Distributed Interior-point Method for Loosely Coupled Problems
Sina Khoshfetrat Pakazad∗ Anders Hansson∗
Martin S. Andersen**

Abstract: In this paper, we put forth distributed algorithms for solving loosely coupled unconstrained and constrained optimization problems. Such problems are usually solved using algorithms that are based on a combination of decomposition and first order methods. These algorithms are commonly very slow and require many iterations to converge. In order to alleviate this issue, we propose algorithms that combine the Newton and interior-point methods with proximal splitting methods for solving such problems. Particularly, the algorithm for solving unconstrained loosely coupled problems, is based on Newton’s method and utilizes proximal splitting to distribute the computations for calculating the Newton step at each iteration. A combination of this algorithm and the interior-point method is then used to introduce a distributed algorithm for solving constrained loosely coupled problems. We also provide guidelines on how to implement the proposed methods efficiently, and briefly discuss the properties of the resulting solutions.

1. INTRODUCTION
Distributed optimization methods are specially crucial in the absence of a centralized computational unit or when this unit lacks the necessary computational power for solving the problem at all or in a timely manner. In either case, distributed algorithms enable us to solve the problem without the need for a central computational unit and through collaboration of a number of computational agents. Such methods also allow us to solve the problem by solving subproblems that are easier and possibly faster to solve. Distributed methods for solving optimization problems have been studied for many years and there are different approaches for devising such algorithms, see e.g., Bertsekas and Tsitsiklis (1997); Eckstein (1989); Boyd et al. (2011); Nedic and Ozdaglar (2009); Nedic et al. (2010).

One of the most commonly used approaches for devising distributed optimization algorithms is based on applying gradient/subgradient methods directly to the problem, see e.g., Nedic and Ozdaglar (2009); Nedic et al. (2010). The resulting algorithms from such approaches are usually simple and easy to implement, however, they are very sensitive to the scaling of the problem and suffer from very slow convergence, Bertsekas and Tsitsiklis (1997). Another approach for constructing distributed algorithms is based on decomposition techniques and proximal splitting methods. To this end, the optimization problem is first decomposed and then a splitting method of choice, e.g., the alternating direction method of multipliers (ADMM), is applied to the decomposed problem. This will define the computational routines that each agent should perform locally and also will set the communication/collaboration protocol among the agents, e.g., see Bertsekas and Tsitsiklis (1997); Eckstein (1989); Boyd et al. (2011); Combettes and Pesquet (2011). Even though the resulting algorithms using this approach tend to be more complicated than the previous type of algorithms, they enjoy a faster rate of convergence for certain classes of optimization problems, e.g., when the objective function of the equivalent unconstrained reformulation of the problem, has two terms and/or is strongly convex, Goldfarb et al. (2012); Goldstein et al. (2012). For more general classes of problems, however, for instance when the objective function has more than two terms of which several are non-smooth, e.g., indicator functions, they can perform very poorly or may even fail to converge, e.g., see Chen et al. (2013). Although there exist modifications to splitting methods that allow us to apply them to more general classes of optimization problems, the resulting algorithms can become overly complicated to implement (particularly distributedly), Goldfarb et al. (2012); Han and Yuan (2012); Hong and Luo (2012), or the local computations can still be considerable, e.g., they may require each agent to solve an equality/inequality constrained problem at each iteration, Summers and Lygeros (2012); Ohlsson et al. (2013).

In order to allay the aforementioned issues, there has been a recent interest in studying the possibility of using second order methods for designing distributed optimization algorithms, e.g., see Chu et al. (2011); Wei et al. (2013); Necoara and Suykens (2009). For instance, in Wei et al. (2013), the authors propose a distributed Newton method for solving a network utility maximization problem. The cost function for such problems comprise of a summation of several terms where each term depends on a single scalar...
variable. This structure allows the authors to employ a matrix splitting method which in turn enables them to compute the Newton directions distributedly. However, this method relies on the special structure in the considered problem and hence can only be used for problems with such structure. In Necoara and Suykens (2009) the authors propose a distributed optimization method based on an interior-point method. The introduced algorithm is obtained by first performing a Lagrangian decomposition of the problem and then solving the subproblems using interior-point methods, efficiently. However, in the proposed algorithm, the computational cost for solving the subproblems can still be considerable. The authors in Chu et al. (2011) propose a distributed Newton method for solving coupled unconstrained quadratic problems, which is used for anomaly detection in large populations. This distributed method is only applicable to unconstrained quadratic problems.

In this paper, we investigate the possibility of utilizing Newton’s and interior-point methods for designing distributed algorithms for solving loosely coupled optimization problems. Notice that this type of problems constitute a more general class of problems than the ones in the above mentioned papers. For this purpose, we first exploit the coupling in the problem using consistency constraints and use proximal splitting methods to compute the Newton directions in a distributed manner. This then enables us to propose distributed implementations of the Newton method, which can be used for solving unconstrained loosely coupled problems. In order to solve constrained loosely coupled problems, we also put forth a distributed interior-point method which relies on the proposed distributed Newton method. Furthermore, in contrast to the methods proposed in Nedic and Ozdaglar (2009), in all the proposed algorithms each of the agents is only required to solve an unconstrained or equality constrained quadratic program at each iteration.

Notation
We denote by \( \mathbb{R} \) the set of real scalars and by \( \mathbb{R}^{n \times m} \) the set of real \( n \times m \) matrices. The transpose of a matrix \( A \) is denoted by \( A^T \) and the column and null space of this matrix is denoted by \( C(A) \) and \( N(A) \), respectively. We denote the set of positive integers \( \{1,2,\ldots,p\} \) with \( \mathbb{N}_p \). Given a set \( J \subseteq \{1,2,\ldots,n\} \), the matrix \( E_J \in \mathbb{R}^{J \times n} \) is the 0-1 matrix that is obtained by deleting the rows indexed by \( \mathbb{N}_n \setminus J \) from an identity matrix of order \( n \), where \( |J| \) denotes the number of elements in set \( J \). This means that \( E_J x \) is a \( |J| \) dimensional vector with the components of \( x \) that correspond to the elements in \( J \), and we denote this vector with \( x_J \). With \( x^{1,k}_l \) we denote the \( l \)th element of vector \( x^k \) at the \( k \)th iteration. Also given vectors \( x^k \) for \( i = 1,\ldots,N \), the column vector \( (x^1,\ldots,x^N) \) is all of the given vectors stacked.

2. NEWTON’S METHOD FOR SOLVING EQUALITY CONSTRAINT PROBLEMS

Consider the following equality-constrained optimization problem

\[
\begin{align*}
\text{minimize} & \quad F(x) \\
\text{subj. to} & \quad Ax = b,
\end{align*}
\]

where \( F : \mathbb{R}^n \rightarrow \mathbb{R} \) is convex and twice differentiable, and \( A \in \mathbb{R}^{p \times n} \) with \( \text{rank}(A) = p < n \). This problem can be solved iteratively using Newton’s method. Boyd and Vandenberghe, 2004, Ch. 10, where at each iteration the variables are updated as \( x^{(l+1)} = x^{(l)} + \alpha^{(l)} \Delta x^{(l)} \), with \( \Delta x^{(l)} \) and \( \alpha^{(l)} \) denoting the so-called Newton direction and its corresponding step size, respectively. The Newton direction \( \Delta x^{(l)} \) is defined as the solution of

\[
\min_{\Delta x} \quad F(x^{(l)}) + \nabla F(x^{(l)})^T \Delta x + \frac{1}{2} \Delta x^T \nabla^2 F(x^{(l)}) \Delta x
\]

subject to \( A(x^{(l)} + \Delta x) = b \),

which constitutes a quadratic approximation of \( (1) \) at \( x^{(l)} \).

\[
\lambda^{(l)} = \sqrt{\langle \Delta x^{(l)} \rangle^T \nabla^2 F(x^{(l)}) \Delta x^{(l)}},
\]

and it provides an estimate of the sub-optimality of the iterates. Hence, Newton’s method terminates when the Newton decrement falls below a given threshold \( \epsilon \). This iterative scheme is summarized in Algorithm 1.

3. INTERIOR-POINT METHOD

The optimization problem in (1) can be extended to include inequality constraints as

\[
\begin{align*}
\text{minimize} & \quad F(x) \\
\text{subj. to} & \quad Ax = b, \quad g_i(x) \leq 0 \quad i = 1,\ldots,m,
\end{align*}
\]

where \( g_i : \mathbb{R}^n \rightarrow \mathbb{R} \) are all convex and twice differentiable. Also assume that the problem admits a strictly feasible solution, i.e., there exists \( x \) such that \( Ax = b \) and \( g_i(x) < 0 \) for all \( i = 1,\ldots,m \). This allows us to solve (4) using an interior-point method which requires solving a sequence of equality-constraint problems, defined as

\[
\begin{align*}
\text{minimize} & \quad t F(x) + \sum_{i=1}^{m} \phi_i(x) \\
\text{subj. to} & \quad Ax = b_i,
\end{align*}
\]

for increasing values of \( t > 0 \). The functions \( \phi_i(x) := -\log(-g_i(x)) \) are so-called logarithmic barrier functions.

\begin{algorithm}[H]
1: Given \( l = 0 \), \( \epsilon_{nt} > 0 \) and \( x^{(0)} \) such that \( Ax^{(0)} = b \)
2: \textbf{repeat}
3: \quad Compute \( \Delta x^{(l)} \) by solving (2)
4: \quad Compute \( \lambda^{(l)} \)
5: \quad if \( (\lambda^{(l)})^2 / 2 \leq \epsilon_{nt} \) then
6: \quad \quad \( x^* = x^{(l)} \)
7: \quad \textbf{end if}
8: \quad Terminate the iterations
9: \textbf{end if}
10: \( x^{(l+1)} = x^{(l)} + \alpha^{(l)} \Delta x^{(l)} \)
11: \( l = l + 1 \)
12: \textbf{until} iterations are terminated
\end{algorithm}
Algorithm 2 Interior-point Method
1: Given $q = 0$, $t(0) > 0$, $\mu > 1$, $\epsilon_p > 0$ and feasible $x(0)$
2: repeat
3: Compute $x^∗(t^p)$ by solving (5) using Alg. 1 starting at $x^*(0)$
4: $x^*(t^p + 1) = x^∗(t^p)$
5: if $m/t^p < \epsilon_p$ then
6: $x^∗ = x^*(t^p + 1)$
7: Terminate the iterations
8: end if
9: $t^{p+1} = \mu t^p$
10: $q = q + 1$
11: until iterations are terminated

which (since $\phi(x) \to \infty$ as $g_i(x) \to 0$) virtually create barriers that prevent $x$ from violating any of the inequality constraints in (4c). Notice that for every fixed value of $t$ the problem in (5) is precisely in the form of (1), and it can be solved using Newton’s method, as is described in Algorithm 1. After the Newton method has converged to a solution for a given $t$, $x^∗(t)$, then within the interior-point method procedure, we gradually increase $t$ and perform the same procedure again. This is done until $m/t$ is below a certain threshold. This procedure is summarized in Algorithm 2. Next, we define loosely coupled problems and describe how we can take advantage of distributed computations for solving such problems using algorithms 1 and 2.

4. UNCONSTRAINED LOOSELY COUPLED OPTIMIZATION PROBLEMS

4.1 A Definition
Consider the following unconstrained optimization problem

$$\text{minimize } F_1(x) + \cdots + F_N(x),$$

where the convex and twice differentiable functions $F_i$ for $i = 1, \ldots, N$, only depend on a small subset of the elements of the variable $x$. Particularly, let us denote the ordered set of indices of variables that appear in $F_i$ by $J_i$. We also denote the ordered set of indices of terms in the cost function that depend on $x_i$ by $I_i$, i.e., $I_i = \{k \mid i \in J_i\}$. We call an optimization problem loosely coupled if $|I_i| \ll N$ for all $i = 1, \ldots, n$. This problem can be rewritten as the following equality-constrained optimization problem

$$\text{minimize } f_i(s^1) + \cdots + f_N(s^N)$$

subject to $Ex = S$, where $S = (s^1, \ldots, s^N)$ and $E = [E_{J_1} \cdots E_{J_N}]^T$, with $E_{J_i}$ a 0-1 matrix that is obtained from an identity matrix of order $n$ by deleting the rows indexed by $N_i \setminus J_i$. We refer to the constraints in (7b) as consistency constraints. Also $f_i$s are lower dimensional descriptions of $F_i$s such that $F_i(x) = f_i(E_j x)$ for all $x \in \mathbb{R}^n$ and $i = 1, \ldots, N$. Notice that if we define $F(S, x) = f_1(s^1) + \cdots + f_N(s^N)$ and $A = [-I E]$, the problem in (7) is in the same form as the problem in (1). We can now form the quadratic approximation in (2) for this problem, as below

$$\begin{align*}
\text{minimize } & \sum_{i=1}^N f_i(s^{(i)}(l)) + \nabla f_i(s^{(i)}(l))^T \Delta s^i + \frac{1}{2}(\Delta s^i)^T \nabla^2 f_i(s^{(i)}(l)) \Delta s^i \\
\text{subject to } & E(s^{(i)}(l) + \Delta x) = S^{(l)} + \Delta S.
\end{align*}$$

Algorithm 3 ADMM
1: Given $k = 0$, $\rho > 0$, $\epsilon_{pri}, \epsilon_{dual} > 0$, $y^{(0)}$ and $v^{(0)} = 0$
2: repeat
3: $x^{(k+1)} = \text{prox}_{\frac{1}{\rho} T_1}(y^{(k)} + v^{(k)})$
4: $y^{(k+1)} = \text{prox}_{\frac{1}{\rho} T_2}(x^{(k+1)} - v^{(k)})$
5: $v^{(k+1)} = v^{(k)} + (y^{(k+1)} - x^{(k+1)})$
6: if $\|y^{(k+1)} - x^{(k+1)}\|^2 < \epsilon_{pri}$ & $\|y^{(k+1)} - y^{(k)}\|^2 < \epsilon_{dual}$ then
7: Terminate the algorithm
8: end if
9: $k = k + 1$
10: until algorithm is terminated

where $\Delta S = (\Delta s^1, \Delta s^2, \ldots, \Delta s^N)$. Assuming that $S^{(i)} \in C(E)$, (8b) only requires $\Delta S \in C(\bar{E})$. This problem can then be rewritten as the following unconstrained non-smooth optimization problem

$$\begin{align*}
\text{minimize } & \sum_{i=1}^N f_i(s^{(i)}(l)) + \nabla f_i(s^{(i)}(l))^T \Delta s^i + \frac{1}{2}(\Delta s^i)^T \nabla^2 f_i(s^{(i)}(l)) \Delta s^i + \bar{T}_C(\Delta S),
\end{align*}$$

where $\bar{T}_C$ is the indicator function for the column space of $\bar{E}$. Considering the imposed structure in the cost function of the problem in (9), it can be solved distributively using so-called proximal splitting, Combettes and Pesquet (2011); Boyd et al. (2011); Eckstein (1989), which is the subject of the next section.

4.2 Proximal Splitting Methods
In many applications we are faced with convex problems of the form

$$\text{minimize } T_1(x) + T_2(x)$$

where minimizing the joint problem is much harder than solving problems including only individual terms of the cost function, Combettes and Pesquet (2011); Boyd et al. (2011); Khoroshin Pakazad et al. (2013). Proximal splitting algorithms enable us to solve the problem in (10) by solving minimization problems that are based on individual terms in the cost function. This is done through the use of the so-called proximity operators of these terms which are defined as follows. Given a closed convex function $T$, the proximity operator for this function, $\text{prox}_T : \mathbb{R}^n \to \mathbb{R}$, is defined as the unique minimizer of

$$\text{minimize } T(y) + \frac{1}{2}||x - y||^2.$$
Boyd et al. (2011); Bertsekas and Tsitsiklis (1997). Proximal splitting algorithms and particularly ADMM can perform very well when applied to quadratic problems, Boley (2012), and as we will see later they can be implemented very efficiently. Next, we describe how ADMM enables us to distribute the computations for solving (7).

4.3 Distributed Computation of the Newton Direction

As was discussed in Section 4.1, the problems in (1) and (7) have the same structure, and hence we can solve (7) using the Newton method as described in Algorithm 1. Recall that in the Newton method, in order to compute the Newton direction at each iteration, we need to solve the corresponding quadratic approximation of the problem. As was also shown in Section 4.1, the quadratic approximation for the problem in (7), can be equivalently rewritten as in (9). Notice that (9) is in the same format as (10), where $T_1$ involves a summation of $N$ decoupled terms and $T_2$ corresponds to $T_c$. With these definitions for $T_1$ and $T_2$ we now apply ADMM to (9). The 3rd and 4th steps of Algorithm 3 can then be written as

$$
\Delta S^{(k+1)} = \text{prox}_{\frac{1}{\lambda}}(\Delta Y^{(k)} + v^{(k)})
$$

$$
= \arg \min_{\Delta s^1, \ldots, \Delta s^N} \sum_{i=1}^N f_i(\Delta s^i) + \frac{\rho}{2} \|\Delta y^{(k)} + v^{(k)} - \Delta s^i\|^2
$$

(11a)

$$
\Delta Y^{(k+1)} = \text{prox}_{\frac{e}{\lambda}}(\Delta Y^{(k)} + v^{(k)}) = \text{prox}_{\frac{e}{\lambda}}(\Delta S^{(k+1)} - v^{(k)})
$$

$$
= E(E^T E)^{-1}E^T(\Delta S^{(k+1)} - v^{(k)}) = E(E^T E)^{-1}E^T \Delta S^{(k+1)}
$$

(11b)

where $f_i(\Delta s^i) = f_i(s^i(0)) + \nabla f_i(s^i(0))^T \Delta s^i + \frac{1}{2} \langle \Delta s^i, T \Delta s^i \rangle$ and $v^{(k)} = v^{(k-1)} + (\Delta y^{(k)} - \Delta s^{(k)}), which is given by the 5th step of Algorithm 3. Notice that the last equality in (11b) holds since if $v^{(0)} = 0$, then $v^{(k)} \in \mathcal{N}(E(E^T E)^{-1}E^T)$ for all $k \geq 1$. As we see from (11a), the update for $\Delta S^{(k+1)}$ can be computed in a decentralized manner using $N$ computing agents, where at each iteration every agent $i$ calculates

$$
\Delta s^{(k+1)} = \arg \min_{\Delta s^i} f_i(\Delta s^i) + \frac{\rho}{2} \|\Delta y^{(k)} + v^{(k)} - \Delta s^i\|^2
$$

$$
= (\nabla^2 f_i(s^i(0)) + \rho I)^{-1} \left( \Delta y^{(k)} + v^{(k)} - \nabla f_i(s^i(0)) \right)
$$

(12)

where $\Delta y^{(k)}$ and $v^{(k)}$ are locally available to each agent. The update in (11b) can also be performed distributedly, which is explained as follows. Let $\Delta z^{(k+1)} = (E^T E)^{-1}E \Delta S^{(k+1)}$. The update rule in (11b) can then be rewritten as

$$
\Delta Y^{(k+1)} = \Delta y^{(k+1)} + \Delta y^{N(k+1)}
$$

$$
= E \Delta z^{(k+1)} = (\Delta z^{(k+1)}_J, \ldots, \Delta z^{(k+1)}_N).
$$

Notice that $E^T E = \text{diag}(\{T_J, \ldots, T_N\})$, and hence the update for each component, $j$, of $\Delta x$ can be expressed as

$$
\Delta x^{(k+1)} = \frac{1}{|I_j|} \sum_{\ell \in I_j} \left( E^T \Delta z^{(k+1)}_\ell \right)_j.
$$

As a result, for each agent $i$ to compute $\Delta y^{(k+1)} = \Delta z_J^{(k+1)}$, it needs to communicate to all agents in $\text{Ne}(i) = \{ j \mid J_i \cap J_j \neq \emptyset \}$ which are referred to as the neighbors of agent $i$. The ADMM-based Newton direction computation can then be summarized as in Algorithm 4.

**Algorithm 4** ADMM-Based Newton Direction Computation

1: Given $k = 0$, $\rho > 0$, $\epsilon_{\text{proj}}$, $\epsilon_{\text{dual}} > 0$, $\Delta Y^{(0)} \in \mathcal{C}(E)$ and $v^{(0)} = 0$
2: repeat
3: for $i = 1, 2, \ldots, N$ do
4: $\Delta s^i = (\nabla^2 f_i(s^i(0)) + \rho I)^{-1} \left( \rho(\Delta y^{(k)} + v^{(k)} - \nabla f_i(s^i(0))) \right)$
5: Communicate with all agents $j$ belonging to $\text{Ne}(i)$
6: for all $j \in I_j$ do
7: $\Delta z_J = \frac{1}{|I_j|} \sum_{\ell \in I_j} \left( E^T \Delta z^{(k+1)} \right)_j
8: end for
9: $\Delta y^{(k+1)} = \Delta s_J$
10: $v^{(k+1)} = v^{(k)} + (\Delta y^{(k+1)} - \Delta s^{(k+1)})$
11: Check whether $\|\Delta y^{(k+1)} - \Delta y^{(k)}\|^2 \leq \epsilon_{\text{dual}}$ and $\|\Delta s^{(k+1)} - \Delta s^{(k)}\|^2 \leq \epsilon_{\text{proj}}$
12: end for
13: if condition in step (10) satisfied for all $i = 1, \ldots, N$ then
14: $\Delta x_{\text{int}} = \Delta z^{(k+1)}$
15: $\Delta S_{\text{int}} = \Delta Y^{(k+1)}$
16: Terminate the algorithm
17: end if
18: $k = k + 1$
19: until algorithm is terminated

Notice that the termination condition of Algorithm 4 (based on the 11th and 13th steps of the algorithm) can be established distributedly, provided that all agents declare their status of convergence (step 11). Also observe that the satisfaction of this termination condition implies the satisfaction of the termination condition in Algorithm 3. There are other ways of establishing convergence of Algorithm 4 to a solution (and possibly more efficient, e.g., based on Iutzeler et al. (2012)). However, for the sake of brevity we abstain from discussing such methods in this paper.

**Remark 1.** In Algorithm 4, the computational effort at each iteration for each agent is dominated by the update of $\Delta s^{(k+1)}$ which requires factorizing the matrix $\nabla^2 f_i(s^i(0)) + \rho I$. Notice that in case $\rho$ is chosen to be constant for all $k$, this matrix will also be constant for all $k$. This means that each agent would only need to compute this factorization once at the first iteration of the algorithm and use the precomputed factorization in the remaining iterations. This pre-caching of the factorization significantly reduces the overall computational cost of computing the Newton direction using Algorithm 4. Notice that even if the penalty parameter is not chosen to be constant, it is still possible to utilize the method presented in (Liu et al., 2013, sec. 4.2), to use pre-caching of factorizations to achieve similar computational efficiency.

5. **NEWTON’S METHOD WITH DISTRIBUTED STEP COMPUTATION**

We can now combine Algorithm 4 with the Newton method expressed in Algorithm 1, to provide a distributed computational scheme for the problem in (6). This is presented in Algorithm 5. Similar to Algorithm 4, the termination condition for the Newton iterations in Algorithm 5 can also be checked in a distributed manner with limited communications among the agents. To this end, each agent needs to compute its local Newton decrement, $\lambda^{(k)}$, and declare its local convergence status based on this.
Algorithm 5 ADMM-based Newton Method
1: Given \( l = 0, \epsilon_t > 0, S^{(0)}, \Delta S^{(l-1)}_{\text{nt}} \in \mathcal{C}(\mathcal{E}) \)
2: \textbf{repeat}
3: \hspace{2.5em} Compute \( \Delta S^{(l)}_{\text{nt}} \) using Algorithm 4 with starting point \( \Delta Y^{(0)} = \Delta S^{(l-1)}_{\text{nt}} \)
4: \hspace{2.5em} for \( i = 1, 2, \ldots, N \) \textbf{do}
5: \hspace{3.5em} Compute \( (X^{(l)}_{ki})^2 = (\Delta s^i_{\text{nt}})^2/2 \) \textbf{end for}
6: \hspace{2.5em} Compute whether \( (X^{(l)}_{ki})^2/2 \leq \epsilon_t/N \)
7: \hspace{2.5em} 
8: \hspace{2.5em} if condition in step (6) satisfied for all \( i = 1, \ldots, N \) then
9: \hspace{3.5em} \( S^* = S^{(l)} \)
10: \hspace{2.5em} \textbf{end if}
11: \hspace{2.5em} Terminate the algorithm
12: \hspace{2.5em} Compute the step size \( \alpha^{(l)} \) using line search
13: \hspace{2.5em} for \( i = 1, 2, \ldots, N \) \textbf{do}
14: \hspace{3.5em} \( s^i(l+1) = s^i(l) + \alpha^{(l)} \Delta s^i_{\text{nt}} \)
15: \hspace{2.5em} \textbf{end for}
16: \hspace{2.5em} \( l = l + 1 \)
17: \textbf{until} algorithm is terminated

Algorithm 6 ADMM-based Interior-point Method
1: Given \( q = 0, t^{(0)}(\theta) > 0, \mu > 1, \tau > 0 \) and feasible \( S^{(0)} \)
2: \textbf{repeat}
3: \hspace{2.5em} Compute \( S^* = (t^{(0)}(\theta) \text{ by solving (5) using Alg. 5 starting at } S^{(0)} \)
4: \hspace{2.5em} \( S^{(s+1)} = S^* = S^{(q)} \)
5: \hspace{2.5em} if \( m_l t^{(q)} < \epsilon_p \) then
6: \hspace{3.5em} \( S^* = S^{(s+1)} \)
7: \hspace{2.5em} \textbf{end if}
8: \hspace{2.5em} \( l^{(q+1)} = \mu t^{(q)} \)
9: \hspace{2.5em} \( q = q + 1 \)
10: \hspace{2.5em} \textbf{until} iterations are terminated

where the functions \( f_i \) and \( g_i \) are defined similarly as in Section 4.1 for the problem in (7). Notice that this problem is in the same format as (4) and can be solved using the interior-point method, which accordingly requires solving

\[
\min_{{s,x}} \sum_{i=1}^{N} f_i(s^i) + \sum_{i=1}^{N} g_i(s^i) \quad \text{subject to} \quad Ex = S
\]

for a sequence of increasing \( t \). For every given \( t \) then the problem in (17) can be solved using Algorithm 5, with a modification to step 3 of the algorithm. Particularly, the 4th step of Algorithm 4 (that is used in step 3 of Algorithm 5) needs to be modified as

\[
\Delta s^{i(k+1)} = (\nabla^2 h_i(s^i(k)) + \rho)\Delta s^{i(k)} - \nabla h_i(s^i(k))
\]

where \( h_i(s^i(k)) = f_i(s^i) + \frac{1}{g_i(s^i)}\nabla g_i(s^i)\nabla g_i(s^i)^T \) for all \( i \in \mathcal{E} \). Notice that after this modification remarks 1 and 2 still apply. The distributed scheme for solving (15) can then be obtained by combining the modified Algorithm 5 with Algorithm 2. This is summarized in Algorithm 6. The problem in (15) can be further extended by adding equality constraints. Assuming that the equality constraints also enjoy similar type of coupling as above, the problem can be written as

\[
\min_{{s,x}} \sum_{i=1}^{N} f_i(s^i) + \sum_{i=1}^{N} g_i(s^i) \quad \text{subject to} \quad Ex = S
\]

where \( A^i \in \mathbb{R}^{p_i \times |J_i|} \) and \( \text{rank}(A^i) = p_i < |J_i| \). This problem can also be solved using Algorithm 6 where in its 3rd step, Algorithm 5 is applied to

\[
\min_{{s,x}} \sum_{i=1}^{N} f_i(s^i) + \sum_{i=1}^{N} g_i(s^i) \quad \text{subject to} \quad A^i x = b^i \quad i = 1, \ldots, N
\]

This in turn requires another modification to Algorithm 4, that is used in the 3rd step of Algorithm 5. Specifically, the 4th step of Algorithm 4 should be changed to

\[
\Delta s^i(k+1) = \arg \min_{{A^i s^i = b^i}} \left\{ \nabla h_i(s^i)^T \Delta s^i + \frac{1}{2} (\Delta s^i)^T \nabla^2 h_i(s^i) \Delta s^i \right\}
\]

\[
\min_{{s,x}} \sum_{i=1}^{N} f_i(s^i) + \sum_{i=1}^{N} g_i(s^i) \quad \text{subject to} \quad Ex = S
\]

This solving problem is equivalent to solving the following linear system of equations.
\[
\begin{align*}
\nabla^2 h_i(s^{(l)}) + \rho I (A^i)^T \begin{bmatrix}
\Delta s^i \\
u
\end{bmatrix} &= \rho(\Delta y_i^{(k)} + v_i^{(k)}) - \nabla h_i(s^{(l)}) \\
0
\end{bmatrix}. \quad (20)
\end{align*}
\]

The computational cost for this is dominated by the cost for calculating the factorization of the coefficient matrix. Notice that, similar to (12), this coefficient matrix is also constant within the ADMM iterations and hence, the comments made in Remark 1 still apply. Even if \( \rho \) is varying similar techniques as in Liu et al. (2013) can be applied.

**Remark 3.** Notice that the resulting solution from Algorithm 6, \( S^* \), satisfies the local constraints, i.e., \( g_i(s^{(*)}) \leq 0 \) and \( A^i s^{(*)} = b^i \) for all \( i = 1, \ldots, N \). However, \( S^* \) does not necessarily satisfy the consistency constraints. In fact it is possible to upper-bound the consistency error in \( S^* \) as follows. Assume that \( S^{(0)} \) is both locally feasible and consistent. Then the consistency error of \( S^{(1)} \) can be computed as

\[
e_{c}(1) = \|S^{(1)} - E(E^T E)^{-1} E^T S^{(1)}\|^2 = \sum_{j=0}^{\text{iter}} (\alpha(j-1)^2 \|\Delta S^{j-1} - E(E^T E)^{-1} E^T \Delta S^{j-1}\|^2)
\]

\[
\leq \sum_{j=0}^{\text{iter}} (\alpha(j-1)^2 \epsilon_{\text{pri}})
\]

where \( \Delta S^{(1)} \) and \( \alpha(j-1) \) denote the Newton direction step size used in the \( j \)th Newton iteration in the 1st iteration of the interior-point method, respectively. Also \( \text{iter} \) denotes the number of required Newton iterations to converge to \( S^{(1)} \). Now assuming that Algorithm 6 requires \( q_{\text{max}} \) iterations to converge, we can bound the consistency error of \( S^* \) as \( e_c(q_{\text{max}}) \leq \sum_{q=1}^{\text{iter}} \sum_{j=0}^{q_{\text{max}}} (\alpha(j,q)^2 \epsilon_{\text{pri}})
\]

**Remark 4.** The proposed methods in this paper are closely related to inexact Newton methods, where the Newton direction is computed in a distributed manner. This indicates that the methods described in this paper, can potentially suffer from slow convergence when the iterates are very close to the optimal solution. The similarities between the two classes of methods can also give us insight on the convergence properties of the proposed methods and how to avoid slow convergence near optimal solution.

7. CONCLUSIONS

In this paper, we proposed distributed optimization algorithms for solving unconstrained and constrained loosely coupled optimization problems. These algorithms are based on the Newton and interior-point methods where we used the inherent structure in the problem to distribute the required computations within these methods. Particularly, by exploiting the coupling in the problem and using proximal splitting methods we showed how the Newton direction can be computed in a distributed manner. Notice that the proposed methods in this paper, all require a feasible starting point. As a future research directions, we intend to extend the proposed algorithms to also accommodate infeasible starting points. Also we intend to investigate primal-dual interior-point methods.

REFERENCES


