
- assign resources (workers, servers, ships, ...) to tasks.

Sometimes, the original formulation of the problem has even nothing to do with optimization:

- determine the flow capacity of a network of tubes;
- compute the (p, q)-norm of a matrix.

In general, the user has to choose decision variables in order to minimize an objective function with respect to some constraints.

The first step to the solution of this problem is to *formalize* it, i.e., to bring it into a form which is amenable to a mathematical treatment. A formalized finite-dimensional optimization problem looks as follows:

$$\min_{x \in X} f(x). \tag{1}$$

In order to transform the problem to this form we have to identify

- the *decision variable* $x \in \mathbb{R}^n$
- the *objective function* $f : \mathbb{R}^n \to \mathbb{R}$
- the *feasible set* $X \subset \mathbb{R}^n$.

This is often itself a far from nontrivial task. When setting up the mathematical description we must keep in mind the solvability of the resulting problem. This may necessitate to deviate from the original problem formulation and to make *approximations*.

The feasible set $X \subset \mathbb{R}^n$ can, e.g., be represented by

- scalar equalities $g_i(x) = 0$
- scalar inequalities $h_j(x) \leq 0$
- matrix inequalities $A_k(x) \succeq 0$
- norm inequalities $||Ax|| \leq c$
- binary or integer constraints $x_i \in \{0, 1\}$, $x_i \in \mathbb{Z}$
- black-box oracles, ...

The cost function may be given

- analytically
- by a black-box oracle
- with a (sub-)gradient
- with a Hessian, ...

Some combinations of cost function and constraints lead to standard problem classes:

- Linear program (LP): linear cost function, linear equality and inequality constraints
- Quadratic program (QP): quadratic cost function, linear equality and inequality constraints
- Standard quadratic program (StQP): quadratic cost function, feasible set is the simplex $\{x \in \mathbb{R}^n \mid \langle \mathbf{1}, x \rangle = 1,\ x \geq 0\}$
- Mixed integer linear program (MILP): linear cost function, binary, integer, and linear equality and inequality constraints
- Quadratically constrained quadratic program (QCQP): quadratic cost function and constraints
- Semi-definite program (SDP): linear cost function, linear equality constraints, linear matrix inequality constraints

Standardization of optimization problems has the advantage that solution methods can be used which do not take into account the specifics of the problem and are designed to solve whole problem classes. This works primarily for problems that are simple enough, and relatively unspecific information about the problem is sufficient to solve it in a reasonable time. The more difficult a problem is, the more dedicated should the solution methods be to take into account any information that is available.

Given an optimization problem, one should first look whether it can be reduced to one of the standard problem classes. On the other hand, research is under way to extend or design solution algorithms for new problem classes.

For many problem classes nowadays ready-to-use optimization software is available, both open source and commercial. The boundary is not well-defined, open source software often requires a license for commercial use, and commercial software is freely available for academic use.

Some software packages are listed below:

- cvx (LP, SOCP, SDP plus binary / integer constraints), passes the problem to a solver
- SDPT3, SeDuMi, SDPA, MOSEK (LP, SOCP, SDP)
- SCIP, CBC, GLPK (LP, MILP)
- SoPlex (exact solution of LP)
- Gurobi, CPLEX (LP, SOCP, MILP)
- SOStools, GloptiPoly (polynomial problems)

If a problem cannot be transformed to an equivalent problem in one of the easily solvable problem classes, one may try to *approximate* it, e.g., by dropping constraints. Such approximations are called *relaxations.*

1.2 Examples

Uniform Approximation: We want to approximate a (complicated) function $g(x)$ uniformly on a domain $G \subset \mathbb{R}^m$ by a linear combination of basis functions $f_k : G \to \mathbb{R}$, $k = 1, \dots, n$. In order to make this task feasible, we approximate the domain G by a *discrete* set $D = \{x_1, \dots, x_N\} \subset G$, e.g., a dense grid. Then we may write the problem as follows:

$$\min_{c \in \mathbb{R}^n} \max_{j=1,\dots,N} \left| g(x_j) - \sum_{k=1}^{n} c_k f_k(x_j) \right|.$$

This is an unconstrained problem of the desired form (1), with $c = (c_1, \dots, c_n)$ being the decision variable, and the maximum of the absolute value being the cost function. Note that this function is considered here as a function of c. The other components of the function are known from the original formulation of the problem and constitute the *data.*

However, the cost function is a piece-wise linear function of c and is as such too complicated for being minimized straightforwardly. We shall therefore introduce an *auxiliary variable* $\tau \in \mathbb{R}$ and add it to the decision variables, and introduce constraints, as follows:

$$\min_{(\tau,c)\in\mathbb{R}\times\mathbb{R}^n} \tau : \qquad -\tau \leq g(x_j) - \sum_{k=1}^{n} c_k f_k(x_j) \leq \tau.$$

Now the cost function is *linear* in the decision variables, and the feasible set is given by *linear* inequalities. Such a problem is called a *linear program* (LP) and can be solved by standard optimization software.

Suppose the function values $g(x_j)$ are collected in a row vector $g \in \mathbb{R}^N$, and the basis function values $f_k(x_j)$ in a matrix $F \in \mathbb{R}^{n\times N}$. Then we may solve the problem by the cvx program

```
cvx_begin
    variable tau
    variable c(1,n+1)
    minimize( tau )
    -tau <= g - c*X
    g - c*X <= tau
cvx_end
```

The process of adding additional auxiliary variables to the problem is called *lifting*. The feasible set of the new augmented problem then projects to the feasible set of the original problem, i.e., it is a *lift* of the original feasible set.

Resource Allocation: Suppose we may fabricate a number of products which we can sell at prices $p_1, \dots, p_n$, respectively. The production of a unit of product l consumes a_{kl} units of raw material k, $k = 1, \dots, K$, of which a total quantity of r_k units is available. We wish to choose the quantities $x_1, \dots, x_n$ of each product to be produced in order to maximize the revenue.

The problem can be formalized as follows:

$$\min_x -\langle p, x\rangle : \qquad Ax \leq r,\ x \geq 0,$$

where A is the $K \times n$ matrix made up of the coefficients a_{kl}, and x, r, p are the vectors made up of the corresponding elements. The constraint $x \geq 0$ is necessary to prevent the conversion of products back to raw materials, which would correspond to a negative quantity x_l.

Again the cost function and the constraints are linear in the decision variables, and the problem has been formalized as an LP. It can be solved by the following cvx program:

```
cvx_begin
    variable x(n,1) nonnegative
    maximize( p*x )
    A*x <= r
cvx_end
```

Max-Cut: We are given a weighted graph $G = (V, E)$ with vertex set $V = \{v_1, \dots, v_n\}$ and edge set $E = \{e_1, \dots, e_m\}$, where each edge e_k has been attached a nonnegative weight w_k. The Max-Cut problem consists in separating (cutting) the vertex set into a disjoint union $S \cup T$ of two subsets such that the sum of the edge weights between the two subsets is maximized.

We shall represent a cut by a vector $x \in \{-1, +1\}^n$, i.e., a vertex of a hyper-cube, where the indices of elements $x_j = -1$ correspond to the vertices in S and the indices of elements $x_j = 1$ to vertices in T. Note that $-x$ and x represent the same cut, as the transformation $x \mapsto -x$ corresponds to an exchange of the sets S, T. Consider the real symmetric matrix $A(x) = \frac{1}{4}(\mathbf{1} - xx^T)$, where $\mathbf{1}$ denotes the all-ones matrix. Then $A_{ij} = 0$ if v_i, v_j are in the same subset S or T, and $A_{ij} = \frac{1}{2}$ otherwise.

Construct a real symmetric $n \times n$ matrix W such that the element W_{ij} equals the edge weight w_k if the vertices v_i, v_j are linked by edge e_k, and zero if v_i, v_j are not linked. Then the sum of the edge weights of the cut is given by the expression $\langle A, W\rangle = \sum_{i,j=1}^n A_{ij}W_{ij}$. We thus arrive at the formulation

$$\min_{x\in\{-1,+1\}^n}(-\langle A(x), W\rangle) = \min_{X\in\mathcal{MC}} -\frac{1}{4}\langle \mathbf{1} - X, W\rangle,$$

where $\mathcal{MC}$ is the *MaxCut polytope*, which is defined as the convex hull of the set of matrices

$$\{X = xx^T \mid x \in \{-1,+1\}^n\} = \{X \succeq 0 \mid X_{ij} \in \{-1,+1\}\},$$

which yields

$$\mathcal{MC} = \{X \succeq 0 \mid \text{diag } X = \mathbf{1},\ \text{rk } X = 1\}.$$

In the first formulation the cost function is quadratic, in the second formulation it is linear. In the second formulation we also have a semi-definite constraint. However, due to the binary constraints on x or X we obtain a MIQP or mixed integer SDP. It can actually be proven that Max-Cut is an NP-hard problem.

The standard relaxation of the Max-Cut problem is obtained by dropping the rank constraint. This yields the semi-definite program

$$\min_{X\succeq 0} -\frac{1}{4}\langle \mathbf{1} - X, W\rangle : \qquad \text{diag } X = \mathbf{1}.$$

It can be solved by the cvx program

```
cvx_begin
    variable X(n,n) semi_definite
    maximize( trace((ones(n)-X)*W)/4 )
    diag(X) == 1
cvx_end
```

An exact solution can be obtained by combining this procedure with a branch-and-bound method.

Min-Cut: Here were are confronted with the same problem as for Max-Cut, but we want to *minimize* the weight of the cut. The weights w_k are assumed to be nonnegative. The cut is to be chosen to separate two given vertices s, t. In contrast to Max-Cut this problem can be reduced to a polynomially sized LP. We shall show below that Min-Cut is equivalent to Max-Flow.

Max-Flow: Let G be a directed graph with vertex set $V = \{v_1 = s, v_2, \dots, v_{n-1}, v_n = t\}$ and edge set $E = \{e_1, \dots, e_m\}$. The distinguished vertices s, t are called the source and the sink. To each edge e_k there are attached weights $w_k^\pm \geq 0$. These are interpreted as flow capacities in and against the direction of the edge. As in the MaxCut problem we build a matrix W, with $W_{ij} = w_k^+$ if edge e_k is from vertex v_i to v_j, $W_{ij} = w_k^-$ if e_k is from v_j to v_i, and $W_{ij} = 0$ if v_i, v_j are not linked by an edge. Note that W is not necessarily symmetric, as $w_k^\pm$ do not need to coincide. The problem consists in finding the maximal flow from the source to the sink through the network.

We shall represent a flow through the network by a skew-symmetric $n \times n$ matrix F. The element F_{ij} designates the actual flow from vertex v_i to vertex v_j. If there is no flow between v_i and v_j, then $F_{ij} = 0$, if the flow is from v_j to v_i, then $F_{ij} = -F_{ji} < 0$.

The Max-Flow problem can then be formalized as follows:

$$\min_{F=-F^T} \sum_{i=1}^{n-1} F_{i1} : \quad F \leq W, \quad \sum_{i=1}^{n} F_{ij} = 0 \quad \forall\, j = 2, \dots, n-1.$$

Here the decision variable is the skew-symmetric matrix F of flows. The flow from s to t is given by the sum of outflows from the source minus the sum of inflows in the source, which is equal to $-\sum_{i=1}^{n-1} F_{i1}$. The inequality $F \leq W$ has to be interpreted element-wise and ensures that the flows remain bounded by the corresponding capacities, $\max(0, F_{ij}) \leq W_{ij}$ for all i, j. The equalities are balance equations that ensure the sum of inflows into an intermediate vertex equals the sum of outflows.

The cost function and the constraints are linear in the $\frac{n(n-1)}{2}$ decision variables F_{ij}, $i < j$, and the problem reduces to an LP. It can be solved with the cvx program

```
cvx_begin
    variable F(n,n) skew_symmetric
    minimize( ones(1,n)*F(:,1) )
    F <= W
    for j = 2:n-1
        ones(1,n)*F(:,j) == 0
    end
cvx_end
```

Equivalence with Min-Cut: To the Min-Cut problem we associate a Max-Flow problem as follows. The vertex set in the Max-Flow problem is the same as in the Min-Cut problem, and to any undirected edge e_k in the Min-Cut problem there corresponds a directed edge in the Max-Flow problem, with weights $w_k^{\pm} = w_k$. An undirected edge in the Min-Cut problem is hence interpreted as a tube allowing a flow in both directions and bounded by the edge weight in absolute value. The resulting matrix W will be symmetric.

Clearly every flow from s to t is bounded from above by the minimum cut. We shall now show that the maximal flow is actually equal to the minimum cut. To this end it suffices to *construct* a cut from the maximal flow with value equal to the flow.

Let F^* be the flow matrix corresponding to the maximal flow. We then define the *residual network* as the network allowing flows F bounded by the inequalities $F \leq W - F^*$, i.e., flows F such that $F + F^*$ is a valid flow through the original network. Note that $W - F^* \geq 0$, because F^* represents a feasible flow. Since F^* is the maximal flow, the residual network does not allow any positive flow from s to t. We then define S as the set of vertices which can be reached from s through the residual network (i.e., vertices v such that there exists a positive flow from s to v), and T as the set of remaining nodes. The maximal flow F^* then flows through the cut defined by S and T. Suppose that the value of this cut is strictly larger than the value of F^*. Then there exists an edge e_k in the cut, linking $v_i \in S$ and $v_j \in T$, such that $w_k > F_{ij}^*$. But then v_j is reachable from v_i through the residual network, contradicting the definition of S and T. Hence the value of the cut must equal the value of the flow F^*.

Since the value of F^* is a lower bound to every cut separating s and t, the value of the cut defined by S and T must be minimal.

Ford-Fulkerson method: For the Max-Flow / Min-Cut problem there exist several algorithms that are more efficient than solving the LP. The conceptually simplest class are the Ford-Fulkerson methods. They use the fact that as long as a given flow is not maximal, the residual network will allow to link the source node to the target node. However, the sub-problem of finding a linking path is solvable in a number of steps which is proportional to the number of edges e. Having found a path connecting s to t in the residual network, we may add the flow along this path to the current flow and obtain the next iterate with a strictly larger capacity.

Whether the method terminates in a finite number of steps depends on the specific algorithm looking for the linking path. The version of Edmonds-Karp finds the shortest linking path by performing a breadth-first search. It has a $O(ne^2)$ overall complexity.