A Note on Extended Formulations for Cardinality-based Sparsity

Cong Han Lim  
Wisconsin Institute for Discovery, University of Wisconsin-Madison

clim9@wisc.edu

Abstract

We provide compact convex descriptions for the \(k\)-support and ordered-weighted \(\ell_1\) (OWL) norms. This can be used in convex optimization solvers by non-experts to evaluate the utility of these norms in their applications and also to incorporate additional constraints. The first set of formulations we provide are based on simple dynamic programming concepts and have \(O(nk)\) constraints for the \(k\)-support norm and \(O(n^2)\) constraints for the OWL norm. The second set uses sorting networks to achieve \(O(n \log k)\) and \(O(n \log n)\) respectively. We assume no prior background on extended formulations and discuss practical issues in CVX.

1 Introduction

The \(\ell_1\) norm is synonymous with sparsity within the machine learning and statistics communities, and two generalizations of this have been introduced recently. The first is the \(k\)-support norm [2]:

\[
\|x\|_{k}^{sp} := \min \left\{ \sum_{I \in \mathcal{G}_k} \|v_I\|_2 : \text{supp}(v_I) \subseteq I, \sum_{I \in \mathcal{G}_k} v_I = x \right\},
\]

where \(\mathcal{G}_k\) is the set of all cardinality \(k\) subsets of \([n]\). This an alternative to the elastic net (which simply adds the \(\ell_1\) and \(\ell_p\) norm) by using \(k\)-sparse vectors with unit \(\ell_p\) norm. Another extension of the \(\ell_1\) norm is to the OSCAR/OWL/SLOPE norms [5, 12, 4] (we use OWL from here on)

\[
\|x\|_{owl} := \sum_{i \in [n]} w_i |x_i^+|,
\]

a variant of the weighted \(\ell_1\) norm where \(x^+\) denotes the vector obtained by sorting the entries in descending magnitude and \(w_1 \geq w_2 \geq \ldots \geq w_n \geq 0\). Both of these norms are known to offer better prediction performance and handle correlated variables better than the \(\ell_1\) norm. The OWL norm also clusters correlated variables and controls the false discovery rate.

Our goal is to make these norms more accessible by providing efficient ways to model the cones

\[
\{(x, \lambda) \in \mathbb{R}^{n+1} : \|x\|_1 \leq \lambda\}
\]

using linear and convex constraints for each of the two norms, and also for variants induced by different \(\ell_p\) norms [10]. These can be used in existing frameworks for modeling and solving convex programs (e.g. CVX and JuMP) and we can easily incorporate the norm in both the objective and constraints. For example, we can model the standard forms of regularized problems:

\[
(a) \min_{x \in \mathbb{R}^n} f(x) \text{ s.t. } \|x\|_1 \leq \lambda, \quad (b) \min_{x \in \mathbb{R}^n} \|x\|_1 \text{ s.t. } f(x) \leq \sigma, \quad (c) \min_{x \in \mathbb{R}^n} f(x) + \lambda \|x\|_1.
\]

We use two different methods to construct these formulations. The first resembles dynamic programming and is simple to describe, but has many linear and nonlinear constraints. The second leverages the sorting network-based construction of the permutahedron by Goemans [8] and is significantly more compact. The extended formulations we present here are new, and with the exception of the OWL norm [5], no polynomial-sized formulations for these norms have been provided in the literature.

When to use extended formulations. We are able to solve regression problems with \(n = 100\) within seconds and \(n = 1000\) within minutes using CVX with Gurobi on a modern dual core laptop. We provide our code at https://github.com/limconghan, which can be used by anyone with basic knowledge of CVX and can be used for small to medium-sized problems. We can also easily incorporate additional constraints or further change the formulation without having to modify any algorithms. This allows one to rapidly prototype new uses for the norms.

For practitioners familiar with convex optimization techniques who need faster methods, we recommend using the efficient \(O(n \log n)\) algorithms for the projection operator [7] or the proximal operator [4, 12] for these norms. If more constraints are present, we can still use these operators with splitting methods such as ADMM.

Preliminaries. An extended convex formulation of a set $S \in \mathbb{R}^n$ is a collection of convex constraints describing a higher-dimensional set $T \in \mathbb{R}^{n+m}$ such that $S = \{ s \in \mathbb{R}^n : (s, t) \in T \}$. Using additional variables can significantly reduce the number of constraints needed. For example, the $\ell_1$ ball requires $O(2^n)$ constraints in $\mathbb{R}^n$, but with another $n$ variables only requires $O(n)$ constraints.

We adopt the atomic norm perspective [6] in this paper. Let $A$ denote a set of vectors (atoms). The atomic norm is given by the gauge function of $A$: $\|x\|_A := \inf \{ \lambda > 0 : x \in \lambda \text{conv}(A) \}$. In other words, the unit ball of the norm $\{ x : \|x\|_A \leq 1 \}$ is given by $\text{conv}(A)$.

We now describe the atoms of the norms induced by a particular choice of an $\ell_p$ norm. The atoms of the $(k, p)$-support norm are precisely the set of vectors with unit $\ell_p$ norm that are supported by at most $k$ indices (the standard $k$-support norm has $p = 2$). Let $\ell_q$ denote the dual of $\ell_p$ (i.e. $1/p + 1/q = 1$). The OWL norm is a special case of the smooth OWL norm [10, 11] with $p = \infty$, and the atoms are

$$\bigcup_{i=1}^m \{ x : \|x\|_p = \left( \sum_{j=1}^i w_j \right)^{-1/q}, |\text{supp}(x)| = i \}. \quad (2)$$

The atoms corresponding to each $i$ are precisely the atoms for the $(k, p)$-support norm where $k = i$ with a scaling factor. This last fact means that we can leverage constructions for $(k, p)$-support norms to obtain those for the smooth OWL norm. For notational convenience, we will also refer to the generalized $\ell_p$ variants by the original name.

2 Extended Formulations via Dynamic Programming Ideas

We first describe how to model the atoms of the $k$-support norm. Consider the following process for picking $k$ indices from $[n]$:

1. Start at the first index with a sparsity budget of $k$ and proceed through the indices in order.
2. At index $i$, decide if we want to pick it. If yes, decrease the sparsity budget by one. Repeat this until we exhaust the sparsity budget or reach the end.

Figure 1 illustrates this process. The pair in each box represents the current index and the current remaining sparsity budget. The diagonal arrow pointing out from $(i, j)$ means that we have picked index $i$ when we have $j$ budget left.

Figure 1: Picking $k$ indices.

The formulation for the $k$-support norm builds on the process. We now have to allocate an $\ell_p$ budget as we pick the support. Assume all variables (except for $x_i$) are nonnegative:

$$b_0 \geq b_{1, k} \quad \text{(3)}$$
$$b_{i, j} \geq c_{i, j} + u_{i, j} \quad \text{for } i \in [n], j \in [k] \quad \text{(4)}$$
$$c_{i, j} \geq \|a_{i, j}, l_{i, j}\|_p \quad \text{for } i \in [n], j \in [k] \quad \text{(5)}$$
$$b_{i+1, j} \leq u_{i, j} + l_{i, j+1} \quad \text{for } i \in [n-1], j \in [k-1] \quad \text{(6)}$$
$$|x_i| \leq \sum_{j \in [k]} a_{i, j} \quad \text{(7)}$$

The $b$ variables correspond to the $\ell_p$ budget, the $c$ and $u$ variables correspond to the case where we pick or did not pick the index respectively, and the $a$ and $l$ variables describe the allocated and left over $\ell_p$ budget after picking the particular index.

The OWL formulation exploits the fact that the unit ball is the convex hull of many $k$-support balls over different levels of $k$. The formulation makes us to pick a $k$ corresponding to one set in the union (2) up front, and the rest of the process then proceeds in the same manner. Thus, the only change we need to make is to replace (3) in the $k$-support norm formulation with

$$b_0 \geq \sum_{j=1}^n \left( \sum_{i=1}^j w_i \right)^{-1/q} b_{i, j} \quad \text{(8)}$$

We now show that these formulations model the correct sets. If at most one term on the right side of inequalities (4) and (6) is nonzero, then we have at most $k$ nonzero $x_i$ and we obtain an atom of the set. If both terms in (4) are nonzero, then we are handling a convex combination of atoms.

2
Proposition 2.1. The dynamic programming based constructions give us \(O(nk)\) and \(O(n^2)\) constructions for the \(k\)-support and OWL norms respectively.

3 Extended Formulations via Sorting Networks

We can obtain more compact extended formulations by exploiting the symmetry of these norms. For any set \(S\), let \(\mathcal{P}(S) := \text{conv}\{x : x \text{ is the permutation of some } s \in S\}\), and let \(\mathcal{P}_\pm(S) := \text{conv}\{x : x \text{ or } -x \in \mathcal{P}(S)\}\). We say that a set \(S\) is closed under sorting if \(s^i \in S\) for all \(s \in S\).

Theorem 3.1. [8, 9] If \(S\) is nonnegative and closed under sorting, then we can describe the set \(\mathcal{P}(S)\) and \(\mathcal{P}_\pm(S)\) with \(O(n \log n)\) additional variables and linear inequalities given a representation of \(S\).

Construction behind Theorem 3.1. Sorting networks are the key ingredient here. They can be viewed as a comparison-based sorting algorithm that a priori decides on all comparisons to make, as opposed to traditional ones which choose comparisons adaptively. A classic result by Ajtai et al. [1] shows that \(O(n \log n)\) comparators suffice, though more practical sorting networks with simple recursive constructions require \(O(n \log^2 n)\) comparators Batcher [3].

Let us first describe how to model \(\mathcal{P}(\{v^i\})\) for any sorted vector \(v^i \in \mathbb{R}^n\). Suppose we have a sorting network on \(n\) inputs with \(m\) comparators. We first introduce a set of nonconvex constraints for each comparator:

\[
\alpha_1^k + \alpha_2^k = \beta_1^k + \beta_2^k, \quad \beta_1^k = \max(\alpha_1^k, \alpha_2^k), \quad \beta_2^k = \min(\alpha_1^k, \alpha_2^k).
\]

Let \(x_i\) and \(z_i\), \(i = 1, 2, \ldots, n\) denote the \(x\) variables corresponding to the \(i\)th input or output to the entire sorting network, respectively. Setting \(z = v^*\), it is easy to see that \(x\) can now take on the value of any permutation of \(v\). We can convexify (9) to obtain

\[
\alpha_1^k + \alpha_2^k = \beta_1^k + \beta_2^k, \quad \beta_1^k \geq \alpha_1^k, \quad \beta_2^k \geq \alpha_2^k,
\]

and this gives us a construction for \(\mathcal{P}(\{v^i\})\).

The construction above more generally gives us \(\mathcal{P}(S)\) for any \(S\) closed under sorting, and obtaining \(\mathcal{P}_\pm(S)\) is straightforward. We simply need to introduce \(n\) variables and we have \(\mathcal{P}_\pm(S) = \{x : -s \leq x \leq s\}\) for some \(s \in \mathcal{P}(S)\).

Finding appropriate \(S\) for the norms. Theorem 3.1 reduces the problem to finding \(S\) that are closed under sorting where \(\mathcal{P}_\pm(S)\) gives us the correct cones. For the \(k\)-support norm, this is just

\[
\{x \in \mathbb{R}^n : \|x_1, \ldots, x_k\|_p \leq \lambda, x_{k+1}, \ldots, x_n = 0\}.
\]

The OWL norm requires a little more work, and we use a construction akin to the ones in the previous section. See figure 3 for the intuition.

Figure 3: OWL construction of \(S\).

Improving the bound to \(O(n \log k)\) for \(k\)-support norm. Instead of using a full sorting network, we consider a networks that is only guaranteed to sort \(k\)-sparse positive inputs (inputs that have only \(k\) positive entries and are zero everywhere else). We will term these as \(k\)-sparse sorting networks. (Since these do not sort every input, this is strictly speaking an abuse of the term ‘sorting network’.)

A \(k\)-sparse sorting network can be constructed by composing many smaller full sorting networks. Given a set of indices \(A \subseteq [n]\), we let \(\mathcal{SN}(A)\) denote a \(O(|A| \log |A|)\) sorting network over indices \(i \in A\) that sorts all inputs over just those indices and \(\mathcal{SN}_{k\text{-sparse}}(A)\) denote a \(k\)-sparse sorting network over those indices. We will now describe how to construct \(\mathcal{SN}_{k\text{-sparse}}(A)\) recursively.

For \(|A| \leq 2k\), we can use \(\mathcal{SN}(A)\) as \(\mathcal{SN}_{k\text{-sparse}}(A)\). As for \(A = [i, j]\) where \(|A| > 2k\), we can compose the following three sorting networks to get \(\mathcal{SN}_{k\text{-sparse}}(A)\): (1) \(\mathcal{SN}_{k\text{-sparse}}([i, \lceil \frac{i+j}{2} \rceil])\), \(\mathcal{SN}_{k\text{-sparse}}([\lfloor \frac{i+j}{2} \rfloor + 1, j])\), and \(\mathcal{SN}([i, i + k - 1] \cup [\lfloor \frac{i+j}{2} \rfloor + 1, \lceil \frac{i+j}{2} \rceil + k])\).
**Proposition 3.2.** There are $k$-sparse sorting networks with $O(n \log k)$ comparators.

If the vector $v^{\downarrow}$ is also $k$-sparse and positive, then the construction (10) guarantees that every permutation of $v$ can be obtained. So this ‘sorting network’ suffices if these are the atoms we want.

**Proposition 3.3.** The $k$-sparse sorting network gives us an extended formulation for $k$-support norms with a $\|\cdot\|_p \leq 1$ constraint and $O(n \log k)$ variables and linear constraints.

The bounds given in Propositions 3.2 and 3.3 assume that we use $O(n \log n)$-sized sorting networks. Using the more practical $O(n \log^2 n)$ sorting networks by Batcher [3], we obtain $O(n \log^2 k)$ instead.

### 4 CVX Implementation: Code and Observations

We tested the running times of the various formulations on a Intel Core i5-4200U (dual core) laptop with 4GB of RAM in MATLAB R2017a, CVX 2.1, and Gurobi 6.0. The main implementations we compared are (1) the full sorting network-based $k$-support formulation, and (2) the full sorting network-based OWL formulation. We use the $O(n \log^2 n)$ bitonic sorting network by Batcher [3] in both cases. We also briefly discuss the dynamic programming-based $k$-support formulation. Again, the implementation is available at [https://github.com/limconghan](https://github.com/limconghan).

The test problem we used was $\min_x (1/2)\|y - Ax\|^2 + \lambda \|x\|_p$, where $A$ is a $m \times n$ matrix with each entry picked uniformly at random from $[0, 1]$. We compare the running time (preprocessing + solve times) of the different methods as we scale $n$ up from 100 to 1000 and set $m = n/5$. We also discuss the case where $n$ is fixed while $m$ varies. We set $p = 2$ and fixed $k = 20$ in all cases.

The observations from these initial experiments are:

- Conic norm constraints (i.e. inequalities (5) and (14), which have an $\ell_p$ term) significantly increases the preprocessing time and this makes the larger dynamic programming formulations very slow.
- The dynamic programming $k$-support formulation takes about 200 seconds for $k = 20$, $n = 100$, $m = 20$, and about 500 for $k = 20$, $n = 200$, $m = 40$.
- The time to solve the programs are fairly similar (within a factor of 1.5) between the two sorting network formulations. The main difference is in the preprocessing time.
- As the number of rows in $A$ approaches $n$, the solve times for both formulations jumps up significantly while the processing time remains roughly similar.

### Acknowledgements

We would like to thank Stephen Wright for all the discussions and feedback. This work was supported by NSF award CMMI-1634597, ONR Award N00014-13-1-0129, and AFOSR Award FA9550-13-1-0138.
References


