Partitioning Problems for Distributed Optimization

Madeleine Udell

June 3, 2011

Abstract

The Alternating Directions Method of Multipliers (ADMM) ([BPC+11]) paradigm for distributed optimization assumes that the objective of an optimization problem splits into two additive components which are separately easy to optimize. Here, I generalize the ADMM paradigm to problems with an arbitrary number of additive components in the objective, and ask the question: how can we partition the optimization problem to promote speedy convergence?

1 Introduction

In this section I describe the basic ADMM iteration and some simple changes to the structure of the algorithm that reduce variable storage. The remainder of the paper is organized as follows: in section 2 I construct a theoretical framework for the partitioning problem in ADMM. Section 3 gives examples of problems in which this framework might be useful, and section 4 shows numerical results from a variety of partitioning schemes which validate the theoretical framework. I conclude in section 5 with a few recommendations, conjectures and directions for future research. Numerical details are relegated to the appendix.

1.1 Alternating Directions Method of Multipliers

The ADMM paradigm for distributed optimization is based on the premise that the objective function splits into two additive components, each of which is separately easy to optimize but the sum of which is difficult to optimize due to the presence of complicating variables shared by the two objectives. The canonical problem for ADMM is to solve

$$\text{minimize} \quad f(x) + g(x)$$

where $x \in \mathbf{R}^n$ and f and g are convex functions $\mathbf{R}^n \mapsto \mathbf{R}$. In ADMM, we duplicate the variable x to form a constrained problem.

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x = z \end{array}$$

The constrained problem is then solved via a primal-dual method. More specifically, we find the solution via the iteration

$$\begin{aligned} x^{(k+1)} &= \operatorname{argmin} f(x^{(k)}) + (\rho/2)\|x^{(k)} - z^{(k)} + u^{(k)}\|_2^2 \\ z^{(k+1)} &= \operatorname{argmin} g(z^{(k)}) + (\rho/2)\|z^{(k)} - x^{(k)} - u^{(k)}\|_2^2 \\ u^{(k+1)} &= u^{(k)} + x^{(k)} - z^{(k)} \end{aligned}$$

It is convenient to define the prox operator of f with parameter ρ to be $\operatorname{prox}_{f,\rho}(z) = \operatorname{argmin}_x f(x) + (\rho/2)\|x - z\|_2^2$. Thus the x and y updates are simply given by the prox of z with respect to f and g, respectively, and in fact the only interaction of the algorithm with the functions f and g is via their prox operator. This property is often numerically advantageous; for example, when f is convex, prox_f is strictly convex.

1.2 Generalizing ADMM via the Consensus Problem

One can generalize ADMM to optimize a function that consists of arbitrarily many additive components. The general form *consensus problem* is written as

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

With the objective function written in this form, we can simplify the ADMM iteration (see [BPC+11]). We rewrite the iteration as

$$\begin{aligned} x_i^{(k+1)} &= \operatorname{argmin} f(x_i^{(k)}) + (\rho/2)\|x_i^{(k)} - z^{(k)} + y_i^{(k)}\|_2^2 \\ z^{(k+1)} &= \frac{1}{N}\sum_{i=1}^{N} x_i^{(k+1)} \\ y_i^{(k+1)} &= y_i^{(k)} + x_i^{(k)} - z^{(k)} \end{aligned}$$

2 Optimizing ADMM

ADMM can be made to converge more quickly by choosing a wise partition of the terms in the objective function into subproblems.

2.1 Choosing a Partition

We have a choice in how to partition the M additive terms in the objective function into N subproblems when we formulate the consensus problem.

2.1.1 Optimal Partition Size

If our processors are very small, but we have a large number of them, we may choose to let $N = M$, creating M copies of the variables and computing the prox operator for each term in the sum independently. However, if our processors are slightly larger, it may be advantageous to set $N < M$, partitioning the terms in the objective function

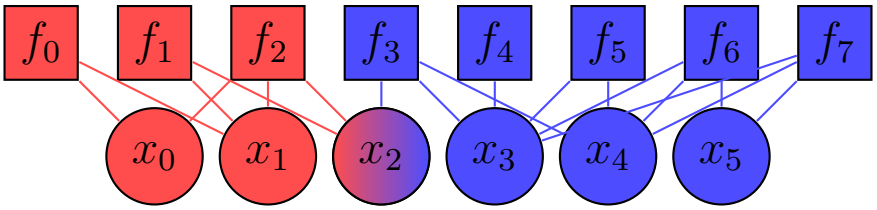


Figure 1: An ADMM schema. The two subproblems are represented as two different colors. The blue subproblem depends only on the blue variable nodes, while the red subproblem depends only on the red variable nodes. Note that only one variable node is shared between the two subproblems

into N subproblems. By so doing, we reduce the amount of space required to store the variables for our algorithm and reduce the number of iterations until convergence. However, we may increase the time for each iteration to complete.

In particular, we can consider problems in which each term in the objective depends only on a subset of the coordinates in the variable x, and in which the complexity of the x update scales nonlinearly in the number of variables included. This nonlinear scaling may come from the complexity of computing the prox function, in which case we expect it to scale as some polynomial in the number of variables, or from constraints on the size of the processor, in which case we expect a sharp increase as soon as we exceed the limitations of the processor.

If the complexity of the subproblem does scale nonlinearly in the number of variables, then when we allow the number of subproblems N to decrease from M, we may reduce the number of iterations required at the cost of a more difficult computation at each iteration. In these cases we expect to find a optimal size $N^*(M, C)$ where C is the capacity of a single processor.

2.2 Optimal Partition

Furthermore, we expect that some ways of assigning terms in the objective to processors may promote speedier convergence than others. When terms are assigned to the same subproblem, they can optimally trade off the value of the variables that they both depend on so as to maximize their sum. When terms are assigned to different subproblems, then the same wrangling over variable values is transmitted via the lagrange multiplier variables and must occur over the course of multiple iterations. Hence we may wish to place terms that depend strongly on the same variables in the same subproblem in order to optimize the convergence time of ADMM. On the other hand, if subproblems are connected but connected weakly (in a sense to be defined below), then convergence will be delayed, since these weak or distant links may take more time to propagate information.

In order to better understand the structure of this problem, we propose the following framework.

The duplication of variables in ADMM lends itself to an interpretation as a bipartite graph, with variables x_i on one side and function terms f_j on the other. We draw an

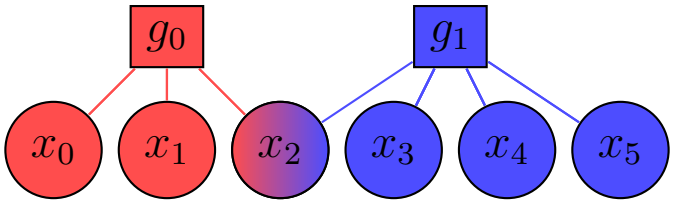


Figure 2: An ADMM schema. The two subproblems are represented as two different colors. The blue subproblem depends only on the blue variable nodes, while the red subproblem depends only on the red variable nodes. Note that only one variable node is shared between the two subproblems

edge from x_l to f_j if the term f_j depends on the variable x_k. An ADMM schema of size N for the problem is an assignment of the vertices $f_j, j = 1, \ldots, M$ into N subproblems $g_i, i = 1, \ldots, N$

$$g_i = \sum_{j=1}^{M} w_{ij} f_j$$

where $\sum_{i=1}^{N} w_{ij} = 1$ for every j and $w_{ij} \in [0, 1]$. The update of the local variable x_l for each subproblem is computed by $x_l = \text{prox}_{g_k,\rho}(z)$. We wish to minimize the convergence time of the ADMM algorithm over all choices of N and $\{w_{ij}\}$.

In what follows, we restrict our attention to the case $w_{ij} \in \{0, 1\}$, so that each function term is assigned to only one subproblem. In this case we can define the contracted graph corresponding to an ADMM schema by contracting each of the terms f_j for which $w_{ij} = 1$ into a single node g_i. Then each subproblem g_i depends only on the variables x_l with which it shares an edge.

A *strongly connected* schema is one in which many variables are connected to multiple subproblems, whereas a *weakly connected* schema is one in which the variable-subproblem graph is relatively sparse.

3 Examples

In order to determine the effect of weakly and strongly connected ADMM schema on the convergence rate of ADMM, we focus on two problems which allow us to tune the connection strength of the corresponding ADMM schemas.

3.1 Generalized Graph Laplacian

Our first example problem is a generalized graph laplacian.

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j)\in E} \|x_i - x_j\| \\ \text{subject to} & x \in \Omega \end{array}$$

Here, we interpret each x as a real number or vector located at the nodes of a graph, and wish to minimize the sum of the distances between the x vectors across edges in

the graph. We can consider the norm to be arbitrary, so long as the proximal operator can be easily computed. For example, if $x_i \in \mathbf{R}$, $\|\cdot\| = \|\cdot\|_2^2$, and $\Omega = \{x : \|x\|_2 = 1, \mathbf{1}^T x = 0\}$ then this problem computes the Fiedler value of the graph (the second smallest eigenvalue of the graph laplacian), which measures the connectedness of the vertices and controls the mixing time of a random walk on the graph ([Fie73],[Boy04]).

To compute the solution more easily, we can remove the norm constraint in Ω and replace it with a constraint on the values at particular nodes, which may be a reasonable gambit for, eg, a graph-embedding or graph-partitioning problem. We hypothesize that partitioning the nodes of the graph according to a minimal cut or multicut in the graph will lead to fastest convergence.

3.2 Model fitting

As another example, we can consider fitting a model to categorical data. We wish to solve

$$\text{maximize} \quad \sum_{i=1}^{m} f_i(x_i; \tilde{\lambda}_i, \mu) + g(\lambda, \mu)$$

Here, we interpret $f_i(x_i; \tilde{\lambda}_i, \mu)$ as the log probability of the datum x_i given the parameter vector λ, and assume that f_i depends on only a small number of the coordinates of λ. $g(\lambda)$ represents an optional regularization term. We denote by $\tilde{\lambda}_i$ the restriction of λ to the particular coordinates of λ that matter in f_i. The total dimension of λ is the number of categories d summed over all the k categorical variables. μ is to be interpreted as a complicating variable that links the terms. The graph structure of this optimization problem comes from the sparse dependence of the f_i on the components of the vector λ.

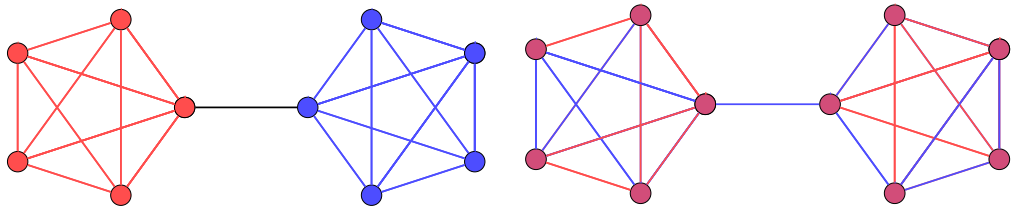
4 Results

4.1 Optimal Partition

To explore the effect of the choice of partition on the convergence of ADMM, I consider ADMM schemas for the generalized graph laplacian problem with minimal and maximal connectivity. In order to compute these minimal and maximal schemas, I restrict my attention to the barbell graph, which is defined as two Erdos-Renyi graphs connected to each other by a single edge. This graph is constructed by sampling the edges in two complete graphs with probability p, and connecting a single node in the first graph to a single node in the second.

In this case, the ADMM schema of size 2 with minimal connectivity (Figure 3a) is obtained by allowing each subproblem to include all the edges in one of the Erdos Renyi graphs. The last edge connecting the two graphs is placed arbitrarily into either subproblem. The ADMM schema with (approximately) maximal connectivity (Figure 3b) is obtained by placing each edge into each subproblem uniformly at random.

I experiment on a barbell graph with 200 vertices. I find that the minimum connectivity ADMM schema converges more quickly than the maximum connectivity schema (Figure 3). The minimum schema also has the added advantage that each subproblem



(a) Maximum Connectivity **(b)** Maximum Connectivity

relies on only half of the variables, so that the prox function can be computed much more quickly than would be possible in the general graph without explicit invocation of sparse matrix libraries.

4.2 Optimal Partition Size

To determine explore the dependence of the number of subproblems and their degree of connection on the convergence of the ADMM algorithm, I specialize the model fitting problem to the special case of the Lasso problem ([Tib96]).

$$\text{minimize} \quad \sum_{i=1}^{m} \|Ax - b\|_2^2 + \|x\|_1$$

The Lasso encourages sparsity in the solution to a least-squares problem by the addition of an L1 penalty term to the objective function. Each entry a_{ij} in the matrix A produces an edge between variable node i and term node j. If A is sparse, then the canonical bipartite graph associated with the problem is also sparse.

I experiment with the number of subproblems used to solve a LASSO problem with consensus ADMM. I choose rows in A and place them into each of the groups uniformly at random. Figure 4 gives results for $A \in \mathbf{R}^{200\times100}$, with an average of 3 and 15 nonzero entries per row, respectively.

We see that the iterations required for convergence increases with the number of subproblems in the ADMM schema. However, this increase is more modest when the problems are well-connected, as is the case when the matrix A is less sparse.

We conjecture that this difference may be understood in terms of the connectivity of the contracted bipartite graph corresponding to the different sparsity levels. On the Lasso problem with n variables for which the rows of A have on average d nonzeros per row, the expected degree of the contracted nodes for an ADMM schema with k groups chosen at random is $\tilde{d} = n(1 - (\frac{n-d}{n})^k)$. As d increases, the expected degree of the random bipartite graph increases, and so the graph becomes progressively more well connected. In a well connected graph, fewer iterations are required for the rows of A to share the information required to come to a mutually optimal solution. In future work we hope to derive a theoretical understanding of the tradeoff between the iteration complexity of finding an optimum in a densely connected graph and the iteration complexity of sharing information over a graph with high diameter.

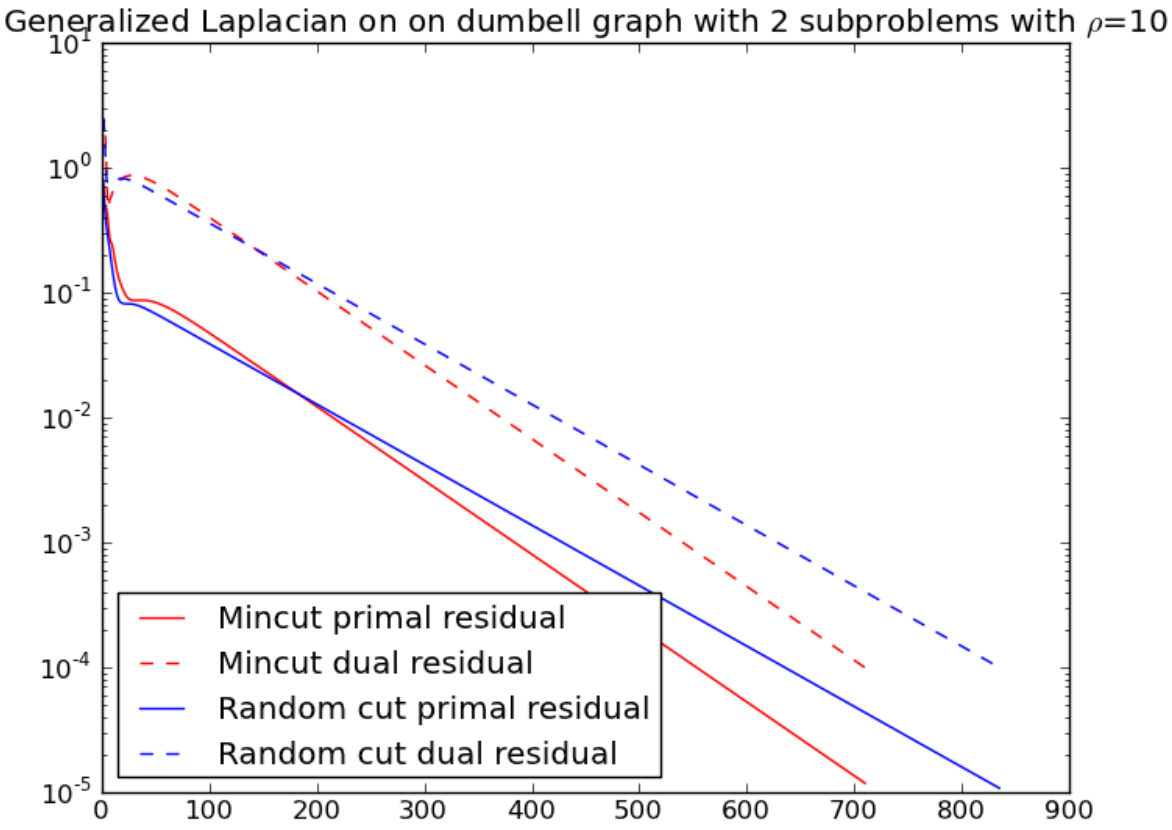


Figure 3: Convergence of ADMM for the generalized Laplacian problem on a dumbell graph. The minimal connectivity ADMM schema gives more rapid convergence.

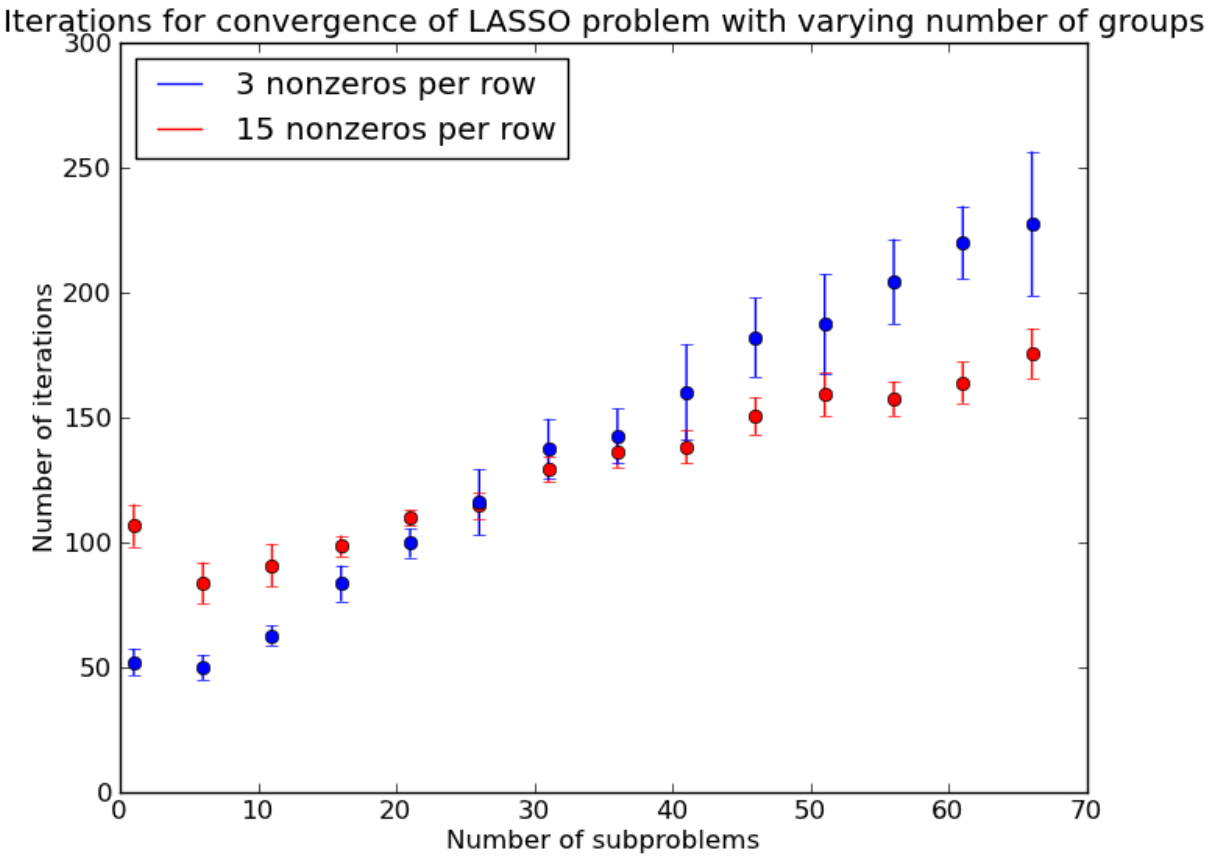


Figure 4: ADMM iterations until convergence for the Lasso problem increases with number of groups

5 Conclusion

The results of this project indicate that the performance of ADMM depends strongly on the selection of an appropriate ADMM schema. Here, I show evidence that

- When problems are highly structured, as in the case of the generalized laplacian on a barbell graph, an ADMM schema should be chosen that respects this structure. This choice has the added advantage of simplifying the size of the subproblems.
- The number of iterations required for convergence increases with the number of subproblems. Hence, one ought not to choose schema in which subproblems are arbitrarily small, but to select subproblem size so as to optimize the product of the number of iterations and the time complexity of each iteration.

5.1 Future Work

These preliminary results raise a number of interesting questions, and suggest a number of lines for future research.

- How does the number of iterations required for ADMM convergence depend on the graph structure of the problem? Can we find a theoretical dependence of the iteration complexity on the Fiedler value or the diameter of the contracted bipartite graph?
- How does the parameter ρ affect the results found in this paper? Are the results qualitatively the same when using an optimally tuned ρ?
- Can the result on optimal partitioning for the dumbell graph be generalized to other functions and other graph structures using a balanced mincut?

A Appendix: Numerical Considerations

A.1 Disciplined Variable Scoping

If f_i depends only on some subset of the coordinates $\Omega_i = \{l_1, \ldots, l_j\}$ with $j < n$, we need only record the values of x_i and y_i on the coordinates in Ω_i. In this case, denote by $\tilde{x}, \tilde{y} \in \mathbf{R}^k$ the reduced variables defined only on the coordinates in Ω, and denote by $\tilde{z}_i$ the linear function of z restricting of z to the coordinates in Ω_i. The $\tilde{x}$ and $\tilde{y}$ updates are then written simply by writing a $\sim$ above each variable in the iteration above.

To derive the new z update, consider the values of x_i and y_i in the original iteration on a coordinate $j \in \Omega^C$. We find that

$$\begin{aligned}(x_i^{(k+1)})_j &= 2(z^k)_j - (z^{k-1})_j \\ (y_i^{(k+1)})_j &= (z^k)_j - (z^{k+1})_j\end{aligned}$$

In the limit as $k \to \infty$, $(x_i^{(k+1)})_j \to (z^k)_j$ and $(y_i^{(k+1)})_j \to 0$, but in the meantime it is essential to consider how these trivial coordinates affect the iteration. In terms of the reduced variables, the z update is written as

$$(z^{(k+1)})_l = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{l \in \Omega_i, l_j = l} (\tilde{x_i}^{(k+1)})_j + \mathbf{1}_{l \in \Omega_i^C} (2(z^k)_j - (z^{k-1})_j)$$

This separation of variables leads to a very nice object-oriented approach to the consensus problem. Each subproblem should record the values of the variables that occur in its own objective function, and should have the ability to compute the prox operator corresponding to its objective function. The master problem need only keep a record of z at the current and previous iteration, and maintain the ability to add and subtract.

A.2 Python Interface

In order to test the effects of the ADMM schema graph on convergence, I designed a Python interfact to ADMM that explicitly relies on the problem graph. The ADMM routine is encoded in an object oriented framework. An ADMM problem consists of a set of subproblems, a global consensus variable z, and a record of the value of z at the previous iteration. Each subproblem consists of a prox function, a local primal variable x, a local lagrange multiplier variable y, and a maping Ω of coordinates in the local variable x onto the global variable z. In this framework, the ADMM iteration appears as follows:

while not problem.stop **do**
  for subproblem in problem **do**
    subproblem.y ← subproblem.y + subproblem.x - subproblem.z(problem.z)
    subproblem.x ← subproblem.prox(subproblem.z(problem.z) - subproblem.y)
  end for
  problem.z.prev ← problem.z
  problem.z ← mean(problem.x(subproblem.x))
end while

where for consistency with the iterates in the original ADMM iteration we set

$$\text{problem.x(subproblem.x)} = \begin{cases} \text{subproblem.x}_i & i \in \text{subproblem.}\Omega \\ 2\ \text{problem.z}_i - \text{problem.z.prev}_i & i \notin \text{subproblem.}\Omega \end{cases}$$

References

[Boy04] S Boyd. Fastest mixing Markov chain on a graph. *SIAM review*, 2004.

[BPC+11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 2011.

[Fie73] M Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 1973.

[Tib96] R Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, 1996.