Consensus Optimization with Automatic Variable Splitting

Tri Dao
trid@cs.stanford.edu

Yushi Wang
yushiw@cs.stanford.edu

June 5, 2015

1 Introduction

By solving problems using a divide-and-conquer technique, distributed optimization has been useful for a variety of applications requiring large datasets or limited information between cooperating parties. In such situations, the scheme in which the problem is divided is crucial for its efficiency. For our project, we implemented and investigated a distributed optimization method called Consensus Optimization using Alternating Direction Method of Multipliers (ADMM).

In section 2, we describe the basic ADMM iteration, highlight simple changes to the structure of the algorithm that reduce variable storage, and construct a theoretical framework for the partitioning problem in ADMM. Section 3 describes our implementation of the solver and variable splitting method. Section 4 shows numerical results from a variety of partitioning schemes which validate the theoretical framework. We conclude in section 5 and 6 with a few recommendations, conjectures and directions for future research.

2 Background

2.1 Consensus optimization

When using distributed optimization algorithms, such as the ADMM algorithm, the optimization problem is assumed to split into separate additive components, or subproblems. These subproblems are only connected to each other via equality linear constraints. The general form of such a problem is called the consensus form:

$$
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{N} f_i(x_i) \\
\text{subject to} & \quad x_i - z = 0, \quad i = 1, \ldots, N.
\end{align*}
$$

Other constraints can be included as terms in the objective function using the indicator function. The distributed optimization algorithm minimizes each subproblem separately, making it suitable for distributed systems. In the ADMM algorithm [BPC+11], we have the update per iteration (for $i = 1, \ldots, N$):

$$
\begin{align*}
x_i^{(k+1)} &:= \arg\min_{x_i} \left( f_i(x_i) + y_i^k(x_i - z^{(k)}) + (\rho/2)\|x_i - z^{(k)}\|^2 \right) \\
z^{(k+1)} &:= \frac{1}{N} \sum_{i=1}^{N} (x_i^{(k+1)} + (1/\rho)y_i^k) \\
y_i^{(k+1)} &:= y_i^k + \rho(x_i^{(k+1)} - z^{(k+1)}).
\end{align*}
$$
In each iteration, each subsystem computes $x_i^{(k+1)}$ separately by solving a local optimization problem, with parameter $\rho$. They then send the optimal values of the variables to the master consensus solver to update the public variables $z^{k+1}$. The consensus solver then sends the value of $z^{(k+1)}$ to each subsystem to update its local variables.

### 2.2 Variable Splitting

The standard method of transforming a general convex problem into the consensus form is through variable replication, or variable splitting. For example, to minimize $f(x) + g(x)$, we introduce variable $z$ and constraint $z = x$ to reformulate the problem as: minimize $f(x) + g(z)$ subject to $x = z$.

In general, there is usually more than one way to split the variables. For problems that are highly structured, or for small problems, variable splitting can be done by hand. However, this is clearly not feasible for larger problems, which can have upwards of tens of thousands of variables. Therefore, we aim to develop an automatic process to transform a general convex problem into the consensus form in (1). In particular, we seek a simple automatic variable splitting scheme that minimizes the running time of the distributed optimization algorithm.

### 3 Implementation

#### 3.1 Variable splitting via spectral graph partitioning

In our preliminary investigation, the number of iterations required for convergence in ADMM increases with the number of complicating variables (the number of equality constraints), and the number of subproblems. Moreover, in a distributed setting, the bottleneck in each iteration is the time taken to minimize the largest subproblem. Therefore, our variable splitting scheme has the following objectives:

- Transform a general convex problem into the consensus form, with a specified number of subproblems (depending on the number of cores/workers).
- Create subproblems of roughly equal size (in terms of the number of variables).
- Minimize the number of linear equality constraints between subproblems.

We can minimize the number of new equality constraints added by expressing this as a graph partitioning problem. From the abstract syntax tree of the convex problem, we form a graph of components of the objective function and the complicating variables. We represent the components of the objective function in the convex optimization problem as vertices and the complicating variables (variables that appear in more than one components) as edges. The variable splitting problem turns into a graph partitioning problem: we partition this graph into $N$ pieces of roughly equal size, while keeping the number of edges cut small. Specifically, given the graph $G = (V, E)$, if we partition $V$ into disjoint nonempty subsets $U$ and $W$, we aim to minimize cost of cut $(U, W)/(|U| \cdot |W|)$.

In general, optimal graph partitioning is an NP-hard problem. However, there are good heuristics, one of which is based on spectral graph theory: we can approximate the sparsest cut of a graph (into two pieces) through the second eigenvalue of its Laplacian matrix, which can be computed efficiently (in $n^3$, where $n$ is the number of variables, which can be much smaller then the problem size). We apply this step recursively to get to the desired number of pieces/subproblems.

Given a simple graph $G = (V, E)$ with $n$ vertices $1, …, n$, its Laplacian $L = D - A$, where $D$ is the degree matrix $\text{diag}(\text{deg}(1), \ldots, \text{deg}(n))$ and $A$ is the adjacency matrix of the graph. The Laplacian has the quadratic form

$$x^T L x = \sum_{(i, j) \in E} (x_i - x_j)^2.$$
Thus \( L \) is a positive semidefinite matrix, and its smallest eigenvalue is 0, with the eigenvector \( v_1 = [1 \ldots 1]^T \). Assuming that \( G \) is connected, the eigenvector \( v_2 \) corresponding to the second smallest eigenvalue \( \lambda_2 \) of \( L \) will minimize the quadratic form, subject to the constraint that it is orthogonal to \( v_1 \). Thus vertices that are have edges between them will have similar vector entries, and so \( v_2 \) bisects the graph into two communities with comparable sizes while keeping the number of edges cut small [AS11]. We call this cut the spectral cut.

In our final implementation, we use METIS [KK95], an open-source graph partitioning program, to produce the spectral cut. This ensures robustness and efficiency of the splitting step.

3.2 Consensus solver

Our implementation of the solver is built atop the `Problem.py` module in CVXPY [DCB14]. All user-generated information (i.e. objective and list of constraints) is stored in a `Consensus` object, which inherits from CVXPY’s `Problem` class.

Once a `Consensus` object has been created, the user may either call `solve`, which invokes CVXPY’s solve function, or `consensus_solve`, which:

1. Splits the problem’s objective into a list of smaller constituents for the subproblems.
2. Splits the problem’s list of constraints into lists of subsets of constraints for each subproblem.
3. Performs consensus optimization and updates until convergence.

To perform the consensus updates, we use the `Pipe` and `Process` Python libraries to minimize each of the subproblems, send back values of shared variables to the master solver, who distribute the average values back from subproblems, as described in Section 2.1.

**Example code**  We provide example code for how to invoke our solver below. More documentation and test examples are available online.

```python
# Define variables
v1 = cvx.Variable(4)
v2 = cvx.Variable(4)

# Define objective function and constraints
obj = Minimize(cvx.norm(v1))
constraints = [v1 >= 0, v2 <= v1]

# Define and solve a problem
c = splitter.Consensus(obj, constraints, num_procs=2)
c.consensus_solve()

# Extract information (optimal values, # public variables, splits, etc.)
print c.objective.value
splits, public_vars, subproblems = c.split_info
```

4 Numerical Examples

To demonstrate the advantages and limitations of our solver system, we created and solved several numerical examples of optimization problems.
4.1 Sum of squares

In this example, we engineered a problem where there is a natural way to split the problem. In particular, we minimize a sum of squares, with variables $v_{ij}$:

$$\text{minimize} \quad \sum_{i=1}^{n} \left( \sum_{j} v_{ij} - b_i \right)^2.$$ 

For our example, we created $n = 10$ subproblems for the sum. The shared variables $v_{ij}$ are chosen so that 5 components in the objective share 10 variables, the remaining 5 share 10 other variables, and there is one variable that is shared between the two groups. We show the graph representing the problem (with vertices as components in the objective and edges as complicating variables) in Figure 1. The colored edges and nodes represent private variables and sum components, respectively, grouped by the split. Black edges represent variables that are shared between the two subproblems. Comparing the spectral cut produced by our variable splitting scheme, and a random cut, we can clearly see that the spectral cut results in much fewer shared variables.

![Spectral cut](image1.png) ![Random cut](image2.png)

Figure 1: Graphs representing sum of squares problem, with spectral cut and random cut.

We plot the objective function value versus iteration for the spectral cut and 5 different random cuts in Figure 2.

![Objective function values](image3.png)

Figure 2: Objective function values $f - f^*$ for the sum of squares problem.

We see that the consensus solver converges much faster with the spectral cut (within 20 iterations) than with a random cut (more than 100 iterations). However, we have to keep in mind that the problem was specifically constructed to have a natural cut, and so it is not surprising that the spectral cut yields much faster convergence.
4.2 Routing

We applied our solver to a standard flow control problem, a more natural example in which splitting is important for performance. In this flow control problem, we constructed a routing matrix \( R \in \mathcal{R}^{m \times n} \), where \( R_{ij} = 1 \{ \text{flow } j \text{ passes over link } i \} \), and a capacity vector \( c \in \mathcal{R}^m \). The utility optimization problem to be solved is:

\[
\begin{align*}
\text{maximize} & \quad U(f) = \sum_{j=1}^n U_j(f_j) \\
\text{subject to} & \quad Rf \leq c, \quad f \geq 0.
\end{align*}
\]

We take \( U \) to be a concave function, say \( U(x) = \sqrt{x} \).

We compare the plots of the objective function value versus iteration and versus time taken (in seconds) for the spectral cut and 5 different random cuts in Figure 3. Since this is a constrained problem, and each iteration of ADMM does not generally produce a feasible solution, the objective value might increase beyond \( f^* \). Thus, we plot the absolute value of \( f - f^* \) in order to get a better picture of the convergence. Note that the values \( |f - f^*| \) appear to fluctuate as the iterations alternate between feasible and infeasible solutions to the original problem.

![Figure 3: Objective function values \(|f - f^*|\) for the routing problem.](image)

We observe that not only does the spectral cut reduce the number of iterations required to converge, it also reduces the time taken per iteration to solve the subproblems.
4.3 SVM

One final prominent example that we tested our algorithm on is the standard linear classification problem. In this setting, we have \( n \) training examples and loss functions, one loss function per example. To solve this, we do unconstrained optimization on minimizing the sum over all losses with respect to the weight vector \( w \):

\[
\text{minimize}_w \sum_{i=1}^n \ell(x_i, y_i; w) = \sum_{i=1}^n \max \left\{ 0, 1 - y_i \cdot x_i^T w \right\}.
\]

We plot the objective function value versus iteration for the spectral cut and 3 different random cuts in Figure 4.

![Figure 4: Objective function values \(|f - f^*|\) for the SVM problem.](image)

We observe that unlike the other two examples, there are no advantages to using a spectral cut over any other splitting schema. The convergence rate with respect to both time and iterations elapsed appears to be approximately the same for all three plots. This is easily explained, as in the SVM setting, there is only one variable (the set of weights), which cannot be split efficiently using our framework. In fact, the graph representation of the problem is a complete graph. Thus, any variable splitting schema would pass the same global variable to each subsystem, and would not be able to save time or space.

5 Discussion

In terms of performance, our program is highly unoptimized, since we focused on simplicity of code design within the scope of this class. Nonetheless, we are able to solve most of our unconstrained minimization problems within an order of magnitude of CVXPY’s serial solver.

Our automatic splitting favors problems whose graph representations have “bottlenecks” (formally the graph has a high isoperimetric number, or Cheeger’s constant). Two benefits of splitting problems along bottlenecks:

- Reduce number of iterations.
- Reduce running time per iteration.

For problems with a single large variable, or whose graph representations is highly connected (e.g. SVM), our implementation of automatic variable splitting does not help with convergence, as shown.
6 Future work

Both authors will continue to work on this project over the summer. The next phase of the project consists of the following tasks:

- Adapting value of parameter $\rho$ dynamically. The consensus solver can it can change $\rho$ appropriately each iteration to drive either the primal or dual residual down quickly. This can help reduce the number of iterations required for convergence.

- Accounting for variable sizes in splitting. Our current implementation treat small and large variables equally. We will experiment with minimizing the total size of shared variables instead. The graph partitioning problem will then have edge cost, which is the size of the variable.

We are working with Steven Diamond, the author of CVXPY, to further integrate our consensus solver and variable splitting scheme into CVXPY. We hope that this project will make this convex modeling language more viable for larger convex problems that are better solved in a distributed setting.

Acknowledgement

We thank Professor Stephen Boyd for the project idea, mentorship, and his continued support and direction. We also thank Steven Diamond for his experienced guidance as the author of CVXPY and expert on ADMM.

References


