Opinionated survey of (non)convex optimization tools (and related topics)

BY DENIS ROSSET

Perimeter Institute

A generic optimization problem is written

$$\begin{array}{rll} \text{minimize} & f(\vec{x}) & \\ \text{over} & \vec{x} \in \mathbb{R}^n & \\ \text{such that} & x_i \in \mathbb{Z} & i \in \mathcal{I} \subset \{1, \dots, n\} \\ & g_j(\vec{x}) \leq 0 & j \in \mathcal{J} \\ & h_k(\vec{x}) = 0 & k \in \mathcal{K} \end{array} \tag{1}$$

where $\mathcal{J} = \{1, \dots, |\mathcal{J}|\}$ and $\mathcal{K} = \{1, \dots, |\mathcal{K}|\}$.

1 Reference material

The material below is a synthesis of our first-hand experiences with various solvers and problems. Our reference for optimization is the book [9] by Nocedal and Wright (copy here http://fourier.eng.hmc.edu/e176/lectures/NumericalOptimization.pdf), and an excellent reference for *convex* optimization is the book [2] by Boyd, available on GitHub. For the practical aspects, the MOSEK modeling cookbook [1] is excellent; we cite the Release 3.2.1 available on GitHub.

Those books have been written at the turn of the century, but the theory has not changed much; machine learning has simply spurred interest in first order methods due to the size of the models.

2 Definitions

2.1 Number of objectives

An optimization problem without an objective is a *satisfiability problem* or *feasibility problem*.

When numerical errors are present, we can reformulate the feasibility problem

$$\begin{array}{rll} \text{Find} & \vec{x} \in \mathbb{R}^n & \\ \text{such that} & x_i \in \mathbb{Z} & i \in \mathcal{I} \subset \{1, \dots, n\} \\ & g_j(\vec{x}) \leq 0 & j = 1, \dots, J \\ & h_k(\vec{x}) = 0 & k = 1, \dots, K \end{array} \tag{2}$$

as the minimization problem

$$\begin{array}{rll} \text{minimize} & s & \\ \text{over} & \vec{x} \in \mathbb{R}^n, s \in \mathbb{R} & \\ \text{such that} & x_i \in \mathbb{Z} & i \in \mathcal{I} \subset \{1, \dots, n\} \\ & g_j(\vec{x}) \leq s & j = 1, \dots, J \\ & -s \leqslant h_k(\vec{x}) \leqslant s = 0 & k = 1, \dots, K \end{array} \tag{3}$$

so that there is always a feasible solution with s big enough. This is especially helpful for convex problems (more on that later). Then "solver returns infeasible" means that the problem formulation is not numerically stable; otherwise a solution with $s \leqslant 0$ is feasible and $s > 0$ describes infeasibility.

The formulations we will consider later have a single objective. To change a minimization problem into a maximization problem and vice-versa, write $f(\vec{x}) \to -f(\vec{x})$ as some solvers support only one optimization direction.

When several objectives are present, we are speaking of multiobjective optimization.

The simplest approach is to minimize a linear combination of objectives

$$f(\vec{x}) = \alpha_1 f_1(\vec{x}) + \alpha_2 f_2(\vec{x}) + \cdots$$

for various values of $\alpha_1, \alpha_2, \ldots$; this is part of the *scalariztion* approaches. To organize the results, plot the *Pareto front*. We also had success with genetic algorithms which maintain good diversity in the population of solutions (for example NSGA-II [4]).

2.2 Type of objective

The simplest type of objective function is *linear*:

$$f(\vec{x}) = \vec{c}^{\top} \cdot \vec{x} + d. \tag{4}$$

(Strictly speaking, it is *affine* for $d \neq 0$ although this subtle point is often lost. We use *linear* for both.)

Otherwise, the problem is said to have *nonlinear* constraints.

An objective function is *convex* when [2, Chap. 2 and 3]

$$f(\alpha\, \vec{x}_1 + (1-\alpha)\vec{x}_2) \leqslant \alpha f(\vec{x}_1) + (1-\alpha) f(\vec{x}_2), \qquad \alpha \in [0,1]. \tag{5}$$

This implies that any local minimum of f is also a global minimum [2, 4.2.2].

Polynomial objective functions have the form

$$f(\vec{x}) = \alpha + \sum_{i_1} \beta_{i_1} x_{i_1} + \sum_{i_1 \leqslant i_2} \gamma_{i_1 i_2} x_{i_1} x_{i_2} + \cdots \tag{6}$$

Sometimes, objective functions are not known in closed form: it can be obtained from an integration, solving a differential equation, solving an inner optimization problem, even running an experiment. The computation of $f(\vec{x})$ may even be *noisy*. In those cases, the derivatives $\partial f / \partial x_i$, $\partial^2 f / \partial x_i \partial x_j$ may not be available, and cannot be computed by finite difference schemes: the solving algorithm needs to be *derivative-free* [9, Chap. 9].

2.3 Type of constraints

If $\mathcal{I} = \mathcal{J} = \mathcal{K} = \emptyset$, the problem is an *unconstrainted* optimization problem.

When $\mathcal{I} = \mathcal{K} = 0$ (no integrality or equality constraints), and all $g_j(\vec{x})$ have the form of simple bounds

$$\ell_i \leqslant x_i \leqslant u_i \qquad \Rightarrow \qquad \ell_i - x_i \leqslant 0, \quad , x_i - u_i \leqslant 0, \tag{7}$$

the problem is a *bounded* optimization problem.

A problem has *linear constraints* when all g_j and h_k are linear (4) and there are no integrality constraints ($\mathcal{I} = 0$). Otherwise, the problem is said to have *nonlinear* constraints.

A problem has *convex constraints* when all inequality constraints g_j are convex (5), all equality constraints h_k are linear (4).

Constraints can also be polynomial as in (6).

A *semidefinite programming* (SDP) constraint is constructed as follows. Given fixed symmetric matrices C and $\{A_i\}_i$, we write the affine combination $\chi = C - \sum_{i=1}^{n} A_i x_i$ using the optimization variables (the signs are somewhat arbitrary for now). We then impose that χ is a *semidefinite positive* matrix, i.e. that its eigenvalues $\text{eig}(\chi)$ are all nonnegative [2, 4.6.2]. Semidefinite constraints are quite expressive (two examples: [10] [5]) and relatively efficient to compute.

2.4 Scale of problems

This relates to the number of variables and constraints, and will guide algorithm selection.

Small problems include a handful of variables and constraints (say $1 - 10$). They can sometimes be solved by exact/algebraic methods.

Medium-scale problems usually involve less than a thousand variables and constraints: for those, second-order derivative information (the Hessian) can be stored in memory using a dense matrix.

Large-scale problems are such that second-order derivative information is too expensive to be stored in memory (in our experience, algorithms are more memory bound than CPU bound). Algorithms use the structure of the problem to reduce computational requirements: for example sparsity, approximating second-order information or even not using second-order information at all (at the price of convergence speed).

2.5 Problem types

Objective type	Constraint type	Integrality	Problem type
Linear	Linear	$\mathcal{I} = \emptyset$	Linear program (LP)
Linear	Linear	$\mathcal{I} \neq \emptyset$	Mixed integer linear program (MILP)
Linear	Semidefinite	$\mathcal{I} = \emptyset$	Semidefinite program (SDP)
Convex	Convex	$\mathcal{I} = \emptyset$	Convex program
Polynomial	Polynomial	$\mathcal{I} = \emptyset$	Polynomial optimization problem (POP)
Nonlinear	None	$\mathcal{I} = \emptyset$	Unconstrained optimization problem
Nonlinear	Bounds	$\mathcal{I} = \emptyset$	Bounded optimization problem
Nonlinear	Nonlinear	$\mathcal{I} \neq \emptyset$	Mixed integer nonlinear program (MINLP)

Note that LP, SDP are convex programs. Integrality constraints appear mostly in MILP, generic MINLP are an active area of research.

3 Approaches

3.1 Problems specified by user-defined functions

In this approach, algorithms start with a user-given point $\vec{x}_0$, gather information about the objective and constraints around $\vec{x}_0$, compute a first iterate $\vec{x}_1$ that respects the constraints and such that $f(\vec{x}_1) < f(\vec{x}_0)$, repeat the procedure to obtain $\vec{x}_2$, etc..., until one of the stopping criteria of the algorithm is achieved (for example, the improvement $f(\vec{x}_{i+1}) - f(\vec{x}_i)$ is small, the norm of the estimated gradient is below some tolerance, or the step size $\|\vec{x}_{i+1} - \vec{x}_i\|$ is too small).

The main difference here is between convex and nonconvex programs. If the problem is not convex, there is no guarantee that the returned solution is globally optimal, or even feasible, while convex programs provide such certainty.

(Several heuristics are available for nonconvex programs, but they are outside the scope of this survey).

3.1.1 Derivative-free algorithms

	Objective type	Constraint type	Integrality	Problem type
•	Nonlinear	None	$\mathcal{I} = \emptyset$	Unconstrained optimization problem
	Nonlinear	Bounds	$\mathcal{I} = \emptyset$	Bounded optimization problem

The algorithms below do not require gradient/Hessian information, see [9, Chap. 9].

The Nelder-Mead or amoeba method is common for unconstrained problems: `fminsearch` in Matlab standard toolbox; available in the `https://github.com/JuliaNLSolvers/Optim.jl` Julia toolbox. It works on functions that are not differentiable, but can converge to non-optimal points. It is best used on problems with a number of variables between 10 and 20.

Model-based methods approximate the objective function by taking a number of samples and building a linear or quadratic model. We found the BOBYQA [11] algorithm for bounded optimization problems to work very efficiently, even in the presence of noise; algorithms in this family work up to several hundred variables. See `https://github.com/stevengj/nlopt` for the MATLAB interface and `https://github.com/JuliaOpt/NLopt.jl` for the Julia interface.

3.1.2 Derivative-based algorithms

There, the user supplies the objective and the constraints along with the gradients ∇f, ∇g_j, ∇h_k (some algorithms require second-order derivatives, but we do not recommend them outside of problems with specific structure).

For unconstrained problems, the BFGS algorithm uses the values $f(\vec{x}_i)$ and $(\nabla f)(\vec{x}_i)$ of the known iterates to approximate the second-order derivative information (Hessian). It is provided by the Optimization Toolbox in MATLAB through the `quasi-newton` algorithm of the function `fminunc`; in Julia, see the `https://github.com/JuliaNLSolvers/Optim.jl` toolbox. In Julia, a limited memory version (LBFGS) is provided for large scale problems, where the Hessian matrix is approximated using the last m iterates, reducing memory use.

For constrained problems, interior-point methods try to move in a central path at a certain distance of the constraints. In the MATLAB Optimization Toolbox, `fmincon` has an implementation of the interior-point method; the `Ipopt` solver has Julia bindings: `https://github.com/JuliaOpt/Ipopt.jl`. MATLAB additionally has an sequential quadratic programming algorithm that can works more precisely for medium scale problems (instead of following a central path, it actively searches for inequality constraints that are be saturated).

The Knitro nonlinear solver is state-of-the-art but academic licenses are not free.

Final note: if the Julia code is written with generic types in mind, the gradients can be automatically computed by forward autodifferentiation.

3.2 Problems given in a canonical form

Some problem types have canonical forms. For example, linear programs (LP) have the (primal) form:

$$\begin{aligned} \text{minimize} \quad & \vec{c}^{\top}\vec{x} \\ \text{over} \quad & \vec{x} \in \mathbb{R}^n \\ \text{such that} \quad & A\vec{x} = \vec{b} \\ & \vec{x} \geqslant 0 \end{aligned} \tag{8}$$

while semidefinite programs (SDP) [2, 4.6.2] have the (primal) form:

$$\begin{array}{rl} \text{minimize} & \operatorname{Tr} CX \\ \text{over} & X \in \mathbb{R}^{n\times n}, X = X^\top \\ \text{such that} & \operatorname{Tr} A_k X = b_k, k \in \mathcal{K} \\ & \operatorname{eig}(X) \geqslant 0 \end{array} \tag{9}$$

In that case, the problem is fully specified by the tuples $(A, \vec{b}, \vec{c})$ or $(\{A_k\}, \vec{b}, C)$, and this data can be given to the solver using text files. Then the solver can run independently.

For linear programs in exact precision, see also Section 3.4.1.

The solver Gurobi https://www.gurobi.com/ is excellent for linear and mixed-integer linear programming, though MOSEK is close (and MOSEK supports SDPs).

For convex problems given in canonical forms, the MOSEK https://www.mosek.com/ solver represents the state of the art. It has a free academic license. For extended precision arithmetic, use the SDPA family http://sdpa.sourceforge.net/family.html. We did not have great experiences with the SCS solver (https://github.com/cvxgrp/scs), sometimes recommended in Julia examples, but it seems to have applications. Its code is particularly simple to follow though.

All these solvers interface with Matlab and Julia; for Julia, see in particular https://github.com/JuliaOpt.

3.3 Using a modeling framework

Writing a convex program in a canonical form (8)-(9) can be tedious. Luckily, both MATLAB and Julia have modeling frameworks that enable the use of standard syntax. Under the hood, the problems are transformed in the canonical form and solved. We describe below the different frameworks available. Some of them work even for nonconvex programs whose objective and constraints are given in a closed form.

Still, modeling frameworks are not magical: they often result in canonical formulations that are more complex than necessary, with obvious redundancies (some frameworks, for example, do not substitute simple equality constraints such as $x_1 = x_2$).

3.3.1 YALMIP (Matlab)

YALMIP is the one of the oldest framework in existence. It's based on MATLAB, though it runs on the open source clone Octave (https://www.gnu.org/software/octave/).

The core of YALMIP is convex optimization over real-valued linear or semidefinite programs, and that core is stable, and interfaces with myriads of solvers.

Beyond convex programming, YALMIP has support for nonconvex problems, complex variables, dualization, global optimization, polynomial optimization, ... but often it is not possible to use two extensions at the same time (for example, getting the dual variable corresponding to a complex SDP constraint). We observed that YALMIP performs the translation of the problem to its canonical form quite mechanically, without applying obvious eliminations.

YALMIP has amazing documentation.

3.3.2 CVX (Matlab)

The CVX toolbox (http://cvxr.com/cvx/) is restricted to convex programs, using an approach known as Disciplined Convex Programming [6]: the problem needs to be entered in a way that makes the convexity obvious to the toolbox (in practice, this is not a huge restriction). CVX encourages a modular style of convex programming: convex sets of importance can be defined in user functions and reused.

The version 3 has critical bugs, but the version 2.2 is rock solid. It supports complex variables, including getting the dual variables of complex constraints. The documentation is great. CVX supports a limited number of solvers, compared to YALMIP, but it supports MOSEK, which is often all that is needed.

CVX performs model translation better than YALMIP, and applies simple variable substitutions.

3.3.3 JuMP (Julia)

JuMP (https://github.com/JuliaOpt/JuMP.jl) looks like the Julia equivalent of YALMIP. It interfaces myriads of solvers, offers nonconvex modeling and a quantity of extensions. It is actively used in research. It does not seem to be as versatile as YALMIP yet, and the documentation is sometimes lacking, in particular for its extensions. Of course YALMIP has nearly two decades of existence.

3.3.4 Convex.jl (Julia)

On the other hand, Convex.jl (https://github.com/JuliaOpt/Convex.jl) is a Julia implementation of Disciplined Convex Programming, and has a limited scope compared to JuMP. In our limited experience, it seems more robust and ready to tackle medium or large scale optimization problems; it supports a large variety of solvers. Its documentation is pretty great.

3.4 Special cases

3.4.1 Exact arithmetic

To solve linear programs, including those with multiple objectives, the feasible set can be manipulated using Fourier-Motzkin elimination [12]. We employed succesfully, for combinatorial problems.

Software	Link	Platform
PORTA	http://porta.zib.de/	Binary
PANDA	http://comopt.ifi.uni-heidelberg.de/software/PANDA/	Binary

Julia has bindings to many libraries: https://juliapolyhedra.github.io/ under a common interface, which we haven't tried.

We also used rational LP solvers, which can be very efficient for medium scale problems:

Software	Link	Platform
QSopt_ex	https://www.math.uwaterloo.ca/~bico/qsopt/ex/index.html	Binary
SoPlex	https://soplex.zib.de/index.php	Binary
GLPK (exact mode)	https://www.gnu.org/software/glpk/	Julia, Matlab

For polynomial problems, the elimination of quantifiers can be done using the Cylindrical Algebraic Decomposition [3], implemented in Mathematica.

3.4.2 Verified bounds

Semidefinite programs are often too expensive to be solved exactly, apart from small examples [7]. Nevertheless certified bounds can be obtained using the VSDP package in Matlab [8].

3.4.3 Discrete/combinatorial optimization

When all variables are discrete and bounded, ideally all $x_i \in \{0, 1\}$, consider using a SAT solver. We recommend MiniSat, http://minisat.se/. Beyond that, the Z3 solver https://github.com/Z3Prover/z3.

Bibliography

[1] MOSEK ApS. MOSEK modeling cookbook. 2020.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, UK ; New York, New. edition, mar 2004.

[3] Bob F. Caviness and Jeremy R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Texts & Monographs in Symbolic Computation. Springer-Verlag, Wien, 1998.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, apr 2002. Conference Name: IEEE Transactions on Evolutionary Computation.

[5] Hamza Fawzi and Omar Fawzi. Efficient optimization of the quantum relative entropy. *J. Phys. A: Math. Theor.*, 51(15):154003, mar 2018.

[6] Michael Grant, Stephen Boyd, and Yinyu Ye. Disciplined Convex Programming. In Leo Liberti and Nelson Maculan, editors, *Global Optimization*, number 84 in Nonconvex Optimization and Its Applications, pages 155–210. Springer US, 2006.

[7] Didier Henrion, Simone Naldi, and Mohab Safey El Din. SPECTRA-a Maple library for solving linear matrix inequalities in exact arithmetic. *ArXiv preprint arXiv:1611.01947*, 2016.

[8] Christian Jansson. VSDP: A MATLAB software package for verified semidefinite programming. *DELTA*, 1:4, 2006.

[9] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, New York, 1st. ed. 1999. Corr. 2nd printing edition edition, apr 2000.

[10] Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program., Ser. B*, 96(2):293–320, may 2003.

[11] Michael JD Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, pages 26–46, 2009.

[12] H. P. Williams. Fourier's Method of Linear Programming and Its Dual. *The American Mathematical Monthly*, 93(9):681–695, nov 1986.