Epigraph proximal algorithms for general convex programming

Matt Wytock, Po-Wei Wang and J. Zico Kolter
Machine Learning Department
Carnegie Mellon University
mwytock@cs.cmu.edu

Abstract

This work aims at partially bridging the gap between (generic but slow) "general-purpose" convex solvers and (fast but constricted) "specialized" solvers. We develop the Epsilon system, a solver capable of handling general convex problems (inputs are specified directly as disciplined convex programs); however, instead of transforming these problems to cone form, the compiler transforms them to a higher-level "sum-of-prox" which can be solved via splitting methods, and which maintains significantly more problem structure. To make such transformations feasible for general convex problems, a key contribution is the development of efficient proximal and *epigraph projection* operators a wide range of convex functions. As we show, the resulting system improves substantially, often by an order of magnitude or more, over CVXPY combined with the splitting cone solver (SCS), a state-of-the art method for solving DCPs via their conic form.

1 Introduction

In convex optimization, there has long existed a dichotomy between "general-purpose" and "specialized" convex solvers. General-purposes solvers, such as those that solve general conic optimization problems using interior point methods [7], can solve a very broad set of problems (particularly using modeling frameworks such as those based upon disciplined convex programming [3]), but often do not scale to large problems. Conversely, specialized algorithms are often orders of magnitude faster than these general purpose solvers, but can be difficult to adapt to even slightly different problems.

In this paper, we take a step towards bridging this gap. We develop the Epsilon system and solver framework, currently available as an open source library at `http://epopt.io/`. As with general purpose solvers, Epsilon can solve a wide class of optimization problems: it takes as input any problems specified as a disciplined convex program. However, instead of then converting these problems to conic form, as is done in existing DCP methods, we convert the problems to an intermediate representation we call *prox-affine* form, a sum of functions that each have efficient proximal operators. This form maintains more structure in the original problem, and allows for efficient solvers based upon operator splitting methods. A key innovation to the approach is that we define a broad set of proximal operators, specifically including *epigraph projections*, that is sufficient to represent any convex program expressable in current DCP modeling frameworks, without resorting to cone form. As we show below, the resulting system is able to solve many standard problems an order of magnitude faster (or more) than existing state-of-the-art solvers (even those already based upon large-scale algorithms).

2 Background

Our work builds heavily upon two main lines of work: 1) DCP methods [3] and systems built upon these methods; 2) proximal algorithms, specifically operating splitting algorithms like the

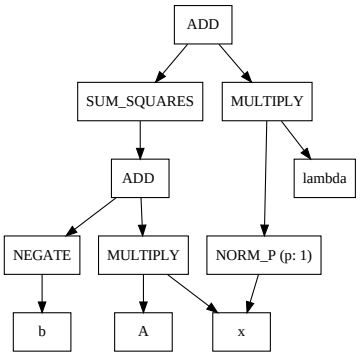

Figure 1: Abstract syntax tree for lasso $\|Ax - b\|_2^2 + \lambda\|x\|_1$.

alternating direction method of multipliers (ADMM) [1]. DCP methods allow users to construct convex programs in an intuitive syntax by a set of compositions rules (which provide a calculus for guaranteeing that a given problem is convex), and by a set of transformations that translate convex or concave functions to a canonical cone form. CVX [4] and CVXPY [2] are two well-known implementations of the DCP ruleset for the MATLAB and Python languages respectively, and we use the native CVXPY problem representation as input to our system.

With respect to algorithms for solving convex problems, proximal-based methods have seen a surge of interest in recent years. The proximal operator for a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\text{prox}_f(v) = \underset{x}{\text{argmin}}\, \lambda f(x) + \frac{1}{2}\|x - v\|_2^2. \tag{1}$$

Several convex functions have simple closed form-solutions for their prox operator (see e.g. [6]), which can be exploited when optimizing convex problems that involves these functions. ADMM is one such algorithm that we will use below, but there are a wide range of similar algorithms.

3 The Epsilon compiler and algorithm

The main idea of Epsilon is that instead of reducing a general convex problem to conic form, we reduce it to a set of functions with efficient proximal operators. Specifically, we reduce the optimization problem to *prox-affine* form

$$\underset{x}{\text{minimize}} \quad \sum_{i=1}^{N} f_i(H_i(x)) \tag{2}$$

where each $f_i \circ H_i$ is "prox-friendly" in that there exists a efficient proximal operator for the function: this include linear equalities, cone constraints, many convex functions, and epigraph projection, a class of operators we introduce shortly. While this may appear difficult for general constraints and function compositions, a key innovation of Epsilon is that we can implement a set of operators that are sufficient for capturing the full generality of disciplined convex programs without resorting to conic form.

Briefly, the compiler proceeds by recursively processing the abstract syntax tree corresponding to the DCP objective and constraints (see Figure 1 for a simple example) and applying the transformation rules (these are simplified slightly to ignore multiplicative scaling, but the intuition is similar):

1. If the expression is itself a prox-friendly function, we return.
2. If the expression is a sum of expressions, we recursively parse each one.
3. If the expression is a convex composition of two functions $g(f(x))$ for convex f, we introduce the variable t, add the epigraph indicator $I(f(x) \leq t)$, and add the function $g(t)$ (for affine or concave f we equivalently add $=$ or $\geq$ constraints).

After applying these transforms, the problem will be in the form of (2) above. Indeed, note that traditional disciplined convex programming systems apply similar transformation to reduce every

problem to include only linear functions and cone constraints—our prox-affine representation simply allows for a much richer set of operators.

Next, we further transform the problem to a *separable* prox-affine form by introducing new variables and additional equality constraints, i.e.

$$\underset{x_1,\ldots,x_n}{\text{minimize}} \quad \sum_{i=1}^{N} f_i(H_i(x_i)) \quad \text{subject to} \sum_{i=1}^{N} A_i(x_i) = 0 \tag{3}$$

where $x_i, \ldots, x_n$ are the optimization variables. In the simplest case, the A_i terms are consensus constraints that enforce the x_i variables to be multiple "copies" of the original x variables, but more complex equality constraints can also be moved from indicator functions in the objective to direct equality constraints in the A_i terms.

Finally, once the problem has been transformed to have a separable objective, we can solve it via several potential splitting algorithms. In the current version of Epsilon, we solve the problem using ADMM, which has the updates:

$$\begin{aligned} x_i^{k+1} &:= \underset{x_i}{\text{argmin}}\, f_i(H_i(x_i)) + \frac{1}{2}\left\| \sum_{j=1}^{i-1} A_j(x_j^{k+1}) + A_i(x_i) + \sum_{j=i+1}^{N} A_j(x_j^k) + u^k \right\|_2^2 \\ u^{k+1} &:= u^k + \sum_{i=1}^{N} A_i(x^{k+1}), \end{aligned} \tag{4}$$

where the x_i-update is handled using the proximal operator and where u denotes a set of (scaled) dual variables.

Implementation of proximal and epigraph projection operators A key development in making Epsilon an efficient yet fully general algorithm is to develop efficient proximal operators for all *all* resulting terms in the prox-affine form. Due to the nature of the DCP formulation, when we perform these reductions, many of the resulting constraints will be of the form: $I(f(x) \leq t)$, for some prox-friendly function f and variable t. This is the indicator of an *epigraph set* $\mathcal{E} = \{(x,t) \mid f(x) \leq t\}$, and its corresponding proximal operator will be the projection onto this set:

$$\text{prox}_I(v,s) = \underset{x,t}{\text{argmin}} \frac{1}{2}\|x-v\|_2^2 + \frac{1}{2}(t-s)^2 \quad \text{subject to } f(x) \leq t. \tag{5}$$

While the epigraph projection is similar to the proximal operator (the dual function of (5) can be expressed in terms of the proximal operator), it is seemingly more computationally involved to compute in many cases. A key contribution of this work is the development of both proximal *and* epigraph projection operators for many of the functions in the DCP ruleset, in particular often developing epigraph projection algorithms with the same computational complexity as their proximal counterpart.

In particular, we develop proximal and epigraph algorithms for the following functions and their sums: absolute value, square, hinge, deadzone, quantile loss, logistic, inverse positive, negative log, exponential, negative entropy, KL divergence, quadratic over linear, max element, sum largest, ℓ_1 norm, ℓ_2 norm, ℓ_∞ norm, k-largest norm, log sum exp, least-squares, fused lasso, matrix operator norm, trace norm, Frobenius norm, log det, matrix fractional, trace inverse, trace matrix exponential, maximum eigenvalue, linear equalities (in addition to standard cone projections). To solve these operators, we apply some mix of: 1) linear/quadratic/cubic solvers; 2) Newton method on the primal; 3) dual Newton method (when dual problem of epigraph projection has an explicit form); 4) primal-dual Newton (for unconstrained primal problems) 5) implicit dual Newton (a novel algorithmic development for the case with non-trivial constraints in primal problem where the dual has no closed form); 6) quickselect sum-of-clip (a novel $O(n)$ algorithm developed finding zero of a monotonic, piecewise linear function). For the proximal and epigraph projection of matrix-variable functions that are orthogonally invariant, the operator can be solved by using the SVD of input matrix and then applying the corresponding operator to the singular values.

Example: Robust SVM. An "ℓ_∞" robust SVM can be written as the optimization problem

$$\underset{w}{\text{minimize}} \quad \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{m} \left[1 - y_i \cdot w^T x_i + \|P^T w\|_1\right]_+ \tag{6}$$

	Epsilon		CVXPY+SCS	
	Time	Objective	Time	Objective
basis_pursuit	2.96s	1.45×10^2	16.99s	1.45×10^2
covsel	0.77s	3.63×10^2	23.50s	3.62×10^2
group_lasso	9.61s	1.61×10^2	23.31s	1.61×10^2
hinge_l1	5.42s	1.50×10^3	52.62s	1.50×10^3
huber	0.51s	2.18×10^3	3.39s	2.18×10^3
lasso	3.88s	1.64×10^1	22.02s	1.63×10^1
least_abs_dev	0.38s	7.09×10^3	3.81s	7.10×10^3
logreg_l1	4.84s	1.04×10^3	55.53s	1.04×10^3
lp	0.56s	7.77×10^2	5.47s	7.75×10^2
mnist	1.44s	1.53×10^3	227.65s	1.60×10^3
quantile	16.40s	3.64×10^3	88.60s	4.99×10^3
tv_1d	0.50s	2.13×10^5	47.28s	3.51×10^5
tv_denoise	24.17s	1.15×10^6	372.86s	1.69×10^6

Table 1: Comparison of running time and objective value for Epsilon and CVXPY+SCS. Code for problem definitions is available at `http://epopt.io/`.

where w is our optimization variable and $[x]_+ = \max(0, x)$. This formulation is identical to the standard SVM formulation except that each x_i is itself an uncertain set $x_i + Pu_i$ for all $\|u_i\|_\infty \leq 1$.[1] After transformation to prox-affine form and some simplifications, the problem takes on the form

$$\underset{w,z,t_1,t_2}{\text{minimize}} \quad \frac{1}{2}\|w\|_2^2 + C[t_1]_+ + I(1 - \text{diag}(y)Xw + t_2 1 = t_1) + I_{\text{epi}\,\|\cdot\|_1}(z, t_2) + I(P^T w = z) \quad (7)$$

with optimization variables $w \in \mathbb{R}^n$, $z \in \mathbb{R}^n$, $t_1 \in \mathbb{R}^m$, $t_2 \in \mathbb{R}$. Each term here has an efficient prox operator (including the epigraph projection for the ℓ_1 norm), so the above algorithm can be applied. Though the above transformation seems complex in practice, the Epsilon compiler and algorithm carries out all the steps involved with a simple set of rules.

4 Results

In this section we compare Epsilon to an existing approaches for general convex programming based on transforming problems to cone form and using conic solvers. In particular, we compare to CVXPY using the splitting cone solver (SCS) [5]. At a high level, both CVXPY+SCS and Epsilon work similarly in that they reduce a disciplined convex problem to an intermediate representation and apply a variant of ADMM. The difference is in the form of this intermediate representation (prox-affine vs. cone program) and in the larger set of computational atoms available to Epsilon (namely, the proximal operators described in Section 3) relative to the set of cone projections implemented by SCS. In both systems the symbolic transformations are implemented in Python and the backend numerical routines are written in C/C++.

The running times in Table 1 show that on all problem examples considered, Epsilon is faster than CVXPY+SCS and often by a wide margin. While we use the default stopping criteria for both systems, since they solve different intermediate representations using first-order methods, the objective values are not always identical. For most examples, the objective values are within a relative tolerance of 10^{-2} with the exceptions being case in which Epsilon achieves a significantly lower objective value (these typically are examples in which the problem is poorly-conditioned in cone form). In general, we observe that Epsilon tends to solve problems in fewer ADMM iterations and for many problems the iterations are faster due in part to operating on a smaller number of variables.

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[1] This is usually formulated with uncertain defined by an ℓ_2 norm, but we describe an ℓ_∞ version specifically to highlight the epigraph project operations separate from the traditional second order cone.

[2] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. 2015.

[3] Michael Grant, Stephen Boyd, and Yinyu Ye. *Disciplined convex programming*. Springer, 2006.

[4] Michael Grant, Stephen Boyd, and Yinyu Ye. CVX: Matlab software for disciplined convex programming, 2008.

[5] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Operator splitting for conic optimization via homogeneous self-dual embedding. *arXiv preprint arXiv:1312.3039*, 2013.

[6] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):123–231, 2013.

[7] Florian A Potra and Stephen J Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281–302, 2000.